(ESTD-1995)

RGM COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous)
Approved by AICTE, New Delhi.
Accredited by NAAC with A+ Grade.
Affiliated to J.N.T.University, Ananthapuram.
Nandyal – 518501. Kurnool (dist.), A.P.

YEAR/SEMESTER: II/I **REGULATIONS: R-19**

PYTHON PROGRAMMING (A0503193)

COURSE MATERIAL

PREPARED BY:

Mr. P. PRATHAP NAIDU

ASSISTANT PROFESSOR

DEPARTMENT OF CSE

RGMCET (Autonomous)

NANDYAL - 518501

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

R G M COLLEGE OF ENGINEERING & TECHNOLOGY, NANDYAL
AUTONOMOUS
COMPUTER SCIENCE ENGINEERING

	T	C
II B.Tech. I-Sem (CSE)	**T**	**C**
	2+1	3

PYTHON PROGRAMMING (A0503193)
(Common to all Branches)

COURSE OBJECTIVES: This course will enable students to:
- ❖ Learn Syntax and Semantics of various Operators used in Python.
- ❖ Understand about Various Input, Output and Control flow statements of Python.
- ❖ Handle Strings and Files in Python.
- ❖ Understand Lists, Tuples in Python.
- ❖ Understand Sets, Dictionaries in Python.
- ❖ Understand Functions, Modules and Regular Expressions in Python.

COURSE OUTCOMES: The students should be able to:
- ❖ Examine Python syntax and semantics and be fluent in the use of various Operators of Python.
- ❖ Make use of flow control statements and Input / Output functions of Python.
- ❖ Demonstrate proficiency in handling Strings and File Systems.
- ❖ Create, run and manipulate Python Programs using core data structures like Lists and Tuples.
- ❖ Apply the core data structures like Sets and Dictionaries in Python Programming.
- ❖ Demonstrate the use of functions, modules and Regular Expressions in Python.

MAPPING OF COs & POs

CO/ PO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	3								1				1	1	1
CO2	2	3							1				1	1	1
CO3	1		2						1				1	1	1
CO4	2		2						1				1	1	1
CO5	2		2						1				1	1	1
CO6	2		2						1				1	1	1

UNIT – I:
Introduction: History of Python, Need of Python Programming, Applications Basics of Python Programming Using the REPL(Shell), Running Python Scripts, Variables, Assignment, Keywords, Input-Output, Indentation. Overview on data types: Numbers, Strings, Lists, Set, Tuple and Dictionaries.
Operators in Python: Arithmetic Operators, Comparison (Relational) Operators, Assignment Operators, Logical Operators, Bitwise Operators, Shift Operators, Ternary operator, Membership Operators, Identity Operators, Expressions and order of evaluations. Illustrative examples on all the above operators.

UNIT – II:
Input and Output statements: input() function, reading multiple values from the keyboard in a single line, print() function, 'sep' and 'end' attributes, Printing formatted string, replacement operator ({}). Illustrative examples on all the above topics.
Control flow statements: Conditional statements – if, if-else and if-elif-else statements. Iterative statements – for, while. Transfer statements – break, continue and pass. Illustrative examples on all the above topics.

UNIT – III:
Strings: Introduction to strings, Defining and Accessing strings, **Operations on string** - String slicing, Mathematical Operators for String, Membership operators on string, Removing spaces from the string, Finding Substrings, Counting substring in the given String, Replacing a string with another string, Splitting of Strings, Joining of Strings, Changing case of a String, Checking starting and ending part of the string, checking type of characters present in a string. Illustrative examples on all the above topics.
Files: Opening files, Text files and lines, Reading files, Searching through a file, Using try, except and open, Writing files, debugging.

UNIT – IV:
Lists: Creation of list objects, Accessing and traversing the elements of list. **Important functions of list** – len(), count(), index(), append(), insert(), extend(), remove(), pop(), reverse() and sort(). **Basic Operations on list:** Aliasing and Cloning of List objects, Mathematical Operators for list objects, Comparing list objects, Membership operators on list, Nested Lists, List Comprehensions. Illustrative examples on all the above topics.
Tuples: Creation of Tuple objects, Accessing elements of tuple, Mathematical operators for tuple, Important functions of Tuple – len(),count(),index(), sorted(), min(), max(), cmp().Tuple Packing and Unpacking. Illustrative examples on all the above topics.

UNIT – V:
Sets: Creation of set objects, Accessing the elements of set. Important functions of set – add(), update(), copy(), pop(),remove(),discard(),clear(). Basic Operations on set **-** Mathematical Operators for set objects, Membership operators on list, Set Comprehensions. Illustrative examples on all the above topics.
Dictionaries: Creation of Dictionary objects, Accessing elements of dictionary, Basic operations on Dictionary - Updating the Dictionary, Deleting the elements from Dictionary. Important functions of Dictionary – dict(), len(), clear(), get(), pop(), popitem(), keys(), values(), items(), copy(), setdefault(). Illustrative examples on all the above topics.

UNIT – VI:
Functions - Defining Functions, Calling Functions, Types of Arguments - Keyword Arguments, Default Arguments, Variable-length arguments, Anonymous Functions, Fruitful functions (Function Returning Values), Scope of the Variables in a Function - Global and Local Variables. Recursive functions, Illustrative examples on all the above topics**.**
Modules: Creating modules, **import** statement, from Import statement.
Regular Expressions: Character matching in regular expressions, Extracting data using regular expressions, Combining searching and extracting, Escape character.

TEXT BOOKS

1) Python for Everybody: Exploring Data Using Python 3, 2017 Dr. Charles R. Severance

REFERENCE BOOKS

1) Think Python, 2 Edition, 2017 Allen Downey, Green Tea Press
2) Core Python Programming, 2016 W.Chun, Pearson.
3) Introduction to Python, 2015 Kenneth A. Lambert, Cengages
4) https://www.w3schools.com/python/python_reference.asp
1) https://www.python.org/doc/

UNIT - 1

Python Language Fundamentals

Topics Covered:

- Introduction
- Application areas of Python
- Features of Python
- Limitations of Python
- Flavours of Python
- Python Versions
- Identifiers
- Reserved Words
- Datatypes
- Typecasting

NOTE: If you really strong in the basics, then remaining things will become so easy

L1: What is Pyhton?

- It is a **Programming Language.**

  ```
  - We can develop applications by using this programming language.
  ```

- We can say that, Python is a High-Level Programming Language. Immediately you may get doubt that, What is the meaning of **High-Level Programming Language**.

  ```
  - High-Level means Programmer friendly Programming Language. This means we are not
  required to worry about Low-level     things [i.e., Memory management, security, d
  estroying the objects and so on.]


  - By simply seeing the code programmer can understand the program. He can write th
  e code vey easily.


  - High level languages are Programmer friendly languages but not machine friendly
   languages.
  ```

Let's take the following example,

Let s take the following example,

a = 10

b = 20

c = 30 if a>b else 40

print(c)

Do you Know, if we can take this code, are in a position to understand this code?

You are not required to have any programming knowledge.

By observing the code you can say that, the value stored in C is **40**

If you have the kid, can you please show this 4 lines code and ask what is the ouput, your kid is able to answer without having any hesitation. This type of thing is called as High level programming.

Examples of High Level Programming Languages :

- C
- C++
- Java
- C#
- Python

You may get doubt that, you are writing just 4 lines code, is it really a Python code? Is it going to Work?

Let's execute and see what happens,

In [1]:

```python
a = 10

b = 20

c = 30 if a>b else 40

print(c)
```

```
40
```

It is perfectly Python code. This is called High level programming.

Python is a General Purpose High Level Programming Language.

Here, **General Purpose** means Python is not specific to a particular area, happily we can use Python for any type of application areas. For example,

- Desktop Applications
- Web Applications

- Data Science Applications
- Machine learning applications and so on.

Everywhere you can use **Python.**

L2: Who Developed Pyhton?

Who provides food for most of the programmers across the worldwide.

Guido Van Rossum developed Python language while working in National Research Institute (NRI) in Netherland.

When he developed this Language?

Most of the people may think that so far we are heared about Java, C and C++ and we are recently knowing about python (especially in our INDIA) and they may assume that Python is a new programming language and Java is a old programming language.

- Java came in 1995 and officially released in 1996.
- Python came in 1989, this means that Python is Older programming language tha Java. Even it is developed in 1989, but it is not released to the public immediately.
- In 1991, Pyhton made available to the public. Officially Python rleased into the market on 21-02-1991 (i.e., First version).

Then, Immediately you may have a doubt that, **Why Python suddenly (in 2019) became Popular**?

- Generally Market rquirements are keep on changing from time to time.
- Current market situation is, every one talks about

```
  - I need Simple Language (i.e., Easy to understandable)

  - I have to write very less (or) concise  code to fulfill my requirement.

  - In these days, everyone talks about AI, Machine Learning, Deep Learning, Neural
networks, Data Science, IOT.For these trending requirements, best suitable program
ming language is Python.
```

That's why in these days, Python becomes more popular programming language.

For example, Suppose you have a diamond. When there is a value for that diamond is, if the market requirement is good for that, then automatically this diamond value grows.

Date: 10-04-2020 - Day 2

L3. Easyness of Python compared to other programming languages

If you want to learn Python programming, what is the prerequisite knowledge required?

The answer for the above Question is:

Nothing is required, If you are in a position to read English statements, that is enough to learn Pyhton programming.

Eg:

If you are learning any programming language, the first application which we discuss is **Hello World** application.

In 'C'

1) #include <stdio.h>

2) void main()

3) {

4) printf("Hello World");

5) }

In 'Java'

1) public class HelloWorld

2) {

3) public static void main(String[] args)

4) {

5) System.out.println("Hello world");

6) }

7) }

In 'Python

In [1]:

```
print("Hello World");
```

```
Hello World
```

In [3]:

```
print("Hello World")          # ';' is also optional
```

```
Hello World
```

Just to print 'Hello World',

C language takes 5 lines of code

Java takes 7 lines of code

But, Python takes only one line of code.

When compared with any other prgramming language (C, C++, C## or Java), the easiest programming langague is **Python**.

Let us take another example,

To print the sum of Two numbers

Java

1) public class Add

2) {

3) public static void main(String[] args)

4) {

5) int a,b;

6) a =10;

7) b=20;

8) System.out.println("The Sum:"+(a+b));

9) }

10) }

C

1) #include <stdio.h>

2) void main()

3) {

4) int a,b;

5) a =10;

6) b=20;

7) printf("The Sum:%d",(a+b));

8) }

Python

In [6]:

```
a=10
b=20
print("The Sum:",(a+b))    # 3 line of code
```

```
The Sum: 30
```

Even we can combine first two lines of the above code into a single line

In [8]:

```
a,b=10,20
print("The Sum: ",a+b)    # 2 linesof code
```

```
The Sum:  30
```

This is the biggest advantage of python programming. We can do many things with very less code.

a=10

b=20

print("The Sum:",(a+b))

By seeing the above code, you may get one doubt that, In C & Java we declared variables 'a' and 'b' as **int** type. But in Python we didn't declare 'a' and 'b' types. You may ask that, in Python we need not to declare the type of the variables.

In Python, type concept is applicable (int,float .. types are there in Python), but **we are not required to declare type explicitly.**

In Python, whenever we are assigning some value to a variable, based on the provided value, automatically type will be considered. Such type of programming languages are known as **Dynamically Typed Programming Languages**.

In [9]:

```
a = 10
print(type(a))b
```

```
<class 'int'>
```

In [11]:

```
a = 10.5
print(type(a))
```

```
<class 'float'>
```

In [13]:

```
a = True
print(type(a))
```

```
<class 'bool'>
```

In [15]:

```
a = "Karthi"
print(type(a))
```

```
<class 'str'>
```

In python, same variable can be used with mutiple types of data.

In [17]:

```
a = 10
print(type(a))
a = 10.5
print(type(a))
a="Karthi"
print(type(a))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

Examples of Dynamically Typed Programming Languages:

- Python
- JavaScript etc.,

Examples of Statically Typed Programming Languages:

- C
- C++
- Java etc.,

There is no prerequisite for learning Python programming. So, **Python is recommended as first programming language for beginners.**

Key points

1. Python is a general purpose high level programming language.

2. Python was developed by Guido Van Rossam in 1989, while working at National Research Institute at Netherlands.

3. Officially Python was made available to public in 1991. The official Date of Birth for Python is : Feb 20th 1991.

4. Python is recommended as first programming language for beginners.

5. Python is an Example of Dynamically typed programming language

Why the name 'Python'

Why Guido Van Rossum selected the name **Python**?

- If you are a fan of any hero or heroine or political leader, generally you are trying to use same name for your passwords or user ids. It's a very common practice.

- In The same, Guido Van Rossum also very much impressed with one fun show, **The Complete Monty Python's Flying Circus**, which was broadcasted in BBC from 1969 to 1974. From this show name, he selected the word **Python** to his programming language.

L4. Python as All Rounder

C --> Procedural/Functional Programming Language

C++, Java --> Object Oriented Programming Languages

Perl, Shell Script ---> Scripting Languages

- C language missing the benefits of Object oriented programming features like, Encapsulation, Inheritance and Polymorphism etc.,
- Similarly OOP languages are not make use of the functional programming feactures up to the maximium extent.
- Scrpting language: Group of lines one by one will have to execute.

Every Programming language having it's own specific behaviour, that specific pradigm benefits only they are going to get.

What about Python?

Is it Functional Programming language?

(OR)

Is it Object Oriented Programming language?

(OR)

Is it Scripting Language?

While developing Python, Guido Van Rossum borrowed -

- Functional programming features from C

- Object Oriented Programming features from C++ (Because, Java was not developed at that time)

- Scripting language features from Perl,Shell Script.

So, Python is considerd as **All Rounder**. Python can enjoy the benefits of all types of programming language paradigms.

1. Python as Scripting Language:

Scripting Language: Scripting language means a grou of lines of code will be there and they will be executed line by line.

No functions concept, No classes concept, just a group lines will be executed one by one.

In [18]:

```
print("Python as Scripting language")
print("Python as Scripting language")
print("Python as Scripting language")
print("Python as Scripting language")
print("Python as Scripting language")
print("Python as Scripting language")
```

```
Python as Scripting language
Python as Scripting language
Python as Scripting language
Python as Scripting language
Python as Scripting language
Python as Scripting language
```

2. Python as Functional Programming Language:

In [21]:

```
def f1():
    print("Python as Functional Programming language")
    print("Python as Functional Programming language")
    print("Python as Functional Programming language")
    print("Python as Functional Programming language")

f1()
```

```
Python as Functional Programming language
Python as Functional Programming language
Python as Functional Programming language
Python as Functional Programming language
```

3. Python as Object Oriented Programming Language:

In [22]:

```
class Test:
    def m1(self):
        print("Python as Object Oriented Programming Language")

test = Test()
test.m1()
```

```
Python as Object Oriented Programming Language
```

Note:

- Most of the syntax used in Python borrowed from 'C' & 'ABC' Programming Language.

L5. Where We Can Use Python

We can use Python everywhere. The most common important application areas are as follows:

1. For developing **Desktop Applications**

- The Applications which are running on a single systems (i.e., Stand alone applications)

  Eg: Simple Calculator application

2. For developing **Web Applications**

   Eg: Gmail Application, Online E-commerce applications, Facebook application, Blog applications etc.,

3. For **Network Applications**

   Eg: Chatting applications, Client-Server applictaions etc.,

4. For **Games development**

5. For **Data Analysis Applications**

6. For **Machine Learning applications**

7. For developing **Artificial Intelligence, Deep Learning, Neural Network Applications**

8. For **IOT**

9. For **Data Sciene**

That's why Python is called as **General Purpose Programming Language.**

Which Software companies are using Python

- Internally Google and Youtube use Python coding
- NASA and Nework Stock Exchange Applications developed by Python.
- Top Software companies like Google, Microsoft, IBM, Yahoo, Dropbox, Netflix, Instagram using Python.

L6. Features of Python

1. Simple and easy to learn

- Consider English Language, how many words are there in english? Crores of words are there in english language. If you want to be perfect in english, you should aware about all these words.
- If you consider Java, You should aware of 53 words. That means when compared to English, learning Java is easy.
- If you consider Python Programming language, You should aware about 33 Words (Reserved Words). The person who can understand these 33 words, then he will become expert in Python.
- So, Python is a simple programming language. When we read Python program, we can feel like reading english statements.
- For example, if you consider ternary operator in Java,

x = (10>20)?30:40; ---> Java Statement

- If we ask any person, what this line is doing? 99% of Non-programming people are going to fail unless and until if they know Java.
- If I write the same thing in python,

x = 30 if 10>20 else 40 ----> Python Statement

- If Iwe ask any person, what this line is doing? 99% of Non programming people are going to give correct answer.
- When compared with other languages, we can write programs with very less number of lines (i.e, Concise Code) . Hence more readability and simplicity in the python code.
- Because of the concise code, we can reduce development and cost of the project.

Let us take an example,

Assume that We have one file (abc.txt) with some data in it. Write a Python program to read the data from this file and print it on the console.

If you want to write the code for the above problem in C or Java we need to make use of more lines of code. But if you write program in Python, just 1 line is more enough.

In [ ]:

```
# Execution pending
```

2. Freeware and Open Source

- Freeware and Open source are not same.

Freeware:

- To use Python, How much Licence fee we need to paid?

We need not pay single rupee also to make use of Python. It is freeware, any person can use freely, even for business sake also.

- If you consider Java, Java is vendered by Oracle. (Commercial)
- If you consider C# , C# is vendered by MicroSoft. (Commercial)
- If you consider Python, who is vendor for Python? There is no vendor for Python, there is one charitable Foundation, Python Software Foundation (PSF) is responsible for maintainane of Python. PSF is Non-Profit oriented Organization.
- To use Python, you need not pay any money to PSF. If you want to donate voluntarily for this foundation, you can pay.
- The corresponding Website for PSF is python.org, from where you have to download Python software.
- But for Java, from it's 11 version onwards it is the paid version. If you want to use for your personal use or business sake, compulsory licence must be required.
- C# & .Net also requires licence to use.

Open Source:

The Source code of the Python is open to everyone, so that we can we can customize based on our requirement. Because of this multiple flovours of Python is possible.

Eg:

1. Jython is customized version of Python to work with Java Applications.
2. Iron Python is customized version of Python to work with C## & .Net Applications.
3. Anaconda Python is customized version of Python to work with Bigdata Applications.

One main advantage of Python is for every requirement specific version is availble in the market.

We can use our specific version of python and fulfill our requirement.

3. High Level Programmimg Language

- High level programming language means Programmer friendly language.
- Being a programmer we are not required to concentrate low level activities like memory management and security etc..
- Any programmer can easily read and understand the code. Let's see the below example,

In [3]:

```
a = 20
b = 30
print(a+b)
```

```
50
```

4. Platform Independent

Assume that one C program is there, We have three machines are there which are working on three platforms (i.e., Windows,Linux, MAC). Now, we want to distribute One C application to the clients who are working on different platforms. Then what we need to do is,

- For Windows, a seperate C program must be reqired. A C program for Windows system can't run on Linux machine or MAC machine.
- For Linux, a seperate C program must be reqired. A C program for Linux system can't run on Windows machine or MAC machine.
- For MAC, a seperate C program must be reqired. A C program for MAC system can't run on Linux machine or Windows machine.-

So, Compulsory for every platform you have to write the platform specific application. Now we have three applications and three platforms.

So, **C programming language is platform dependent language.**

Assume that one Python program is there, We have three machines are there which are working on three platforms (i.e., Windows,Linux, MAC). Now, we want to distribute One Python application to the clients who are working on different platforms. Then what we need to do is,

- Write the Python program once and run it on any machine. This is the concept of Platform Independent nature.

So, **Python programming language is Platform Independent Language.**

How platform independent nature is implemented in Python?

If you want to run Python application on Windows platform, what must be required is **Python Virtual Machine (PVM)** for Windows is required. If we provide Python application to the PVM for Windows, PVM is responsible to convert the python program into Windows specific and execute it.

In the same way, If you want to run Python application on Linux platform, what must be required is **Python Virtual Machine (PVM)** for Linux is required. If We provide Python application to the PVM for Linux, PVM is responsible to convert the python program into Linux specific and execute it.

In the same way, If you want to run Python application on MAC platform, what must be required is **Python Virtual Machine (PVM)** for MAC is required. If We provide Python application to the PVM for MAC, PVM is responsible to convert the python program into MAC specific and execute it.

- **Platform specific conversions are taken care by PVM**.
- **Python program is Platform independent**
- **Python Virtual Machine is Platform dependent**

5. Portability

- In general poratble means movable. In our childhood days, we heard about portable TV (14 inches), which can be moved from one place to another place very easily.
- Another place where we commonly used the term Portablity is mobile number portability.
- Now Python application Portability means,

```
- Assume that one windows machine is there, in this machine your python applicatio
n is running without any difficulty.  Because of licence issue or security issue y
ou want to move to Linux machine. If you are migrating to Linux machine from Windo
ws is it possible to migrate your python application or not?  Yes, because Python
 application never talks about underlying platform. Without performing any changes
in your Python application, you can migrate your Python application from one platf
orm to another platform. This is called Portability.
```

6. Dynamically Typed

In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically.Hence Python is considered as dynamically typed language.

But Java, C etc are Statically Typed Languages because we have to provide type at the beginning only.

This dynamic typing nature will provide more flexibility to the programmer.

7. Python is both Procedure oriented and Object orieted

Python language supports both Procedure oriented (like C, pascal etc) and object oriented (like C++,Java) features. Hence we can get benefits of both like security and reusability etc.

8. Interpretted

- We are not required to compile Python code.
- If you consider C program, we have to compile and execute the code.
- If you consider Java program, we have to compile and execute the code.
- If you consider Python program, we have execute the code. We are not required to compile. Internally Interpretter is responsible for compilation of the Python code.
- If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python Virtual Machine) is responsible to execute.

9. Extensible

You can extend the functionality of Python application with the some other languages applications. what it means that -

Let us assume that some C program is there, some Java program is there, can we use these applications in our Python program or not?

- **yes, we can use other language programs in Python.**

What is the need of that?

1. Suppose We want to develop a Python application, assume that some **xyz** functionality is required to develop the Python application.
2. There is some java code is already there for this **xyz** functionality. It is non python code. Is it possible to use this non-python code in side our python application.

Yes, No problem at all.

The main advantages of this approach are:

1. We can use already existing legacy non-Python code
2. We can improve performance of the application

10. Embedded

Embedded means it is same as **extensible in reverse.**

We can use Python programs in any other language programs. i.e., we can embedd Python programs anywhere.

11. Extensive Library

- In Python for every requirement, a readymade library is availbale.
- Lakhs of libraries are there in Python.No other programming language has this much of librrary support.
- Python has a rich inbuilt library.
- Being a programmer we can use this library directly and we are not responsible to implement the functionality.

Eg: Write a Python program to generate 6 digit OTP

- In Python to generate random numbers already a library is availbale. By make use of that library we can write the code in easy manner.

In [11]:

```
from random import randint
print(randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9))
```

```
5 0 8 8 3 1
```

If dont want space between numbers, include **sep = ''** at the end. sep means seperator, that is assigned with empty.

In [10]:

```
from random import randint
print(randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),sep =''
```

```
836593
```

Suppose, if we want 10 OTPs, then Python code looks like this:

In [8]:

```
from random import randint
for i in range(1,11):
    print(randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),sep
```

```
650052
666097
558558
743920
295868
950438
319213
198749
795225
269510
```

In [9]:

```
from random import randint
for i in range(10):    # another way of using 'range()'
    print(randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),sep
```

```
587659
654352
508094
836302
761498
517296
116376
704038
937445
920345
```

Conclusion:

1. These are the 11 key features of Python programming language.
2. Among various features of Python discussed above, the following 3 features are specific to Python

```
- Dynamically Typed


- Both Procedural Oriented and Object Oriented


- Extensive Library


These 3 Features are not supported by any other programming languages like C,C++ a
nd Java etc.,
```

Date: 11-04-2020 - Day 3

L7: Limitations and Flavours of Python

Limitaions of Python:

Eventhough Python is effective programming language, there are some areas where Python may not work up to the mark.

Now a days, Machine Learning (ML) is the trending word. To develop ML application, Python is the best choice. The reason is Python contains several libraries, using those libraries we can develop ML applicatios very easily.

For example, in Python we have the following modules to perform various operations:

- There is a module called as **numpy**, which adds mathematical functions to Python.
- To import and read data set, we have another module in Python called as **pandas**.

- To project the data in the form of Graphs, there is an another module is available in Python called as **mathplotlib**.

1.Suppose We want to develop mobile applications, Python is the worst choice. Why?

The main reason for this is, Python, as of now, **not having library support** to develop mobile applications.

Which programming language is the best choice for mobile applications?

- Android, IOs Swift are thekings in Mobile application development domain.

2.Suppose We want to develop Enterprise applications such as Banking, Telecom applications where multiple services are required (For ex, transaction management, Security, Messaging etc.,).

- To develop these end-to-end applications Python is not best suitable, because, Python doesn't have that much Library support to develpo these applications as of now.

3.We are already discussed that, Python is interpretted programing language, here execution can be done line by line. That's why performance wise Python is not good. Usually Interpreted programming languages are performance wise not up to the mark.

To improve performance, Now people are added **JIT compiler to the PVM**. This works in the following manner:

- Instead of interpreting line by line everytime, a group of lines will be interprtted only once and everytime that interpretted code is going to used directly. JIT compiler is responsible to do that.
- JIT compiler + PVM flovour is called pypy. If you want better performance then you should go for pypy(Python for speed) version.

Note : These 3 are the various limitations of the Python.

Flavours of Python:

As we are already discussed that Python is an Open source. That means it's source code is available to everyone. Assume that the standard Pyhthon may not fulfill our requirement. So, what we need to do is, we have to access the source code and make some modifications and that **customized Python version** can fulfill my requirement.

For Python, multiple floavours are available, each flavour itself is a customized version tofulfill a particular requirement.

Folowwing are the various flavours of Python:

1. CPython:

- It is the standard flavor of Python. It can be used to work with C lanugage Applications

2. Jython or JPython:

- It is for Java Applications. It can run on JVM

3. IronPython:

- It is for C#.Net platform

4. PyPy:

- The main advantage of PyPy is performance will be improved because JIT compiler is available inside PVM.

5. RubyPython

- For Ruby Platforms

6. AnacondaPython

- It is specially designed for handling large volume of data processing.

L8: Python Identifiers

What is an Identifier?

A name in Python program is called identifier. It can be class name or function name or module name or variable name.

Eg: a = 20

It is a valid Python statement. Here 'a' is an identifier.

Rules to define identifiers in Python:

1. The only allowed characters in Python are
   - alphabet symbols(either lower case or upper case)
   - digits(0 to 9)
   - underscore symbol(_)

In [5]:

```
cash =10    # it is a valid identifier
```

In [4]:

```
cash$ = 10    # '$'is not allowed as valid identifier
```

```
  File "<ipython-input-4-e65405076c1e>", line 1
    cash$ = 10    # '$'is not allowed as valid identifier
        ^
SyntaxError: invalid syntax
```

In [9]:

```
all!hands = 30   # '!'is not allowed as valid identifier
```

```
  File "<ipython-input-9-1d588afcae49>", line 1
    all!hands = 30   # '@'is not allowed as valid identifier
       ^
SyntaxError: invalid syntax
```

2. Identifier should not starts with digit

In [10]:

```
total123 = 33
```

In [11]:

```
123total = 122
```

```
  File "<ipython-input-11-e0b9e967153d>", line 1
    123total = 122
           ^
SyntaxError: invalid syntax
```

3. Identifiers are case sensitive. Of course Python language itself is case sensitive language.

In [12]:

```
total=10
TOTAL=999
print(total)   #10
print(TOTAL)   #999
```

```
10
999
```

4. There is no length limit for Python identifiers. But not recommended to use too lengthy identifiers.

In [13]:

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

In [14]:

```
print(xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
9999
```

5. We cannot use reserved words as identifiers

In [15]:

```
x = 10        # Valid
```

In [16]:

```
if = 33      # 'if' is a keyword in Python
```

```
  File "<ipython-input-16-0abaf39d7bbf>", line 1
    if = 33
       ^
SyntaxError: invalid syntax
```

Q. Which of the following are valid Python identifiers?

In [17]:

```
123total = 22
```

```
  File "<ipython-input-17-eaf314d05be5>", line 1
    123total
           ^
SyntaxError: invalid syntax
```

In [20]:

```
total1234 = 22    # Valid
```

In [21]:

```
java2share = 'Java'     # Valid
```

In [22]:

```
ca$h = 33
```

```
  File "<ipython-input-22-d315f9380372>", line 1
    ca$h = 33
      ^
SyntaxError: invalid syntax
```

In [23]:

```
_abc_abc_ = 22   # Valid
```

In [24]:

```
def = 44
```

```
  File "<ipython-input-24-be1255e3d0b6>", line 1
    def = 44
        ^
SyntaxError: invalid syntax
```

In [25]:

```
for = 3
```

```
  File "<ipython-input-25-c8ee3642ab3d>", line 1
    for = 3
        ^
SyntaxError: invalid syntax
```

In [26]:

```
__p__ = 33   # Valid
```

L9: Reserved words (or) Keywords in Python

In Python some words are reserved to represent some meaning or functionality. Such type of words are called Reserved words.

There are 33 reserved words available in Python.

- True,False,None
- and, or ,not,is
- if,elif,else
- while,for,break,continue,return,in,yield
- try,except,finally,raise,assert
- import,from,as,class,def,pass,global,nonlocal,lambda,del,with

1. All 33 keywords contains only alphabets symols.

2. Except the following 3 reserved words, all contain only lower case alphabet symbols.

    - True
    - False
    - None

In [29]:

```
a = true
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-c100e6034b0a> in <module>
----> 1 a = true

NameError: name 'true' is not defined
```

In [31]:

```
a = True   # Valid
```

For Boolean values, compulsory you need to use capital letter 'T' in True and capital letter 'F' in False.

Key Points:

- **switch concept** is not there in Python.
- Similarly **do while** loop is not there in Python.
- **'int','float','char' and 'double'** such type of words are not reserved words in python, because Python is dynamically typed language.

Note : Learning Python language itself learning of all the keywords of Python.

In [30]:

```
import keyword
for kword in keyword.kwlist:
    print(kword, end = ' ')    #33 keywords are displaying
```

```
False None True and as assert async await break class continue def del elif
else except finally for from global if import in is lambda nonlocal not or p
ass raise return try while with yield
```

In [31]:

```
import keyword
print(keyword.kwlist,end=' ')
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'fo
r', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'no
t', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

L9. Data types in Python - Introduction

Data Type represent the type of data present inside a variable.

In Python we are not required to specify the type explicitly. Based on value provided,the type will be assigned automatically. Hence Python is **Dynamically Typed Language.**

Among the following two statements, which are correct statements?

1. In Python, 'type' concept is not applicable.
2. In Python, 'type' concept is availbale, but we are not require to declare explicitly.

statement 2 only correct.

Python contains the following in-built data types

1. int
2. float
3. complex
4. bool
5. str
6. bytes
7. bytearray
8. range
9. list
10. tuple
11. set
12. frozenset
13. dict
14. None

Almost about 14 data types we need to discuss in detail.

Before going to discuss about these data types, let us know few imporatant points now.

Note: In Python, every thing is an Object.

Let us take an example,

a = 10

- In this statement **10** is an object of class **'int'**.
- Here, 'a' is the reference variable which is pointing to 'int' object. The vale representing in the 'int' object is **10**.

How can you find the type of 'a'?

- By using an in-built function **type()**, we can find the type of any variable.

In [4]:

```
a = 10
type(a)
```

Out[4]:

```
int
```

Where the object 10 is stored in the memory? (or) **What is the address of the object 'a'?**

- By using an in-built function **id()**, we can find the address of an object.

In [5]:

```
a = 10
id(a)
```

Out[5]:

```
140712714015840
```

How can you print the value of 'a'?

- By using an in-built function **print()**, we can print the value of a variable.

In [3]:

```
a = 10
print(a)
```

```
10
```

Note:

The most commonly used in-built functions in Python are as follows:

1. type()

- It is to check the type of variable

2. id()

- It is used to get address of object

3. print()

- It is used to print the value

L10. Data types in Python - int data type

1. Integer type

- We can use **'int'** data type to represent whole numbers (integral values)

In [6]:

```
a=10
type(a) #int
```

Out[6]:

```
int
```

Suppose we want to represent the following number, 1234567888897333. Is it Integral number or not? Yes, it is not taking any fractional part, so it is an integer number.

This number is too long number. To represent too long numbers, we have **long** data type is there. To represent small integer numbers, we have **int** data type is there.

Note:

In Python2 we have long data type to represent very large integral values. But in Python3 there is no long type explicitly and we can represent long values also by using int type only.

In [8]:

```
a = 12345678889733333333333333333333333333333333333333
type(a)
```

Out[8]:

```
int
```

We can represent integral values in the following ways:

1. Decimal form (base 10)
2. Binary form (base 2)
3. Octal form (base 8)
4. Hexa decimal form (base 16)

1. Decimal form(base-10):

- It is the default number system in Python
- The allowed digits are: 0 to 9

  Eg: a =10

2. Binary form(base-2):

- The allowed digits are : 0 & 1
- Literal value should be prefixed with **0b** or **0B**

  Eg:

  a = 0B1111

  a =0B123

  a=b111

In [10]:

```
a = 1111   # it is ot treated as binary number, by default every number is treated as decim
print(a)
```

```
1111
```

In [11]:

```
a = 0b1111
print(a)
```

```
15
```

In [12]:

```
a = b111
print(a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-08fe5f31f204> in <module>
----> 1 a = b111
      2 print(a)

NameError: name 'b111' is not defined
```

In [13]:

```
a = 0b111
print(a)
```

```
7
```

3. Octal Form(base-8):

- The allowed digits are : 0 to 7
- Literal value should be prefixed with **0o** or **0O**.

In [15]:

```
a = 0o123   # 64 + 16 + 3 = 83
print(a)
```

```
83
```

In [17]:

```
a = 0o786   # error, because 8 is not an valid octal number
print(a)
```

```
  File "<ipython-input-17-7f92d3a1f0b2>", line 1
    a = 0o786   # error, because 8 is not an valid octal number
           ^
SyntaxError: invalid syntax
```

4. Hexa Decimal Form(base-16):

- The allowed digits are : 0 to 9, a-f (both lower and upper cases are allowed)
- Literal value should be prefixed with 0x or 0X

In [18]:

```
a = 0x10
print(a)
```

```
16
```

In [24]:

```
a = 0xface
print(a)
```

```
64206
```

In [25]:

```
a = 0xbeef
print(a)
```

```
48879
```

In [26]:

```
a=0xbeer     # 'r' is not valid hexa decimal digit
print(a)
```

```
  File "<ipython-input-26-b96cba1ff699>", line 1
    a=0xbeer     # 'r' is not valid hexa decimal digit
           ^
SyntaxError: invalid syntax
```

Note:

Being a programmer we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only in decimal form.

L11. Data types: Base conversion functions

In this lecture, we will learn about how to convert integral values from one base to another base. Python provide three in-built functions for base conversions.

1. bin():

We can use **bin()** to convert from any other base to binary

In [1]:

```
bin(15)
```

Out[1]:

```
'0b1111'
```

In [4]:

```
bin(0o123)
```

Out[4]:

```
'0b1010011'
```

In [5]:

```
bin(0xface)
```

Out[5]:

```
'0b1111101011001110'
```

2. oct()

We can use **oct()** to convert from any other base to octal

In [6]:

```
oct(0b111101)
```

Out[6]:

```
'0o75'
```

In [7]:

```
oct(0xface)
```

Out[7]:

```
'0o175316'
```

In [8]:

```
oct(100)
```

Out[8]:

```
'0o144'
```

3. hex()

We can use **hex()** to convert from any other base to hexa decimal

In [9]:

```
hex(1000)
```

Out[9]:

```
'0x3e8'
```

In [10]:

```
hex(0b10111111)
```

Out[10]:

```
'0xbf'
```

In [11]:

```
hex(0o123456)
```

Out[11]:

```
'0xa72e'
```

By using these base conversion functions, we can convert from one base to another base.

Note :

All these base conversion functions are applicable only for Integral numbers.

L12. Data types: float data type

We can use **float** data type to represent floating point values (Number with decimal values)

In [12]:

```
f=1.234
type(f)
```

Out[12]:

```
float
```

Note: We can represent integral values in decimal, binary, octal and hexa decimal forms. But **we can represent float values only by using decimal form.**

In [13]:

```
f=1.234
```

In [14]:

```
f = 0b1.234
```

```
  File "<ipython-input-14-ce35d4648a2f>", line 1
    f = 0b1.234
             ^
SyntaxError: invalid syntax
```

In [15]:

```
f=0o1.234
```

```
  File "<ipython-input-15-872cc5f22444>", line 1
    f=0o1.234
           ^
SyntaxError: invalid syntax
```

In [16]:

```
f=0x1.23
```

```
  File "<ipython-input-16-4e3a508aa64c>", line 1
    f=0x1.23
          ^
SyntaxError: invalid syntax
```

We can also represent floating point values by using exponential form (scientific notation)

In [17]:

```
f = 1.2e3
print(f)
```

```
1200.0
```

instead of 'e' we can use 'E'

In [18]:

```
f = 1.2E3
print(f)
```

```
1200.0
```

The main advantage of exponential form is we can represent big values in less memory.

Assume that we need to store a value (12000000000000000.0)

In [20]:

```
f=1.2e16
print(f)
```

```
1.2e+16
```

Even in our calculators also to represent bigger values, we need to go for exponential notation.

L13. Data types: complex data type

In this lecture, we'll discuss about Python specific special data type known as **Complex** data type.

Why Python having this special data type?

- If you want to develop scientific applications, mathematics based applications and Electrical engineering applications, thiscomplex type is very very helpful.

How can we represent a complex number?

- **a + bj** is the syntax for representing a complex number.
  - Here, **a** is called **real part** and **b** is called **imaginary part**.
  - **j** value is **square of J is -1** and **j = square root of -1**

you may get one doubt that in the complex number representation is it compulsory **j*? In mathematics we seen *i** instead of **j**.

It is mandatory, it should be **j** only in Python.

In [38]:

```
x = 10 + 20j
type(x)
```

Out[38]:

```
complex
```

In [22]:

```
x = 10 + 20J
type(x)
```

Out[22]:

```
complex
```

In [23]:

```
x = 10 + 20i
type(x)
```

```
  File "<ipython-input-23-f3370a6b6c4b>", line 1
    x = 10 + 20i
              ^
SyntaxError: invalid syntax
```

Key Points:

- **In the real part if we use int value then we can specify that either by decimal,octal,binary or hexa decimal form.**
- **imaginary part must be specified only by using decimal form.**

In [24]:

```
x = 10 +20j
print(x.real)   # prints only real part
```

```
10.0
```

In [25]:

```
x = 10 +20j
print(x.imag)  # prints only imaginary part
```

```
20.0
```

In [26]:

```
x = 10.5+20j      # real part is a float value; Acceptable
```

In [28]:

```
x = 10 + 20j    # real part is a int value; Acceptable
```

In [30]:

```
x = 10.5 +20.6j  # Both real and imaginary parts also float values; Acceptable
```

In [31]:

```
x = 0b1111 + 20j   # Acceptable
```

In [32]:

```
x = 15 + 0b1111j
```

```
  File "<ipython-input-32-ac01ca58fad6>", line 1
    x = 15 + 0b1111j
                   ^
SyntaxError: invalid syntax
```

Assume that, we have two complex numbers. Can we perform arithmetic operations between these two complex numbers?

Yes, we can perform without any difficulty.

In [33]:

```
x = 10 + 20j
y = 20 + 30j

print(x+y)
```

```
(30+50j)
```

In [34]:

```
x = 10 + 20j
y = 20 + 30j

print(x-y)
```

```
(-10-10j)
```

In [35]:

```
x = 10 + 20j
y = 20 + 30j

print(x*y)
```

```
(-400+700j)
```

In [36]:

```
x = 10 + 20j
y = 20 + 30j

print(x/y)
```

```
(0.6153846153846154+0.07692307692307691j)
```

```
In [37]:

x = 10 + 20j
y = 20 + 30j

print(x//y)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-d6bb0a6e2dfd> in <module>
      2 y = 20 + 30j
      3
----> 4 print(x//y)

TypeError: can't take floor of complex number.
```

Note :

This is about basic introduction about complex data type.
It is not that much frequently used data type in Python.
It is very specific to Scientific, Mathematical and Electrical Engineering Applications.

L14. Data types: bool data type

We can use this data type to represent boolean values.

The only allowed values for this data type are: True and False (true & false are not allowed in Python)

Internally Python represents True as 1 and False as 0

```
In [39]:

b = True
type(b)

Out[39]:

bool
```

```
In [40]:

b = true
type(b)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-40-034f4c6a9dcd> in <module>
----> 1 b = true
      2 type(b)

NameError: name 'true' is not defined
```

```
In [42]:

a = 10
b = 20
c = a>b
print(c)
type(c)


False

Out[42]:

bool
```

```
In [43]:

print(True + True)

2
```

```
In [44]:

print(True - False)

1
```

```
In [45]:

print(True * False)

0
```

L15. Data types: str data type representations by using single, double and triple quotes

> **str** represents String data type.
>
> It is the most commonly used data type in Python

String: A String is a sequence of characters enclosed within single quotes or double quotes.

In Python to represent a string, can we use a pair of single quotes ('') or double quotes ("")?

> The answer is, We can use either single quotes or double quotes.

```
In [46]:

s = 'Karthi'
print(type(s))

<class 'str'>
```

In [47]:

```
s = "Karthi"
print(type(s))
```

```
<class 'str'>
```

In [48]:

```
s = 'a'
print(type(s))   # in Python there is no 'char'data type
```

```
<class 'str'>
```

In [49]:

```
s = "a"
print(type(s))
```

```
<class 'str'>
```

In [51]:

```
s = 'a'
print(s)    # value of 's'
print(type(s)) #type of 's'
```

```
a
<class 'str'>
```

In Python, we can use triple quotes also in the following 3 situations.

1.By using single quotes or double quotes we cannot represent multi line string literals.

For example,

s = "Karthi

sahasra"

For this requirement we should go for triple single quotes(''') or triple double quotes(""").

In [56]:

```
s = "Karthi
sahasra"
```

```
  File "<ipython-input-56-cb6a1bde203b>", line 1
    s = "Karthi
               ^
SyntaxError: EOL while scanning string literal
```

In [55]:

```
s = 'Karthi
sahasra'
```

```
  File "<ipython-input-55-ad08d7556fbe>", line 1
    s = 'Karthi
               ^
SyntaxError: EOL while scanning string literal
```

In [54]:

```
s = '''Karthi
sahasra'''
print(s)
```

```
Karthi
sahasra
```

In [57]:

```
s = """Karthi
sahasra"""
print(s)
```

```
Karthi
sahasra
```

2.We can also use triple quotes, to use single quotes or double quotes as normal characters in our String .

In [59]:

```
s = 'class by 'durga' is very good'
```

```
  File "<ipython-input-59-88963cd2f04c>", line 1
    s = 'class by 'durga' is very good'
                       ^
SyntaxError: invalid syntax
```

In [61]:

```
s = "class by 'durga' is very good"
s    # if you want to include single quotes within the string keep the string in double quc
```

Out[61]:

```
"class by 'durga' is very good"
```

In [62]:

```
s = "class by "durga" is very good"
print(s)
```

```
  File "<ipython-input-62-50981f425bf1>", line 1
    s = "class by "durga" is very good"
                        ^
SyntaxError: invalid syntax
```

In [63]:

```
s = 'class by "durga" is very good'
print(s)                # if you want to include double quotes within the string keep the str
```

```
class by "durga" is very good
```

Now, if we want to use both single quotes and double quotes as the normal characters in the string, then you need to enclose the string in triple quotes.

In [65]:

```
s = "classes by 'durga' for "python" is very good"
print(s)
```

```
  File "<ipython-input-65-aa9719991cfd>", line 1
    s = "classes by 'durga' for "python" is very good"
                                     ^
SyntaxError: invalid syntax
```

In [66]:

```
s = 'classes by 'durga' for "python" is very good'
print(s)
```

```
  File "<ipython-input-66-94674ab79602>", line 1
    s = 'classes by 'durga' for "python" is very good'
                          ^
SyntaxError: invalid syntax
```

In [67]:

```
s = """classes by 'durga' for "python" is very good"""
print(s)
```

```
classes by 'durga' for "python" is very good
```

3.To define doc string, triple quotations will be used. (We will discuss this later)

Date: 12-04-2020 - Day 4

L16. Data types: str data type - positive and negative index

One speciality is there in Python indexing, which is not available in C orJava.

The characters of the string is accessed by using it's relative position in the string, that is called as **index**.

In Python, indexing starts from 0.

```
In [38]:

s = "karthi"
print(s[0])              # The character locationg at 0 index is displayed
print(s[3])
print(s[100])

k
t

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-38-dc4df5484ca1> in <module>
      2 print(s[0])              # The character locationg at 0 index is dis
played
      3 print(s[3])
----> 4 print(s[100])

IndexError: string index out of range
```

Upto this is similar in C or Java like languages. Now we will see what is the speciality regrding indexing in Python.

Python supports both **positive indexing** and **negative indexing**.

As we are already discussed, positive indexing moves in forward direction of string and starts from 0.

Negative indexing moves in reverse direction of string and starts from -1.

In [42]:

```
print(s[-1])   # it won't give index error, that is the speciality of Python. It prints the
print(s[-6])
print(s[-7])
```

```
i
k

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-42-2c0d90d443f9> in <module>
      1 print(s[-1])   # it won't give index error, that is the speciality o
f Python. It prints the last character of the string.
      2 print(s[-6])
----> 3 print(s[-7])

IndexError: string index out of range
```

L17. Data types: str data type - Slice operator

In this lecture, we are going to learn one very important Python specific special operator known as **Slice** operator.

What is Slice?

Suppose, if you have an apple and if you cut it into multiple pieces. Each piece is called as a **slice**.

Similarly, **string slice means a part of the string.** You will get the part of the string by using **slice** operator.

In [43]:

```
s = 'abcdefghijklmnopqrstuvwxyz'
```

Now, If we wnat to get which character is locating at specific index position, simply writing **s[index]** will automaticlly get that character.

If we want to get slice or piece of the string, for example we want the piece of the string from index position 3 to index position 7 (i.e., total 5 characters). You can get this piece of the string by using **slice** operator.

Syntax of slice operator:

stringName [beginIndex:endIndex]

This operator returns the substring (slice) from beginIndex to endIndex - 1.

In [46]:

```
s = 'abcdefghijklmnopqrstuvwxyz'
slice = s[3:8]   # returns characters from 3 to 7 index
print(slice)
```

```
defgh
```

Suppose, If you are not specifying the begin index, then the default value of the begin index is starting index of the string [i.e,. 0].

In [47]:

```
s = 'abcdefghijklmnopqrstuvwxyz'
slice = s[:8]   # returns characters from 3 to 7 index
print(slice)
```

```
abcdefgh
```

Suppose, If you are not specifying the end index, then the default value of the end index is ending index of the string [i.e,. -1].

In [48]:

```
s = 'abcdefghijklmnopqrstuvwxyz'
slice = s[3:]   # returns characters from 3 to 7 index
print(slice)
```

```
defghijklmnopqrstuvwxyz
```

Suppose, If we are not specifying begin index and end index. What happens?

In [50]:

```
s = 'abcdefghijklmnopqrstuvwxyz'
slice = s[ : ]   # returns characters from 3 to 7 index
print(slice)
```

```
abcdefghijklmnopqrstuvwxyz
```

In [51]:

```
s = 'abcdefghijklmnopqrstuvwxyz'
slice = s[3:1000]
print(slice)
```

```
defghijklmnopqrstuvwxyz
```

slice opertaor never goes to raise index error

In [54]:

```
s = 'abcdefghijklmnopqrstuvwxyz'
slice = s[5:1]          # it starts from 5 and goes in farward direction and never gets er
print(slice)            # Empty string will be displayed
```

Note : In this lecture, we discussed briefly about **slice** operator. We will discuss indetail in later.

L17. Data types: str data type - Slice operator Applications

1. Convert the first letter of the string into uppercase letter

In [56]:

```
s = 'karthi'
output = s[0].upper() + s[1:]
print(output)
```

```
Karthi
```

2. Convert the last letter of the string into uppercase letter

In [59]:

```
s = 'karthi'
output = s[0:len(s)-1] + s[-1].upper()
print(output)
```

```
karthI
```

3. Convert the first and last letter of the string into uppercase letter

In [61]:

```
s = 'karthisahasra'
output = s[0].upper() + s[1:len(s)-1]  + s[-1].upper()
print(output)
```

```
KarthisahasrA
```

L18. Data types: + and * operators for str data type

Related to strings there are two important points we want to discuss with respect to mathematical operations.

1. '+' operator for the string:

In [62]:

```
s = 'karthi' + 'sahasra'    # concatenation
print(s)
```

```
karthisahasra
```

In [63]:

```
s = 'karthi' + 10  # in Java, ouput is karthi10, but in python it gives error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-63-8225e0d23012> in <module>
----> 1 s = 'karthi' + 10

TypeError: can only concatenate str (not "int") to str
```

In python, if you are performing concatenation operation (i.e., '+' operation on strings), then both operands must be string type.

2. '*' operator [String repetetion operator] for the string:

This speciality is not there in other programming languages.

In [64]:

```
s = 'karthi' * 3      #string repetetion operator
print(s)
```

```
karthikarthikarthi
```

In [65]:

```
s = 3 * 'karthi'
print(s)
```

```
karthikarthikarthi
```

In python, if you are performing string repetetion operation (i.e., '*' operation on strings), one operand should be an integer type and another one is string type.

In [67]:

```
s = 'karthi'*'sahasra'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-67-bd21e5426aca> in <module>
----> 1 s = 'karthi'*'sahasra'

TypeError: can't multiply sequence by non-int of type 'str'
```

In [68]:

```
print('#' * 10)
print("karthi")
print('#' * 10)
```

```
##########
karthi
##########
```

In [73]:

```
print('#' * 10,end = '')
print("karthi",end='')
print('#' * 10)
```

```
##########karthi##########
```

In [74]:

```
print('#' * 10,end = ' ')
print("karthi",end=' ')
print('#' * 10)
```

```
########## karthi ##########
```

Important Conclusions :

1. So far, we covered the following datatypes of Python:

   ```
   1. int

   2. float

   3. complex

   4. bool

   5. str
   ```

These 5 datatypes are called as fundamental datatypes of Python.

2. **long** datatype is available in Python-2, but not in Python-3. **long** values also you can represent by using **int** type in Python-3.

3. There is no **char** datatype in Python, **char** values also you can represent by using **str** type.

L19. Type casting : Introduction and int() function

Type casting or Type Coersion:

In this lecture, we will learn about how to convert one type value to another type value. The process of converting the value from one type to another type is known as **Type casting or Type Coersion**.

Python provides 5 in-built functions, which are used to convert the values from one type to another type. These are listed as below:

1. int()
2. float()
3. complex()
4. bool()
5. str()

1.int() :

We can use this function to convert values from other types to int

In [1]:

```
int(10.989)
```

Out[1]:

```
10
```

In [2]:

```
int(10+5j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-aeeb86d69c41> in <module>
----> 1 int(10+5j)

TypeError: can't convert complex to int
```

In [3]:

```
int(True)
```

Out[3]:

```
1
```

In [4]:

```
int(False)
```

Out[4]:

```
0
```

In [5]:

```
int('10')   # string internally contains only integral value and should be specified in dec
```

Out[5]:

```
10
```

In [7]:

```
int('0b1111')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-afa2914aea61> in <module>
----> 1 int('0b1111')

ValueError: invalid literal for int() with base 10: '0b1111'
```

In [8]:

```
int("10.5")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-54dd49a25c21> in <module>
----> 1 int("10.5")

ValueError: invalid literal for int() with base 10: '10.5'
```

In [10]:

```
int("ten")   # string should contain only integral value and specified with base 10
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-2eb201668cfb> in <module>
----> 1 int("ten")   # string should contain only integral value and specifi
ed with base 10

ValueError: invalid literal for int() with base 10: 'ten'
```

Note:

1. We can convert from any type to int except complex type.
2. If we want to convert str type to int type, compulsary str should contain only integral value and should be specified in base-10

L20. Type casting :float() and complex() functions

2. float() :

We can use float() function to convert other type values to float type.

In [11]:

```
float(10)
```

Out[11]:

```
10.0
```

In [12]:

```
float(10+5j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-d2f956539d9b> in <module>
----> 1 float(10+5j)

TypeError: can't convert complex to float
```

In [13]:

```
float(True)
```

Out[13]:

```
1.0
```

In [14]:

```
float(False)
```

Out[14]:

```
0.0
```

In [15]:

```
float('10')
```

Out[15]:

```
10.0
```

In [16]:

```
float('10.5')
```

Out[16]:

```
10.5
```

In [17]:

```
float('0b1111')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-8a00d88b4550> in <module>
----> 1 float('0b1111')

ValueError: could not convert string to float: '0b1111'
```

In [18]:

```
float("ten")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-c8abde1341af> in <module>
----> 1 float("ten")

ValueError: could not convert string to float: 'ten'
```

Note:

1. We can convert any type value to float type except complex type.
2. Whenever we are trying to convert str type to float type compulsary str should be either integral or floating point literal and should be specified only in base-10.

3.complex() :

We can use complex() function to convert other types to complex type.

There are two forms of complex() function is there.

Form 1 : complex(x)

We can use this function to convert x into complex number **with real part x and imaginary part 0.**

In [19]:

```
complex(10)
```

Out[19]:

```
(10+0j)
```

In [27]:

```
complex(0b1111)
```

Out[27]:

```
(15+0j)
```

In [20]:

```
complex(10.5)
```

Out[20]:

```
(10.5+0j)
```

In [21]:

```
complex(True)
```

Out[21]:

```
(1+0j)
```

In [22]:

```
complex(False)
```

Out[22]:

```
0j
```

In [23]:

```
complex("10")
```

Out[23]:

```
(10+0j)
```

In [24]:

```
complex("10.5")
```

Out[24]:

```
(10.5+0j)
```

In [25]:

```
complex("ten")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-25-b7bf9859d3d9> in <module>
----> 1 complex("ten")

ValueError: complex() arg is a malformed string
```

Form 2: complex(x,y)

We can use this method to convert x and y into complex number such that **x will be real part and y will be imaginary part.**

In [28]:

```
complex(10,20)
```

Out[28]:

```
(10+20j)
```

In [29]:

```
complex(10.5,20.6)
```

Out[29]:

```
(10.5+20.6j)
```

In [32]:

```
complex("10","20") # Rule 1 is, If you want to pass string in the real part, then second ar
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-2223b6237325> in <module>
----> 1 complex("10","20") # Rule 1 is, If you want to pass string in the re
al part, then second argument you can't pass

TypeError: complex() can't take second arg if first is a string
```

In [33]:

```
complex(10,"20")  #Rule 2 is second argument can't be a string
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-a8404f72f6be> in <module>
----> 1 complex(10,"20")  #Rule 2 is second argument can't be a string

TypeError: complex() second arg can't be a string
```

L21. Type casting :bool() and str() functions

4. bool():

We can use this function to convert other type values to bool type.

If we pass **integer arguments**

```
In [35]:
bool(10)
Out[35]:
True
```

```
In [36]:
bool(0)
Out[36]:
False
```

```
In [37]:
bool(-10)
Out[37]:
True
```

If we pass **float** arguments

```
In [38]:
bool(0.0)
Out[38]:
False
```

```
In [39]:
bool(0.1)
Out[39]:
True
```

```
In [40]:
bool(0.000000000001)
Out[40]:
True
```

```
In [41]:
bool(-0.00000000000001)
Out[41]:
True
```

If we pass **complex type** arguments

```
In [42]:

bool(0+0j)

Out[42]:

False
```

```
In [43]:

bool(0+0.5j)

Out[43]:

True
```

```
In [44]:

bool(1+0j)

Out[44]:

True
```

If we pass **string** type arguments

```
In [45]:

bool("True")

Out[45]:

True
```

```
In [46]:

bool("False")

Out[46]:

True
```

```
In [47]:

bool("yes")

Out[47]:

True
```

```
In [48]:

bool('no')

Out[48]:

True
```

In [52]:

```
bool(" ")   #space is there, not empty
```

Out[52]:

```
True
```

In [51]:

```
bool('')    # empty string
```

Out[51]:

```
False
```

5. str():

We can use this method to convert other type values to str type

In [53]:

```
str(10)
```

Out[53]:

```
'10'
```

In [59]:

```
str(0b1111)
```

Out[59]:

```
'15'
```

In [54]:

```
str(10.7)
```

Out[54]:

```
'10.7'
```

In [55]:

```
str(10+34j)
```

Out[55]:

```
'(10+34j)'
```

In [56]:

```
str(True)
```

Out[56]:

```
'True'
```

In [57]:

```
str(False)
```

Out[57]:

```
'False'
```

In [58]:

```
str(true)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-58-0bd3ca74295a> in <module>
----> 1 str(true)

NameError: name 'true' is not defined
```

L22. Fundamental Data Types vs Immutablity

> All Fundamental Data types are immutable. i.e **once we creates an object,we cannot perform any changes in that object.**
>
> If we are trying to change then with those changes a new object will be created. This non-chageable behaviour is called **immutability.**

In [67]:

```
x = 10
print(id(x))
x = x+1
print(id(x))
```

```
140714560754784
140714560754816
```

In [69]:

```
x = 10
y = x
print(id(x))
print(id(y))
y = y + 1
print(x)
print(y)
print(id(x))
print(id(y))
```

```
140714560754784
140714560754784
10
11
140714560754784
140714560754816
```

L23. Fundamental Data Types vs Immutablity : Need of Immutability

Why Immutability?

Who is responsible for creating an Object in Python?

> Python Virtual Machine (PVM) is responsible for creating an object in Python.
>
> In Python if a new object is required, then PVM wont create object immediately. First it will check is any object available with the required content or not.
>
> If available then existing object will be reused. If it is not available then only a new object will be created.
>
> The advantage of this approach is **memory utilization and performance will be improved.**

In [75]:

```
a = 10
b = 10           # How many objects created? Only one (i.e.,10) with three reference varia
c = 10
print(id(a))
print(id(b))
print(id(c))
```

```
140714560754784
140714560754784
140714560754784
```

> In the last example, we have proved that all the reference variables are pointing to single object, we are used **id()** function to compare their addresses.
>
> Instead of comparing the addresses, we can use a short-cut approach also. i.e., we can make use of **is** operator.

In [18]:

```
x =15.6
y =15.6
print(x is y)         # actually it should return True, BUt this editor is not able to sup
print(id(x))          # If we try same in some standard editor like, 'atom', we will get t
print(id(y))
```

```
False
2270650549472
2270650549688
```

In [81]:

```
a = True
b = True
print(a is b)
```

```
True
```

In [84]:

```
a = 'karthi'
b = 'karthi'
print(a is b)
```

```
True
```

> But the problem in this approach is,several references pointing to the same object,by using one reference if we are allowed to change the content in the existing object then the remaining references will be effected. To prevent this immutability concept is required.
>
> According to this, once creates an object we are not allowed to change content. If we are trying to change with those changes a new object will be created.

Reference-video-31: Go through with Voter Registration application example to demonstrate the need of Immutability concept.

Date: 13-04-2020 Day-5

L24. Immutablity vs Mutablity

Object reusability concept is not applicable for complex type.

In [3]:

```
a=10+20j
b=10+20j
print(a is b)
```

```
False
```

We can execute Python program or script form **Python IDLE console or Python console also**. These things are called as **REPL** (Read Evolve Print under Loop) tools. These tools are not standard editors for executing the Python programs. thesr are used to test small code only.

In [9]:

```
a = 100
b = 100
print(a is b)
```

```
True
```

In [14]:

```
a = 256
b = 256
print(a is b)   # may be this editor range is restricted to 0 ==> 256 only
```

```
True
```

In [19]:

```
a = 257
b = 257
print(a is b)  # editor specific result
```

```
False
```

So far, we have discussed about immutability, now we will discuss about mutability with a small example.

After fundamental data types, next we need to discuss about advanced data types such as lists, tuples,dictionaries etc.,

Eg 1:

In [20]:

```
a = 10   # it represents only one value.
```

If we consider list, multiple values can be represented.

List means group of objects.

In [21]:

```
l = [10,20,30,40]     # group of values represented by a list'l'
```

How you can access these elements in the list?

By using it's index [starts from 0]

In [26]:

```
l = [10,20,30,40]
print(l[0])
print(l[1])
```

```
10
20
```

List is mutable, that means, in existing object only you can perform any modifications. This changeable behaviour is known as **Mutability**.

In [31]:

```
l = [10,20,30]
print(l)
print(id(l))
l[0]=7777    # now object 10 will replaced with 7777
print(l)
print(id(l))
```

```
[10, 20, 30]
2270650144264
[7777, 20, 30]
2270650144264
```

Here, we are getting modified content, but address is not changed. It means, all changes are being performed in existing object only, this changeable behaviour is called as **Mutability**.

Eg 2:

In [32]:

```
l1 =[10,20,30,40]
l2 = l1        # Now, l1 and l2 are pointing to same object.
```

If any refernce variable make some changes to the object, then it affects on the all reference variables pointing to the object.

In [33]:

```
l1 =[10,20,30,40]
l2 = l1
print(l1)
print(l2)
```

```
[10, 20, 30, 40]
[10, 20, 30, 40]
```

In [34]:

```
l1 =[10,20,30,40]
l2 = l1
print(l1)
print(l2)
l1[0] = 7777    # this change will be reflected in l2 also
print(l1)
print(l2)
```

```
[10, 20, 30, 40]
[10, 20, 30, 40]
[7777, 20, 30, 40]
[7777, 20, 30, 40]
```

In [35]:

```
l1 =[10,20,30,40]
l2 = l1
print(l1)
print(l2)
l1[0] = 7777    # this change will be reflected in l2 also
print(l1)
print(l2)
l2[1] = 8888    ## this change will be reflected in l1 also
print(l1)
print(l2)
```

```
[10, 20, 30, 40]
[10, 20, 30, 40]
[7777, 20, 30, 40]
[7777, 20, 30, 40]
[7777, 8888, 30, 40]
[7777, 8888, 30, 40]
```

L25. Python Data Types : List data type

In fundamental data types every variable can hold only single value.
If we want to represent a group of values (i.e., Names of all students, roll numbers of all students or mobile numbers of all students etc.,) as a single entity where **insertion order required to preserve** and **duplicates are allowed** then we should go for **list data type**.

lists can be represented by using square brackets ([ ]).

For example,

In [1]:

```
list1 = [10,20,30,45]                                     # list representation

tuple1 = (10,20,30)                                       # tuple representation

set1 = {10,20,30}                                         # set representation

dictionary1 = {100:'karthi',200:'sahasra',300:'sri'}      # dictionary representation
```

How to represent list?

In [3]:

```
l = [10,'karthi',10.5,30]
print(type(l))
print(l)
```

```
<class 'list'>
[10, 'karthi', 10.5, 30]
```

In list, in which order you specified the values in the list, in the same order the data will be stored in the memory and displayed in that order only.

Important conclusions observed with respect to list data type :

1. insertion order is preserved.
2. heterogeneous objects are allowed.
3. duplicates are allowed.
4. Growable in nature. i.e., based on our requirement you can add or remove the elements of the list.
5. values should be enclosed within square brackets.
6. Indexing concept is applicable.
7. slicing concept is applicable.
8. List is mutable (i.e., we can chnage the content of the list. It is acceptable).

In [7]:

```
l = [10,'karthi',10.5,30]
print(l[0])
print(l[-1])
print(l[1:4])      # It prints the elements of list from 1 index to 4-1 (i.e.,3) index.
```

```
10
30
['karthi', 10.5, 30]
```

In [9]:

```
l = []              # we are creating an empty list

# add an element to the list using append() method.

l.append(10)
l.append(20)
l.append(30)
l.append(40)
print(l)            # prints based on insertion order
l.remove(30)
print(l)
```

```
[10, 20, 30, 40]
[10, 20, 40]
```

In [10]:

```
l = [10,20,30,40]
l[0] = 7777
print(l)
```

```
[7777, 20, 30, 40]
```

L26. Python Data Types : Tuple data type

Tuple data type is exactly same as list data type except that it is immutable, i.e., once we create a tuple object, we cannot perform any changes in that object.

Read-only version of list is tuple.

Tuple elements can be represented within parenthesis.

In [6]:

```
t=(10,20,30,10,"karthi")        # tuple representation
print(type(t))
print(t)
print(t[0])
print(t[-1])
print(t[1:4])
t[0] = 7777
```

```
<class 'tuple'>
(10, 20, 30, 10, 'karthi')
10
karthi
(20, 30, 10)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-e04948b6d213> in <module>
      5 print(t[-1])
      6 print(t[1:4])
----> 7 t[0] = 7777

TypeError: 'tuple' object does not support item assignment
```

In [7]:

```
t=(10,20,30,10)
print(type(t))
print(t)
t.append(80)
```

```
<class 'tuple'>
(10, 20, 30, 10)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-7-39d64c0f6c0f> in <module>
      2 print(type(t))
      3 print(t)
----> 4 t.append(80)
      5 t.remove(10)

AttributeError: 'tuple' object has no attribute 'append'
```

```
In [8]:

t=(10,20,30,10)
print(type(t))
print(t)
t.remove(10)

<class 'tuple'>
(10, 20, 30, 10)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-bb416c549150> in <module>
      2 print(type(t))
      3 print(t)
----> 4 t.remove(10)

AttributeError: 'tuple' object has no attribute 'remove'
```

Note: tuple is the read only version of list

Note : Single valued tuple should compulsory ends with ','(comma).

```
In [10]:

t = ()   # Empty tuple
print(type(t))

<class 'tuple'>
```

```
In [11]:

t = (10)                # if tuple is assigned with single value, it treats as integer type.
print(type(t))        # because, in normal mathematical operations, we may specify integer va
                      # such as, 10 + 20,  (10)+ (20) and ((10)+(20))

<class 'int'>
```

```
In [14]:

t = (10,)               # if you keep ',' after value, PVM considers it as a tuple type, not i
print(type(t))

<class 'tuple'>
```

What is the difference between list and tuple?

1. List is mutable and tuple is non-mutable.
2. List elements are represented by using square brackets. Tuple elements are represented by using paranthesis.
3. To store tuple elements, Python Virtual Memory requires less memory. To store list elements, Python Virtual Memory requires more memory.

4. Tuple elements can be access within less time, because they are fixed (Performance is more). Performance is less compared with tuples.

Note :

If the content is keep on changing, better to opt list type. For example, Youtube comments or facebook comments or Udemy reviews (which are keep on changing day by day).

If the content is fixed, better to opt tuple type. For example, In Banks, account type - Only 2 values are there,

1. Savings
2. Current

At runtime account types never going to change through out Bank Project. So, to represent bank account types, better to go for tuple concept.

Some other Examples, Where allowed inputs are fixed (Best suitable type is **tuple**):

1. Vendor machines (Only accept 2/-,5/- coins only)
2. In the Metro stations also, If you want to get the tickets, you have to insert either 10/- note or 20/- note only.

L27. Python Data Types : Set data type

If we want to represent a group of values **without duplicates** and **where order is not important** then we should go for set Data Type.

For example, we want to send one SMS to all the students of a class. In this duplicate numbers are not allowed and in any order we can send SMS. At last all the students should receive the message. if you have such type of requirement, then it is better to go for **set** data type.

In general mathematics also, if the sets are like shown below

a = {1,2,3}

b = {2,3,1}

c = {1,3,2}

all these 3 sets are equal.

In [1]:

```
s = {10,20,30,40}
print(type(s))
```

```
<class 'set'>
```

In [5]:

```
s = {10,20,10,'karthi',30,40}  # in 'set' duplicate element will be ignored, which order we
print(s)
```

```
{'karthi', 40, 10, 20, 30}
```

In [6]:

```
s = {10,20,10,'karthi',30,40}
print(s[0])        # index concept not applicable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-02c4a3e6686e> in <module>
      1 s = {10,20,10,'karthi',30,40}
----> 2 print(s[0])        # index concept not applicable

TypeError: 'set' object is not subscriptable
```

In [8]:

```
s = {10,20,10,'karthi',30,40}
print(s[2:6])               # slicing concept not applicable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-dabe25494ee6> in <module>
      1 s = {10,20,10,'karthi',30,40}
----> 2 print(s[2:6])               # slicing concept not applicable

TypeError: 'set' object is not subscriptable
```

In [10]:

```
s = {10,20,30,40}
s.add(50)               # append() method applicable in list, add method applicable to set
print(s)
```

```
{40, 10, 50, 20, 30}
```

In [12]:

```
s = {10,20,30,40,50}
s.remove(30)               # remove() method applicable fro list and set to remove an elem
print(s)
```

```
{40, 10, 50, 20}
```

Important conclusions observed with respect to set data type :

1. insertion order is not preserved.
2. duplicates are not allowed.
3. heterogeneous objects are allowed.

4. index concept is not applicable.

5. It is mutable collection.

6. Growable in nature.

append() method vs add() method

In case of list, to add an element, we use **append() method** and in case of set, to add an element, we use **add() method**, **Why the names are different?**

Suppose, if you want to add 50 to the given list, where it will be added in the list?

It will be added at last position of the list.

If you are adding something at last, that operation is called as **append operation.** (i.e., existing + new content)

In [14]:

```
l = [10,20,30,40]
l.append(50)
print(l)
```

```
[10, 20, 30, 40, 50]
```

Suppose, if you want to add 50 to the given set, where it will be added in the list?

We can't say exactly where 50 will be added.

If you are adding something , if it is not guaranteed to add at end, then we can't say that it is an append operation. That's why when you are adding an element to a set, we use **add() method**.

In [15]:

```
s = {10,20,30,40}
s.add(50)
print(s)
```

```
{40, 10, 50, 20, 30}
```

One more important point regarding **set data type**

In [16]:

```
s = {}    # it is not empty set, it is empty dictionary
print(type(s))
```

```
<class 'dict'>
```

Why it is considered as 'dict' type?

Among **set** and **dictionary**, dictionary is the most frequently used data type compared with set data type. That's why Python gave the priority to the dictionary type.

How can we create empty set ?

In [18]:

```
s = set()                    # we required to use set() function to create an empty set.
print(type(s))
print(s)
```

```
<class 'set'>
set()
```

What is the difference between set and list data types?

1. In list, order is preserved, but in set order is not preserved.
2. Duplicate elements are allowed in list, but duplicate elements are not allowed in set.
3. List can be represented by using square brackets and set can be represented by using curly braces.

Date: 14-04-2020 - Day 6

L28. Python Data Types : Frozen set

In general, meaning of frozen is freezing, i.e, No one going to change, can't move, fixed ets.,

Frozen set is exactly same as set except that it is immutable. Hence we cannot use add or remove functions.

In [3]:

```
s = {10,20,30,40}
fs = frozenset(s)
print(type(fs))
```

```
<class 'frozenset'>
```

In [2]:

```
s = {10,20,30,40}
fs = frozenset(s)
print(type(s))
fs.add(50)
```

```
<class 'set'>

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-2-8ccea297b8d2> in <module>
      2 fs = frozenset(s)
      3 print(type(s))
----> 4 fs.add(50)
      5 fs.remove(30)

AttributeError: 'frozenset' object has no attribute 'add'
```

In [4]:

```
s = {10,20,30,40}
fs = frozenset(s)
print(type(s))
fs.remove(30)
```

```
<class 'set'>

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-4-6114db23b7ef> in <module>
      2 fs = frozenset(s)
      3 print(type(s))
----> 4 fs.remove(30)

AttributeError: 'frozenset' object has no attribute 'remove'
```

What is the difference between Frozen set and tup;e data types?

1. In tuple, order is preserved, but in frozen set order is not applicable.
2. In tuple duplicate elements are allowed, but in frozen set duplicates are not allowed.
3. Index, slice concepts are applicable, but in frozen set index,slice concepts are not applicable.

Note : The only similarity between tuple and frozen set is, both are immutable.

L29. Python Data Types : Dictonary (Dict)

So far, we have discussed about list,tuple,set,frozen set. All these data types are having some common point is,

```
            - list       [10,20,30,40]

            - tuple      (10,20,30,40)

            - set        {10,20,30,40}

            -forzen set  frozenset({10,20,30,40})
```

If you observed this, all thase data types are talks about a group of individual values.

Some times, If we want to represent a group of values as **key-value pairs** then we should go for **dict** data type.

-Eg:

```
    - roll Number - name

    - mobile number - address
```

In general, In our english dictionary we found,

Word : Meaning

The same thing will happensin Python dictionary also.

How can you represent dictionary in Python?

d = {key1 : value1, key2 : value2, key3 : value3}

You van take any number of key-value pairs

In [5]:

```
d = {100:"karthi", 101:'sahasra', 150:'guido'}
print(type(d))
print(d)
```

```
<class 'dict'>
{100: 'karthi', 101: 'sahasra', 150: 'guido'}
```

We can create an empty dictionary, and later we can add key-value pairs into that dictionary.

If you want to add key-value pair into a dictionary, the following syntax is used in Python:

dictionaryName[key] = value

In [8]:

```
d = {}          # Empty dictionary
print(type(d))
d[100] = 'karthi'
d[200] = 'ravi'
print(d)             # No guarantee for order wise printing in the dictionary
```

```
<class 'dict'>
{100: 'karthi', 200: 'ravi'}
```

Key points with respect to Dictionary:

1. Duplicate keys are not allowed but values can be duplicated.
2. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.

In [10]:

```
d={10:'karthi',20:'karthi',30:'karthi'}
print(d)
```

```
{10: 'karthi', 20: 'karthi', 30: 'karthi'}
```

In [11]:

```
d={10:'karthi',10:'karthi',30:'karthi'}
print(d)
```

```
{10: 'karthi', 30: 'karthi'}
```

In [14]:

```
d={10:'karthi',10:'sahasra',30:'ravi'}
print(d)
```

```
{10: 'sahasra', 30: 'ravi'}
```

In [15]:

```
d={10:'sahasra',10:'karthi',30:'ravi'}
print(d)
```

```
{10: 'karthi', 30: 'ravi'}
```

```
In [9]:
```

```
d = {}          # Empty dictionary
print(type(d))
d[100] = 'karthi'
d[200] = 'ravi'
print(d)
d[100] = 'shiva'
print(d)
```

```
<class 'dict'>
{100: 'karthi', 200: 'ravi'}
{100: 'shiva', 200: 'ravi'}
```

Important conclusions observed with respect to dict data type :

1. If you want to represent a group of values as key-value pairs then we should go for dict data type.
2. Order is not applicable.
3. Duplicate keys are not allowed but values can be duplicated.
4. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.
5. Heterogeneous objects are allowed.
6. It is mutable.
7. Indexing, slicing is not applicable.

L30. Python Data Types : range

range Data Type represents a sequence of numbers.

range() is the in-built function of Python.

The elements present in range Data type are not modifiable. i.e range Data type is immutable.

```
In [3]:
```

```
r = range(10)        # it represents the sequence of values from 0 to 9
print(type(r))
print(r)
```

```
<class 'range'>
range(0, 10)
```

How you can print the values present in the given range?

We have to make use of loops, such as for, while etc., to display the elements in the given range.

In [4]:

```
r = range(10)        # it represents the sequence of values from 0 to 9
print(type(r))
print(r)

for x in r:
    print(x)
```

```
<class 'range'>
range(0, 10)
0
1
2
3
4
5
6
7
8
9
```

In [6]:

```
r = range(10)        # it represents the sequence of values from 0 to 9
print(type(r))
print(r)

for x in r:
    print(x,end =' ')   # to print the values horizantally
```

```
<class 'range'>
range(0, 10)
0 1 2 3 4 5 6 7 8 9
```

How to create range object? what are the various options are available?

Option 1:

range(n) => It represents the sequence of numbers from 0 to n-1

In [9]:

```
print(range(10))    # it represnts the sequence of numbers from 0 to 9

range(100)   # it represnts the sequence of numbers from 0 to 99
```

```
range(0, 10)
```

Out[9]:

```
range(0, 100)
```

Option 2:

Sometimes, our requirement is, We don't want from 0,we want to print the numbers from any specific number.

range(begin,end) ==> it represents thesequence of numbers from begin to end-1.

In [11]:

```
r = range(1,11)
for x in r:
    print(x,end = ' ')
```

```
1 2 3 4 5 6 7 8 9 10
```

Option 3:

range(begin,end,increment/decrement value) ==> It represents the sequence of numbers from begin to end by increment/decrement value

In [12]:

```
r = range(1,21,1)
for x in r:
    print(x,end = ' ')
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

In [13]:

```
r = range(1,21,2)
for x in r:
    print(x,end = ' ')
```

```
1 3 5 7 9 11 13 15 17 19
```

In [14]:

```
r = range(1,21,3)
for x in r:
    print(x,end = ' ')
```

```
1 4 7 10 13 16 19
```

In [15]:

```
r = range(1,21,4)
for x in r:
    print(x,end = ' ')
```

```
1 5 9 13 17
```

In [18]:

```
r = range(20,1,-5)
for x in r:
    print(x,end = ' ')
```

```
20 15 10 5
```

We can access elements present in the range Data Type by using index.

In [22]:

```
r = range(10,21)
print(r[0])
print(r[-1])
r1 = r[1:5]
print(r1)
for x in r1:
    print(x)
```

```
10
20
range(11, 15)
11
12
13
14
```

Once order is important, obviously indexing, slicing concepts are also applicable.

range object is immutable.

In [23]:

```
r = range(10,21)
print(r[0])
print(r[-1])
r1 = r[1:5]
print(r1)
for x in r1:
    print(x)
r[1] = 100
```

```
10
20
range(11, 15)
11
12
13
14
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-23-980d29ad463b> in <module>
      6 for x in r1:
      7     print(x)
----> 8 r[1] = 100

TypeError: 'range' object does not support item assignment
```

We can create a list of values with range data type

In [24]:

```
l = list(range(10))
print(l)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Important conclusions observed with respect to range data type :

1. range Data Type represents a sequence of numbers.
2. Different forms of ragne data type are as follows:

```
    1. range with one argument   -    range(10)

    2. range with Two arguments   -    range(10,21)

    3. range with three arguments   -    range(10,21,2)
```

3. Once order is important, obviously indexing, slicing concepts are also applicable.
4. range object is immutable.

L31. Python Data Types : bytes and bytearray

1. bytes data type:

It's not that much frequently used data type in Python.

bytes data type represents a group of byte numbers just like an array.

In [25]:

```
l = [10,20,30,40]
b = bytes(l)      # If you wnat to create bytes object, you have to use in-built function '
print(type(b))
```

```
<class 'bytes'>
```

Now, we want to print all values present inside 'b'.

In [26]:

```
l = [10,20,30,40]
b = bytes(l)
for x in b:
    print(x)
```

```
10
20
30
40
```

Where this type of data type is helpful?

If you want to handle binary data, like images, video files and audio files, we need to make use of byte and bytearray data types.

Two important conclusions about bytes data type:

Conclusion 1:

The only allowed values for byte data type are 0 to 255. By mistake if we are trying to provide any other values then we will get value error.

```
In [27]:

l = [10,20,30,40,256]
b = bytes(l)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-735598b68758> in <module>
      1 l = [10,20,30,40,256]
----> 2 b = bytes(l)

ValueError: bytes must be in range(0, 256)
```

```
In [29]:

l = [10,20,30,40,255]
b = bytes(l)
```

Conclusion 2:

Once we creates bytes data type value, we cannot change its values,otherwise we will get TypeError. i.e., **Immutable**

```
In [30]:

l = [10,20,30,40]
b = bytes(l)
print(b[0])
b[0] = 100

10

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-1ccbd30578fd> in <module>
      2 b = bytes(l)
      3 print(b[0])
----> 4 b[0] = 100

TypeError: 'bytes' object does not support item assignment
```

2. bytearray data type:

bytearray is exactly same as bytes data type except that its elements can be modified. i.e.,**Mutable**

In [35]:

```
l = [10,20,30,40]
b = bytearray(l)
print(type(b))
for i in b:
    print(i)
```

```
<class 'bytearray'>
10
20
30
40
```

In [36]:

```
l = [10,20,30,40]
b = bytearray(l)
for i in b:
    print(i)
print(b[0])
b[0] = 288
print(b[0])
```

```
10
20
30
40
10

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-36-3a54375d30ae> in <module>
      4     print(i)
      5 print(b[0])
----> 6 b[0] = 288
      7 print(b[0])

ValueError: byte must be in range(0, 256)
```

In [37]:

```
l = [10,20,30,40]
b = bytearray(l)
for i in b:
    print(i)
print(b[0])
b[0] = 188
print(b[0])
```

```
10
20
30
40
10
188
```

In [38]:

```
l = [10,20,30,40]
b = bytearray(l)
b[0]= 77
for i in b:
    print(i)
```

```
77
20
30
40
```

L32. Python Data Types : None

None means Nothing or No value associated.

> In Python, there are some situations, where, if the value is not available,then to handle such type of cases None introduced.
>
> It is something like null value in Java.
>
> To make an object eligible for garbage collection, we can use None type.

In [1]:

```
a = 10              # 'a' is pointing to object 10
a = None            # 'a' is not pointing to object 10, it is pointing to none or nothing
```

In [2]:

```
def f1():
    return 10

x = f1()
print(x)
```

```
10
```

The above code is clear.

In [3]:

```
def f1():
    print("Hello")       # Here, 'f1()' function is not going to return any value

x = f1()
print(x)
```

```
Hello
None
```

In the above code, If the function won't return any statement, then how you can handle such situation is, internally it is going to represent **None**.

> **None** is also an object in Python.

In [5]:

```
a = None         # 'a' is not pointing any value
print(id(a))
print(type(a))
```

```
140721283812576
<class 'NoneType'>
```

How many None objects are there in Python?

> Throuhg out Python, only one None object is avilable. If you are using any number of references to None,all the references are pointing to the same object only.

In [6]:

```
a = None
b = None
c = None

def f1():
    pass             # empty body of a function represented using 'pass' statement
d = f1()             # 'd' internally contains None only.
print(id(a))
print(id(b))
print(id(c))
print(id(d))
```

```
140721283812576
140721283812576
140721283812576
140721283812576
```

```
Above code in another way.
```

```
In [7]:

a = None
b = None
c = None

def f1():
    pass              # empty body of a function represented using 'pass' statement
d = f1()              # 'd' internally contains None only.
print(id(a),id(b),id(c),id(d))
```

```
140721283812576 140721283812576 140721283812576 140721283812576
```

L33. Escape characters, Comments and Constants

Escape Characters:

> In String literals we can use esacpe characters to associate a special meaning.

The following are various important escape characters in Python:

1. \n==>New Line
2. \t===>Horizontal tab
3. \r ==>Carriage Return (suppose, in a line currentlycursor is locating at some place, i want to move it to the beginning of the same line, we go for carriage return)
4. \b===>Back space
5. \f===>Form Feed (to go to next page)
6. \v==>Vertical tab
7. '===>Single quote
8. "===>Double quote
9. \===>back slash symbol

```
In [12]:

print("RGMcollege")
print("RGM\tcollege")
print("RGM\ncollege")
print('This is ' symbol')
```

```
  File "<ipython-input-12-48bc15eca10a>", line 4
    print('This is ' symbol')
                          ^
SyntaxError: invalid syntax
```

In [13]:

```
print("RGMcollege")
print("RGM\tcollege")
print("RGM\ncollege")
print('This is \' symbol')
```

```
RGMcollege
RGM     college
RGM
college
This is ' symbol
```

In [17]:

```
print("RGMcollege")
print("RGM\tcollege")
print("RGM\ncollege")
print('This is \' symbol')
print('This is \" symbol')
```

```
RGMcollege
RGM     college
RGM
college
This is ' symbol
This is " symbol
```

In [19]:

```
print("RGMcollege")
print("RGM\tcollege")
print("RGM\ncollege")
print('This is \' symbol')
print('This is \" symbol')
print('This is \\ symbol')
```

```
RGMcollege
RGM     college
RGM
college
This is ' symbol
This is " symbol
This is \ symbol
```

Comments

// single line comment in java

/* abc def

```
    --------   Multiline comments in Java or C
```

*/

single line comment in Python

'#' used for single line comments in python programming

Multiline comments:

Multi line comments are not available in Python.

If you have multiple lines are there to comment, use '#' at every line.

In [21]:

```
#print('This a comment, it won't be executed by PVM)
print('This is \\ symbol')
```

```
This is \ symbol
```

In [22]:

```
#print('This a comment, it won't be executed by PVM)
#print('This is \\ symbol')
```

Constants in Python:

Constants concept is not applicable in Python.

But it is convention to use only uppercase characters if we don’t want to change value.

`MAX_VALUE=10`

It is just convention but we can change the value.

In [ ]:

Python Operators

Date: 16-04-2020 Day 1

Introduction

In this lecture, we will discuss about various Python Operators. In general, the person who is doing some operation is known as operator, such as Telephone operator, Camera operator etc.,. In same way, the Python symbol, which is used to perform certain activity is known as operator.

The following topics we are going to discuss as part of this lecture:

1. Arithmetic Operators

2. Relational or Comparison Operators

3. Equality OPerators

4. Logical Operators

5. Bitwise Operators

6. Shift Operators

7. Assignment Operator

8. Ternary Operator (or) Conditional Operator

9. Special Operators

```
 i) Identity Operators

ii) Membership Operators
```

10. Operator Precedence

11. Mathematical functions using math module

These are the 11 topics, we are going to discuss as part of the Python Operators concept.

1. Arithmetic Operators

Following are the arithmetic operators (**7**) used in Python:

1. Addition ==> +
2. Subtraction ==> -
3. Multiplication ==> *
4. Normal Division ==> /
5. Modulo Division ==> %

In addition to these common arithmetic operators, Python supports two more special arithmetic operators:

6. Floor Division ==> //

7. Exponential Operator (or) Power Operator ==> **

In [1]:

```
a = 10
b = 2
print(a+b)
print(a-b)
print(a*b)
print(a%b)
```

```
12
8
20
0
```

In [2]:

```
a = 10
b = 3
print(a+b)
print(a-b)
print(a*b)
print(a%b)
```

```
13
7
30
1
```

Floor Division

suppose 10.3 is there, what is the floor value of 10.3?

- Answer is **10**

What is the ceil value of 10.3?

- Answer is **11**

Eg:1

In [3]:

```
print(10/2)    # In Python division operation always meant for floating point arithmetic ar
               # gives floating point value as it's result.
               # This is Python 3 specific behavior, In Python 2 you are going to get 5 as
```

```
5.0
```

In [4]:

```
print(10/3)
```

```
3.3333333333333335
```

- If you want to get integer value as result of division operation, you need to make use of floor division(//) operator.
- floor division(//) operator meant for integral arithmetic operations as well as floating point arithmetic operations.
- **The result of floor division(//) operator can be always floor value of either integer value or float value** based on your arguments.
- If both arguments are 'int' type, then the result is 'int' type.
- If atleast one of the argument is float type, then the result is also float type.

In [5]:

```
print(10//2)
```

```
5
```

In [6]:

```
print(10/3)
```

```
3.3333333333333335
```

In [7]:

```
print(10.0/3)
```

```
3.3333333333333335
```

In [8]:

```
print(10.0//3)
```

```
3.0
```

In [9]:

```
print(10//3)
```

```
3
```

In [10]:

```
print(10.0//3.0)
```

```
3.0
```

Eg 2:

NOTE:

- Floor integer value of 3.33333 is 3
- Floor float value of 3.33333 is 3.0
- Floor integer value of 5.9997777 is 5
- Floor float value of 5.9997777 is 5.0

In [11]:

```
print(20/2)
print(20.5/2)
print(20//2)
print(20.5//2)
print(30//2)
print(30.0//2)
```

```
10.0
10.25
10
10.0
15
15.0
```

Power Operator or Exponential Operaor :

In [12]:

```
print(10**2)    # meaning of this is 10 to the power 2
print(3**3)
```

```
100
27
```

Note:

- We can use +,* operators for str type also.
- **If we want to use + operator for str type then compulsory both arguments should be str type only otherwise we will get error.**

In [13]:

```
print(10 + 20)
print("karthi" + "sahasra")
print("karthi" + "10")
print("sahasra" + 10)
```

```
30
karthisahasra
karthi10

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-f0ce7b898dfa> in <module>
      2 print("karthi" + "sahasra")
      3 print("karthi" + "10")
----> 4 print("sahasra" + 10)

TypeError: can only concatenate str (not "int") to str
```

- If we use * operator for str type then compulsory one argument should be 'int' and other argument should be 'str' type.

In [19]:

```
print("karthi" * 3)
print(3 * "karthi")
print(2.5 * 'karthi')
```

```
karthikarthikarthi
karthikarthikarthi

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-8c0cbe6387f0> in <module>
      1 print("karthi" * 3)
      2 print(3 * "karthi")
----> 3 print(2.5 * 'karthi')

TypeError: can't multiply sequence by non-int of type 'float'
```

In [20]:

```
print('karthi' * 'sahasra')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-f72e4a5aef05> in <module>
----> 1 print('karthi' * 'sahasra')

TypeError: can't multiply sequence by non-int of type 'str'
```

In [21]:

```
print('karthi' * '3')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-49b3a25e7226> in <module>
----> 1 print('karthi' * '3')

TypeError: can't multiply sequence by non-int of type 'str'
```

In [22]:

```
print('karthi' * int('3'))
```

```
karthikarthikarthi
```

Note :

- ====> '+' operotor for String (Concatenation Operator)
- ====> '*' Operator for String (String Multiplication Operator (or) String Repetetion Operator)

Date: 17-04-2020 Day-2

For any number **x**,

x/0 or x//0 or x%0 always raises **"ZeroDivisionError"**

In [23]:

```
10/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-23-e574edb36883> in <module>
----> 1 10/0

ZeroDivisionError: division by zero
```

In [24]:

```
10.0/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-24-e796260e48b5> in <module>
----> 1 10.0/0

ZeroDivisionError: float division by zero
```

In [25]:

```
10//0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-25-adb5753def1b> in <module>
----> 1 10//0

ZeroDivisionError: integer division or modulo by zero
```

In [26]:

```
10.0//0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-26-2ac47ac31328> in <module>
----> 1 10.0//0

ZeroDivisionError: float divmod()
```

In [27]:

```
10%0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-27-3e1bfe6920c0> in <module>
----> 1 10%0

ZeroDivisionError: integer division or modulo by zero
```

In [28]:

```
10.0%0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-28-8864e72cec86> in <module>
----> 1 10.0%0

ZeroDivisionError: float modulo
```

Arithmetic Operators with bool type

- Internally Boolean values are represented as integer values only.

In [29]:

```
print("karthi" * True)   #True => 1
```

```
karthi
```

In [30]:

```
print("karthi" * False)    #False => 0, Output is an empty string
```

2. Relational Operators (or) Comparison Operators

Following are the relational operators usedin Python:

1. Less than (<)

2. Greater than (>)

3. Leass than or Equal to (<=)

4. Greater than or Equal to (>=)

i) We can apply relational operators for number types,

Eg 1:

In [31]:

```
a = 10
b = 20
print('a < b is', a<b)
print('a <= b is', a<=b)
print('a > b is', a>b)
print('a >= b is', a>=b)
```

```
a < b is True
a <= b is True
a > b is False
a >= b is False
```

ii) We can apply relational operators for **'str' type also, here comparison is performed based on ASCII or Unicode values.**

Eg 2:

How to know the Unicode or ASCII value of any character?

- By using **ord()** function, we can get the ASCII value of any character.

In [32]:

```
print(ord('a'))
print(ord('A'))
```

```
97
65
```

- If you know the ASCII value and to find the corresponding character, you need to use the **chr()** function.

In [33]:

```
print(chr(97))
print(chr(65))
```

```
a
A
```

In [34]:

```
s1 = 'karthi'       # ASCII value of 'a' is 97
s2 = 'sahasra'      # ASCII value of 'b' is 98
print(s1<s2)
print(s1<=s2)
print(s1>s2)
print(s1>=s2)
```

```
True
True
False
False
```

In [35]:

```
s1 = 'karthi'
s2 = 'karthi'
print(s1<s2)
print(s1<=s2)
print(s1>s2)
print(s1>=s2)
```

```
False
True
False
True
```

In [36]:

```
s1 = 'karthi'
s2 = 'Karthi'
print(s1<s2)
print(s1<=s2)
print(s1>s2)
print(s1>=s2)
```

```
False
False
True
True
```

iii) We can apply relational operators evevn for boolean types also.

Eg 3:

In [37]:

```
print(True > False)
print(True >= False)          # True ==> 1
print(True < False)           # False ==> 0
print(True <= False)
```

```
True
True
False
False
```

In [38]:

```
print(10 > 'karthi')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-e2ae37134b58> in <module>
----> 1 print(10 > 'karthi')

TypeError: '>' not supported between instances of 'int' and 'str'
```

Eg 4:

In [39]:

```
a = 10
b = 20
if a>b:
    print('a is greater than b')
else:
    print('a is not greater than b')
```

```
a is not greater than b
```

iv) Chaining of relational operatrs:

- Chaining of relational operators is possible.

- In the chaining, if all comparisons returns True then only result is True.
- If atleast one comparison returns False then the result is False.

Eg 5:

In [40]:

```
print(10<20)             # ==>True
print(10<20<30)          # ==>True
print(10<20<30<40)       # ==>True
print(10<20<30<40>50)    # ==>False
```

```
True
True
True
False
```

3. Equality Operators:

Equality operators are used to check whether the given two values are equal or not. The following are the equality operators used in Python.

1. Equal to (==)
2. Not Equal to (!=)

In [41]:

```
print(10==20)
print(10!=20)
```

```
False
True
```

In [42]:

```
print(1==True)
print(10==10.0)
print('karthi'=='karthi')
```

```
True
True
True
```

We can apply these operators for any type even for incompatible types also.

In [43]:

```
print(10=='karthi')
```

```
False
```

In [44]:

```
print(10=='10')
```

```
False
```

Note:

- Chaining concept is applicable for equality operators.
- If atleast one comparison returns False then the result is False. otherwise the result is True.

Eg:

In [45]:

```
print(10==20==30==40)
print(10==10==10==10)
```

```
False
True
```

Date: 18-04-2020 Day 3

Q) What is the Difference between '==' and 'is' operators?

is Operator:

- 'is' operator meant for reference or address comparison.
- When **a is b** returns true?

Ans: Whenever 'a' and 'b' pointing to the same object, then only 'a is b' returns true, which is nothing but reference comparison (or) Address comparison.

== Operator :

- '==' is meant for content comparison.

In [46]:

```
l1 = [10,20,30]
l2 = [10,20,30]
print(id(l1))
print(id(l2))
print(l1 is l2)   # False
print(l1 == l2)   # True
l3 = l1           # l3 is also pointing to l1
print(id(l3))
print(l1 is l3)   # True
print(l1 == l3)   # True
```

```
2689910878664
2689910921864
False
True
2689910878664
True
True
```

4. Logical operators

Following are the various logical operators used in Python.

1. and
2. or
3. not

You can apply these operators for boolean types and non-boolean types, butthe behavior is different.

For boolean types:

- and ==>If both arguments are True then only result is True
- or ====>If atleast one arugemnt is True then result is True
- not ==>complement

i) 'and' Operator for boolean type:

- If both arguments are True then only result is True

In [14]:

```
print(True and True)
print(True and False)
print(False and True)
print(False and False)
```

```
True
False
False
False
```

ii) 'or' Operator for boolean type:

- If both arguments are True then only result is True.

In [15]:

```
print(True or True)
print(True or False)
print(False or True)
print(False or False)
```

```
True
True
True
False
```

iii) 'not' Operator for boolean type:

- Complement (or) Reverse

In [16]:

```
print(not True)
print(not False)
```

```
False
True
```

Eg :

Now we will try to develop a small authentication application with this knowledge.

- we will read user name and password from the keyboard.
- if the user name is karthi and password is sahasra, then that user is valid user otherwise invalid user.

In [17]:

```
userName = input('Enter User Name : ')
password = input('Enter Password : ')

if userName == 'karthi' and password == 'sahasra':
    print('valid User')
else:
    print('invalid user')
```

```
Enter User Name : karthi
Enter Password : rgm
invalid user
```

In [18]:

```
userName = input('Enter User Name : ')
password = input('Enter Password : ')

if userName == 'karthi' and password == 'sahasra':
    print('valid User')
else:
    print('invalid user')
```

```
Enter User Name : karthi
Enter Password : sahasra
valid User
```

For non-boolean types behaviour:

Note :

- 0 means False
- non-zero means True
- empty strings, list,tuple, set,dict is always treated as False

i) X and Y

Here, X and Y are non boolean types and the result may be either X or Y but not boolean type (i.e., The result is always non boolean type only).

- **if 'X' is evaluates to false then the result is 'X'.**
- **If 'X' is evaluates to true then the result is 'Y'.**

In [53]:

```
print(10 and 20)
print(0 and 20)
print('karthi' and 'sahasra')
print('' and 'karthi')      # first argument is empty string
print(' ' and 'karthi')     # first argument contains space character, so it is not empty
print('karthi' and '')      # second argument is empty string
print('karthi' and ' ')     # second argument contains space character, so it is not empty
```

```
20
0
sahasra

karthi
```

ii) X or Y

Here, X and Y are non boolean types and the result may be either X or Y but not boolean type (i.e., The result is always non boolean type only).

- **if 'X' is evaluates to true then the result is 'X'.**
- **If 'X' is evaluates to false then the result is 'Y'.**

In [54]:

```
print(10 or 20)
print(0 or 20)
print('karthi' or 'sahasra')
print('' or 'karthi')      # first argument is empty string
print(' ' or 'karthi')     # first argument contains space character, so it is not empty
print('karthi' or '')      # second argument is empty string
print('karthi' or ' ')     # second argument contains space character, so it is not empty
```

```
10
20
karthi
karthi

karthi
karthi
```

iii) not X:

Even you apply **not** operator for non boolean type, the result is always boolean type only.

- **If X is evalutates to False then result is True otherwise False**

In [55]:

```
print(not 'karthi')
print(not '')
print(not 0)
print(not 10)
```

```
False
True
True
False
```

5. Bitwise Operators

- We can apply these operators bit by bit.
- These operators are applicable only for **int** and **boolean** types. By mistake if we are trying to apply for any other type then we will get Error.

Following are the various bitwise operators used in Python:

1. Bitwise and (&)
2. Bitwise or (|)
3. Bitwise ex-or (^)
4. Bitwise complement (~)
5. Bitwise leftshift Operator (<<)
6. Bitwise rightshift Operator(>>)

In [56]:

```
print(10.5 & 20.6)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-56-d0942894908d> in <module>
----> 1 print(10.5 & 20.6)

TypeError: unsupported operand type(s) for &: 'float' and 'float'
```

In [57]:

```
print('karthi' | 'karthi')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-57-482742ac27fc> in <module>
----> 1 print('karthi' | 'karthi')

TypeError: unsupported operand type(s) for |: 'str' and 'str'
```

In [61]:

```
print(bin(10))
print(bin(20))

print(10 & 20)    # Valid
print(10.0 & 20.0)  # In valid
```

```
0b1010
0b10100
0


---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-61-0123c653392c> in <module>
      3 
      4 print(10 & 20)    # Valid
----> 5 print(10.0 & 20.0)  # In valid

TypeError: unsupported operand type(s) for &: 'float' and 'float'
```

In [62]:

```
print(True & False)
```

```
False
```

In [63]:

```
print(True | False)
```

```
True
```

Behavior of Bitwise Operators

& ==> If both bits are 1 then only result is 1 otherwise result is 0

| ==> If atleast one bit is 1 then result is 1 otherwise result is 0

^ ==>If bits are different then only result is 1 otherwise result is 0

~ ==> bitwise complement operator, i.e 1 means 0 and 0 means 1

<< ==> Bitwise Left shift Operataor

Bitwise Right Shift Operator ==> >>

In [66]:

```
print(4 & 5)     # 100 & 101
print(4 | 5)     # 100 | 101
print(4 ^ 5)     # 100 ^ 101
```

```
4
5
1
```

Bitwise Complement Operator (~):

- We have to apply complement for total bits.

In [67]:

```
print(~4)    # 4 ==>  100
```

```
-5
```

Here, we have to apply complement for total bits, not for three bits (in case of 4). In Python minimum 32 bits required to represent an integer.

Note:

- The most significant bit acts as sign bit. 0 value represents +ve number where as 1 represents -ve value.
- Positive numbers will be repesented directly in the memory where as Negative numbers will be represented indirectly in 2's complement form.

How you can find two's complement of a number?

- To find Two's complement of a number, first you need to find One's complement of that number and add 1 to it.
- One's complement ==> Interchange of 0's and 1's

Eg.

In [68]:

```
print(~5)
```

```
-6
```

In [69]:

```
print(~-4)    # negative values are stored in the memory in 2's complement form.
```

```
3
```

6. Shift Operators

Following are the various shift operators used in Python:

1. Left Shift Operator (<<)
2. Right Shift Operator (<<)

1. Left Shift Operator (<<):

- After shifting the bits from left side, empty cells to be filled with zero.

In [1]:

```
print(10<<2)
```

```
40
```

2. Right Shift Operator (<<)

- After shifting the empty cells we have to fill with sign bit.(0 for +ve and 1 for -ve)

In [2]:

```
print(10>>2)
```

```
2
```

We can apply bitwise operators for boolean types also.

In [7]:

```
print(True & False)
print(True | False)
print(True ^ False)
print(~True)
print(~False)
print(True<<2)
print(True>>2)
```

```
False
True
True
-2
-1
4
0
```

7. Assignment Operator

- We can use assignment operator to assign value to the variable.

Eg :

In [8]:

```
x = 2
```

- We can combine asignment operator with some other operator to form **compound assignment operator.**

Eg :

x+=10 ====> x = x+10

In [10]:

```
x = 10
x += 20      # x =  x + 20
print(x)
```

```
30
```

The following is the list of all possible compound assignment operators in Python:

+=

-=

*=

/=

%=

//=

**=

&=

|=

^=

<<= and >>=

In [12]:

```
x = 10    # 1010
x &= 5    # 0101
print(x)
```

```
0
```

In [15]:

```
x = 10
x **= 2      # x = x**2
print(x)
```

```
100
```

Now, we want to discuss abou one loop hole in Python Operators. Let us consider the following example,

Case 1:

In [16]:

```
x = 10
x++
print(x)
```

```
  File "<ipython-input-16-60ef4d605145>", line 2
    x++
       ^
SyntaxError: invalid syntax
```

Case 2:

In [17]:

```
x = 10
x--
print(x)
```

```
  File "<ipython-input-17-b058e79bff01>", line 2
    x--
       ^
SyntaxError: invalid syntax
```

In both the cases, we are getting syntax error, because in Python increment/decrement operators concept is not there.

Let us see the following code

In [21]:

```
x = 10
print(++x)
print(++++x)    # Here, + and - are sign bits, not increment and decrement operators
print(-x)
print(--x)
print(++++++++++++-x)
print(-------------+x)
```

```
10
10
-10
10
-10
-10
```

8. Ternary Operator (or) Conditional Operator

Note:

1. If the operator operates on only one operand, we will call such operator as **unary** operator. For eg:, ~a.
2. If the operator operates on Two operands, we will call such operator as **binary** operator. For eg:, a + b.
3. If the operator operates on Three operands, we will call such operator as **Ternary** operator.

Syntax:

x = firstValue if condition else secondValue

- If condition is True then firstValue will be considered else secondValue will be considered.

Eg 1:

In [22]:

```
a,b=23,43                       # a =23  b = 43
c = 50 if a>b else 100
print(c)
```

```
100
```

Eg 2: Read two integer numbers from the keyboard and print minimum value using ternary operator.

In [26]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))

min=a if a<b else b

print("Minimum Value:",min)
```

```
Enter First Number:255
Enter Second Number:22
Minimum Value: 22
```

Nesting of ternary operator is possible.

Eg 3: Program for finding minimum of 3 numbers using nesting of ternary operators

In [27]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))

min= a if a<b and a<c else b if b<c else c

print("Minimum Value:",min)
```

```
Enter First Number:10
Enter Second Number:20
Enter Third Number:30
Minimum Value: 10
```

In [28]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))

min= a if a<b and a<c else b if b<c else c

print("Minimum Value:",min)
```

```
Enter First Number:-10
Enter Second Number:-20
Enter Third Number:-30
Minimum Value: -30
```

In [29]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))

min= a if a<b and a<c else b if b<c else c

print("Minimum Value:",min)
```

```
Enter First Number:30
Enter Second Number:10
Enter Third Number:20
Minimum Value: 10
```

Now, We will write the above program with some small modification.

In [30]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))

# min= a if a<b and a<c else b if b<c else c

min = a if a<b<c else b if b<c else c

print("Minimum Value:",min)
```

```
Enter First Number:10
Enter Second Number:20
Enter Third Number:30
Minimum Value: 10
```

But,the above logic fails for some test data, which is taken below:

In [31]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))

# min= a if a<b and a<c else b if b<c else c

min = a if a<b<c else b if b<c else c

print("Minimum Value:",min)
```

```
Enter First Number:5
Enter Second Number:35
Enter Third Number:30
Minimum Value: 30
```

Eg 4: Program for finding maximum of 3 numbers

In [32]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))

max=a if a>b and a>c else b if b>c else c

print("Maximum Value:",max)
```

```
Enter First Number:34
Enter Second Number:22
Enter Third Number:55
Maximum Value: 55
```

In [33]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))

max=a if a>b and a>c else b if b>c else c

print("Maximum Value:",max)
```

```
Enter First Number:-10
Enter Second Number:-20
Enter Third Number:-30
Maximum Value: -10
```

In [34]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))

max=a if a>b and a>c else b if b>c else c

print("Maximum Value:",max)
```

```
Enter First Number:10
Enter Second Number:30
Enter Third Number:20
Maximum Value: 30
```

Eg 5: Assume that there are two numbers, x and y, whose values to be read from the keyboard, and print the following outputs based on the values of x and y.

case 1: If both are equal, then the output is : Both numbers are equal

case 2: If first number is smaller than second one, then the output is: First Number is Less than Second Number

case 3: If the firts number is greater than second number, then the output is : First Number Greater than Second Number

In [46]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))

print("Both numbers are equal" if a==b else "First Number is Less than Second Number"
if a<b else "First Number Greater than Second Number")
```

```
Enter First Number:10
Enter Second Number:10
Both numbers are equal
```

In [47]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))

print("Both numbers are equal" if a==b else "First Number is Less than Second Number"
if a<b else "First Number Greater than Second Number")
```

```
Enter First Number:10
Enter Second Number:20
First Number is Less than Second Number
```

In [48]:

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))

print("Both numbers are equal" if a==b else "First Number is Less than Second Number"
if a<b else "First Number Greater than Second Number")
```

```
Enter First Number:20
Enter Second Number:12
First Number Greater than Second Number
```

Date: 19-04-2020 Day 4

9. Special Operators

There are two types of special operators are there in Python:

1. Identity Operators

2. Membership Operators

1. Identity Operators

We can use identity operators for address comparison. There are two identity operators used in Python:

i) is

ii) is not

- r1 **is** r2 returns True if both r1 and r2 are pointing to the same object.

- r1 **is not** r2 returns True if both r1 and r2 are not pointing to the same object.

Eg :

In [2]:

```
a=10
b=10
print(a is b)
x=True
y=True
print( x is y)
```

```
True
True
```

In [3]:

```
a="durga"
b="durga"
print(id(a))
print(id(b))
print(a is b)
```

```
2730506434688
2730506434688
True
```

In [5]:

```
list1=["one","two","three"]
list2=["one","two","three"]
print(id(list1))
print(id(list2))
print(list1 is list2)
print(list1 is not list2)      # reference comaprison (is & is not)
print(list1 == list2)          # content comparison   (==)
```

```
2730505561800
2730505562120
False
True
True
```

Note:

- We can use is operator for address comparison where as == operator for content comparison.

2. Membership Operators

- We can use Membership operators to check whether the given object present in the given collection.(It may be String,List,Set,Tuple or Dict)
- There are two types of membership operators used in Python:

i) in

ii) not in

in returns True if the given object present in the specified Collection.

not in retruns True if the given object not present in the specified Collection.

Eg :

In [9]:

```
x="hello learning Python is very easy!!!"
print('h' in x)
print('d' in x)
print('d' not in x)
print('python' in x)    # case sensitivity
print('Python' in x)
```

```
True
False
True
False
True
```

In [7]:

```
list1=["sunny","bunny","chinny","pinny"]

print("sunny" in list1)
print("tunny" in list1)
print("tunny" not in list1)
```

```
True
False
True
```

10. Operator Precedence

- If multiple operators present then which operator will be evaluated first is decided by operator precedence.

Eg :

In [11]:

```
print(3+10*2)
print((3+10)*2)
```

```
23
26
```

The following list describes operator precedence in Python:

- () ==> Parenthesis
- ** ==> exponential operator
- ~,- ==> Bitwise complement operator,unary minus operator
- *,/,%,// ==> multiplication,division,modulo,floor division
- +,- ==> addition,subtraction
- <<,>> ==> Left and Right Shift
- & ==> bitwise And

- ^ ==> Bitwise X-OR
- | ==> Bitwise OR
- <,<=,>,>=,==, != ==> Relational or Comparison operators
- =,+=,-=,*=... ==> Assignment operators
- is , is not ==> Identity Operators
- in , not in ==> Membership operators
- not ==> Logical not
- and ==> Logical and
- or ==> Logical or

Eg 1:

In [1]:

```
a=30
b=20
c=10
d=5
print((a+b)*c/d)      # division operoator in Python always going to provide float value as
print((a+b)*(c/d))
print(a+(b*c)/d)
```

```
100.0
100.0
70.0
```

Eg 2:

In [3]:

```
print(3/2*4+3+(10/5)**3-2)
print(3/2*4+3+2.0**3-2)
print(3/2*4+3+8.0-2)
print(1.5*4+3+8.0-2)
print(6.0+3+8.0-2)
```

```
15.0
15.0
15.0
15.0
15.0
```

11. Mathematical functions using 'math' module

Basic Idea about Python Module

- A Module is collection of functions, variables and classes etc. In simple words, module is nothing but a Python file.

- Python contains many in-built libraries.
- Being a developer, we can use these modules, but we are not responsible for developing the functionalities about these modules.
- For example, **math** is a module that contains several functions to perform mathematical operations.
- If we want to use any module in Python, first we have to import that module.

```
import math
```

- Once we import a module then we can call any function of that module.
- The biggest advantage of using modules is **Code Reusability**.

Eg :

In [4]:

```
a = 888
b = 999

def add(x,y):
    print('Performing Add operation :')
    print('Sum is : ',x + y)

def mul(x,y):
    print('Performing Mul operation :')
    print('Product is : ',x * y)          # Assume that We are saving this cell as pmat
```

In [5]:

```
# use pmath.py module using import statement. In jupyter notebook file we need not to  impo

add(10,20)
mul(10,20)
```

```
Performing Add operation :
Sum is :  30
Performing Mul operation :
Product is :  200
```

How can youfoind which functions are available in 'math' module?

In [7]:

```
import math
print(dir(math))   # provides the list of functions names, variables, constants available i
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'ac
osh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'l
og2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sq
rt', 'tan', 'tanh', 'tau', 'trunc']
```

Eg :

In [1]:

```
import math
print(math.sqrt(16))
print(math.pi)
print(math.e)
print(math.floor(3.98))
print(math.ceil(3.98))
print(math.pow(3,2))
```

```
4.0
3.141592653589793
2.718281828459045
3
4
9.0
```

Module Aliasing:

Whenever we are importing **math** module, so any variable or function we can use within that module by specifying with the name of that module(For example, math.sqrt and math.py etc).

If the module name is bigger then there is a problem of every time make using of that bigger module name when you are calling the function within that module. As result of this length of the code increases. To avoid this, we can create alias name by using **as** keyword.

import math as m

Once we create alias name, by using that we can access functions and variables of that module.

In [5]:

```
import math as m
print(m.sqrt(16))
print(m.pi)
print(math.floor(3.9))
```

```
4.0
3.141592653589793
3
```

We can import a particular member of a module explicitly as follows

Eg :

from math import sqrt

from math import sqrt,pi

If we import a member explicitly then it is not required to use module name while accessing.

In [7]:

```
from math import sqrt,pi

print(sqrt(16))
print(pi)

print(math.pi)
```

```
4.0
3.141592653589793
3.141592653589793
```

Member Aliasing

We can create alias name for member of module also.

In [8]:

```
from math import sqrt as s

print(s(16))
```

```
4.0
```

In [9]:

```
from math import sqrt as s, pi as p

print(s(16))
print(p)
```

```
4.0
3.141592653589793
```

Important functions and variables of math module

Note :

In [10]:

```
def f1():
    print('f1 old function')

def f1():
    print('f1 new function')


f1()        # Most recent function will execute, if the function with same name diffrent fu
```

```
f1 new function
```

Important functions present in math module:

ceil(x)

floor(x)

pow(x,y)

factorial(x)

trunc(x)

gcd(x,y)

sin(x)

cos(x)

tan(x) ....

Important variables of math module:

pi ==> 3.14

e ==> 2.71

inf ==> infinity

nan ==> not a number

Eg : Q. Write a Python program to find area of circle.

In [14]:

```
# Find the area of circle for the given radius

from math import pi

radius = int(input("Enter radius Value : "))
area = pi * (radius**2)

print("Area of the circle for the given radius is : ",area)
```

```
Enter radius Value : 5
Area of the circle for the given radius is :  78.53981633974483
```

Another Way

In [18]:

```
import math
r=5
print("Area of Circle is :",pi*r**2)
```

```
Area of Circle is : 78.53981633974483
```

Another Way

```
In [19]:

from math import *
r=5
print("Area of Circle is :",pi*pow(r,2))
```

```
Area of Circle is : 78.53981633974483
```

```
In [ ]:
```

UNIT - 2

Input and Output Statements in Python

Date: 20-04-2020 Day 1

Introduction

In this lecture we will learn about how to read the user provided input and print ouput to the end user.

The following topics we are going to discuss as part of this lecture:

1. raw_input() vs input()
2. Python-3 input() function
3. How to read multiple values from the keyboard in a single line
4. Command line arguments
5. Output statement : print() function
6. sep attribute
7. end attribute
8. Printing formatted string
9. Replacement Operator : { }

1. raw_input( ) vs input( )

Reading dynamic input from the keyboard:

In Python 2 the following 2 functions are available to read dynamic input from the keyboard.

1. raw_input()
2. input()

- **1. raw_input():**

  This function always reads the data from the keyboard in the form of String Format. We have to convert that string type to our required type by using the corresponding type casting methods.

Eg :

```
In [ ]:
```

```
x = raw_input("Enter First Number : ")
print(type(x))                          # It will always print str type only for any input type
```

2. input():

- input() function can be used to read data directly in our required format.We are not required to perform type casting.

In [ ]:

```
x=input("Enter Value)
type(x)

10 ===> int

"durga"===>str

10.5===>float

True==>bool
```

Note:

- But in Python3 we have only input() method and raw_input() method is not available.
- Python3 input() function behaviour exactly same as raw_input() method of Python2. i.e every input value is treated as str type only.
- raw_input() function of Python 2 is renamed as input() function in Python3.

Eg :

In [5]:

```
type(input("Enter value:"))
```

```
Enter value:10
```

Out[5]:

```
str
```

In [6]:

```
type(input("Enter value:"))
```

```
Enter value:22.8
```

Out[6]:

```
str
```

In [7]:

```
type(input("Enter value:"))
```

```
Enter value:True
```

Out[7]:

```
str
```

Note :

Why input() in Python 3 gave the priority for string type as return type?

Reason: The most commonly used type in any programming language is **str type** , that's why they gave the priority for str type as default return type of input() function.

Demo Program 1: Read input data from the Keyboard

In [1]:

```
x=input("Enter First Number:")
y=input("Enter Second Number:")

i = int(x)
j = int(y)

print("The Sum:",i+j)
```

```
Enter First Number:100
Enter Second Number:200
The Sum: 300
```

above code in simplified form:

In [2]:

```
x=int(input("Enter First Number:"))
y=int(input("Enter Second Number:"))
print("The Sum:",x+y)
```

```
Enter First Number:100
Enter Second Number:200
The Sum: 300
```

We can write the above code in single line also.

In [3]:

```
print("The Sum:",int(input("Enter First Number:"))+int(input("Enter Second Number:")))
```

```
Enter First Number:100
Enter Second Number:200
The Sum: 300
```

Demo Program 2: Write a program to read Employee data from the keyboard and print that data.

In [8]:

```
eno=int(input("Enter Employee No:"))
ename=input("Enter Employee Name:")
esal=float(input("Enter Employee Salary:"))
eaddr=input("Enter Employee Address:")
married=bool(input("Employee Married ?[True|False]:"))

print("Please Confirm your provided Information")
print("Employee No :",eno)
print("Employee Name :",ename)
print("Employee Salary :",esal)
print("Employee Address :",eaddr)
print("Employee Married ? :",married)
```

```
Enter Employee No:874578
Enter Employee Name:Karthi
Enter Employee Salary:23500
Enter Employee Address:Nandyal
Employee Married ?[True|False]:T
Please Confirm your provided Information
Employee No : 874578
Employee Name : Karthi
Employee Salary : 23500.0
Employee Address : Nandyal
Employee Married ? : True
```

In [9]:

```
eno=int(input("Enter Employee No:"))
ename=input("Enter Employee Name:")
esal=float(input("Enter Employee Salary:"))
eaddr=input("Enter Employee Address:")
married=bool(input("Employee Married ?[True|False]:"))

print("Please Confirm your provided Information")
print("Employee No :",eno)
print("Employee Name :",ename)
print("Employee Salary :",esal)
print("Employee Address :",eaddr)
print("Employee Married ? :",married)
```

```
Enter Employee No:245784
Enter Employee Name:Karthi
Enter Employee Salary:34566
Enter Employee Address:Nandyal
Employee Married ?[True|False]:False
Please Confirm your provided Information
Employee No : 245784
Employee Name : Karthi
Employee Salary : 34566.0
Employee Address : Nandyal
Employee Married ? : True
```

In [10]:

```
eno=int(input("Enter Employee No:"))
ename=input("Enter Employee Name:")
esal=float(input("Enter Employee Salary:"))
eaddr=input("Enter Employee Address:")
married=bool(input("Employee Married ?[True|False]:"))

print("Please Confirm your provided Information")
print("Employee No :",eno)
print("Employee Name :",ename)
print("Employee Salary :",esal)
print("Employee Address :",eaddr)
print("Employee Married ? :",married)
```

```
Enter Employee No:245784
Enter Employee Name:Karthi
Enter Employee Salary:34566
Enter Employee Address:Nandyal
Employee Married ?[True|False]:
Please Confirm your provided Information
Employee No : 245784
Employee Name : Karthi
Employee Salary : 34566.0
Employee Address : Nandyal
Employee Married ? : False
```

When you are not providing any value to the married (Just press **Enter**), then only it considers empty string and gives the **False** value. In the above example, to read the boolean data, we need to follow the above process.

But it is not our logic requirement. If you want to convert string to Boolean type, instead of using **bool()** function we need to use **eval()** function.

In [11]:

```
eno=int(input("Enter Employee No:"))
ename=input("Enter Employee Name:")
esal=float(input("Enter Employee Salary:"))
eaddr=input("Enter Employee Address:")
married=eval(input("Employee Married ?[True|False]:"))

print("Please Confirm your provided Information")
print("Employee No :",eno)
print("Employee Name :",ename)
print("Employee Salary :",esal)
print("Employee Address :",eaddr)
print("Employee Married ? :",married)
```

```
Enter Employee No:245784
Enter Employee Name:Karthi
Enter Employee Salary:34566
Enter Employee Address:Nandyal
Employee Married ?[True|False]:False
Please Confirm your provided Information
Employee No : 245784
Employee Name : Karthi
Employee Salary : 34566.0
Employee Address : Nandyal
Employee Married ? : False
```

Date: 21-04-2020 Day-2

3. Reading multiple values from the keyboard in a single line

In [12]:

```
a,b= [int(x) for x in input("Enter 2 numbers :").split()]

print("The Sum is :", a + b)
```

```
Enter 2 numbers :10 20
The Sum is : 30
```

Explanation :

- Here, we are using only one input function (i.e., **input("Enter 2 numbers :")**). So what ever you provide it is treated as only one string.
- Suppose, you are provinding input as **10 20**, this is treated as single string.
- If you want to split that string into multiple values, then we required to use **split() function** .
- If we want to split the given string (i.e., **10 20**) with respect to space, then the code will be as follows:

  ```
  input('Enter 2 numbers :").split()
  ```

- Here, we are not passing any argument to split() function, then it takes default parameter (i.e., space) as seperator.
- Now this single string(i.e.,10 20) is splitted into list of two string values.

- Now the statement : **input('Enter 2 numbers :").split()** returns ['10','20']
- Now, in the list every number is available in string form.
- So, what we will do here is, each value present in this list is typecast to 'int' value.

```
     [int(x)  for  x in input('Enter 2 numbers :").split()]  ===> it retrns [10,2
0]
```

- This concept is known as **list comprehension**.
- a,b = [int(x) for x in input('Enter 2 numbers :").split()] ===> it assigns 10 to a and 20 to b. This concept is called as **list unpacking**.
- a,b = [int(x) for x in input('Enter 2 numbers :").split()]
- print('Sum is : ',a + b) ====> Gives the sum of two values as the result.

Note:

- **split()** function can take space as seperator by default .But we can pass anything as seperator.

In [13]:

```python
# a,b= [int(x) for x in input("Enter 2 numbers :").split()]

s = input("Enter 2 numbers :")
print(s,type(s))
```

```
Enter 2 numbers :10 20
10 20 <class 'str'>
```

In [14]:

```python
s = input("Enter 2 numbers :")
print(s,type(s))    # s holds single value  '10 20'
l=s.split()         # After split, single string will be divided into list of two values of
print(l,type(l))
```

```
Enter 2 numbers :10 20
['10', '20'] <class 'list'>
```

In [18]:

```
s = input("Enter 2 numbers :")
print(s,type(s))    # s holds single value  '10 20'
l = s.split()         # After split, single string will be divided into list of two values
print(l)
l1 = [int(x) for x in l]   # This new list contains two int values after typecastig of each
print(l1)
a,b = l1   # in this list wahtever the values are there, assigns first value to 'a' and sec
           #This is called 'list unpacking'.
print(a)
print(b)
print('Sum is :', a+b)
```

```
Enter 2 numbers :10 20
10 20 <class 'str'>
['10', '20']
[10, 20]
10
20
Sum is : 30
```

By substituting the elements of the above code, we will get the below code (same as above code):

In [19]:

```
a,b = [int(x) for x in input("Enter 2 numbers :").split()]
print('Sum is :', a+b)
```

```
Enter 2 numbers :10 20
Sum is : 30
```

In [20]:

```
a,b = [int(x) for x in input("Enter 2 numbers :").split(',')]
print('Sum is :', a+b)
```

```
Enter 2 numbers :10,20
Sum is : 30
```

In [22]:

```
a,b = [int(x) for x in input("Enter 2 numbers :").split(',')]
print('Sum is :', a+b)
```

```
Enter 2 numbers :10 20

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-d2adfb9ab442> in <module>
----> 1 a,b = [int(x) for x in input("Enter 2 numbers :").split(',')]
      2 print('Sum is :', a+b)

<ipython-input-22-d2adfb9ab442> in <listcomp>(.0)
----> 1 a,b = [int(x) for x in input("Enter 2 numbers :").split(',')]
      2 print('Sum is :', a+b)

ValueError: invalid literal for int() with base 10: '10 20'
```

Date: 22-04-2020 Day 3

Demo Program 3: Q. Write a program to read 3 float numbers from the keyboard with , seperator and print their sum.

In [3]:

```
a,b,c= [float(x) for x in input("Enter 3 float numbers with , seperation :").split(',')]
print("The Sum is :", a+b+c)
```

```
Enter 3 float numbers with , seperation :10.5,20.6,30.7
The Sum is : 61.8
```

eval() Function:

- **eval()** Function is a single function which is the replacement of all the typecasting functions in Python.

In [7]:

```
x = (input('Enter Something : '))
print(type(x))
```

```
Enter Something : karthi
<class 'str'>
```

In [8]:

```
x = (input('Enter Something : '))
print(type(x))
```

```
Enter Something : 10
<class 'str'>
```

In [9]:

```
x = (input('Enter Something :'))
print(type(x))
```

```
Enter Something :10.5
<class 'str'>
```

In [18]:

```
x = eval((input('Enter Something : ')))
print(type(x))
```

```
Enter Something : 'karthi'
<class 'str'>
```

In [11]:

```
x = eval((input('Enter Something : ')))
print(type(x))
```

```
Enter Something : 10
<class 'int'>
```

In [13]:

```
x = eval((input('Enter Something : ')))
print(type(x))
```

```
Enter Something : 10.5
<class 'float'>
```

In [16]:

```
x = eval((input('Enter Something : ')))
print(type(x))
```

```
Enter Something : True
<class 'bool'>
```

In [19]:

```
x = eval((input('Enter Something : ')))
print(type(x))
```

```
Enter Something : [10,20,30]
<class 'list'>
```

In [20]:

```
x = eval((input('Enter Something : ')))
print(type(x))
```

```
Enter Something : (10,20,30)
<class 'tuple'>
```

In [21]:

```
x = eval((input('Enter Something : ')))
print(type(x))
```

```
Enter Something : (10)
<class 'int'>
```

In [22]:

```
x = eval((input('Enter Something : ')))
print(type(x))
```

```
Enter Something : (10,)
<class 'tuple'>
```

-If you provide an expression as a string type, eval() function evaluates that expression and provide the result.

In [24]:

```
x = eval('10+20+30')
print(x,type(x))
```

```
60 <class 'int'>
```

In [25]:

```
x = eval(10+20+30)
print(x,type(x))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-9d177369d9d9> in <module>
----> 1 x = eval(10+20+30)
      2 print(x,type(x))

TypeError: eval() arg 1 must be a string, bytes or code object
```

In [26]:

```
x = eval('10+20/3**4//5*40')
print(x,type(x))
```

```
10.0 <class 'float'>
```

4. Command Line Arguments

- Command line arguments is another way to read the user provided input.
- The Argument which are passing at the time of execution are called Command Line Arguments.
- argv is not Array it is a List. It is available **sys** Module.
- Eg: D:\Python_classes py test.py 10 20 30 (Command Line Arguments).
- Within the Python Program this Command Line Arguments are available in argv. Which is present in **sys** Module.

```
    test.py 10 20 30
```

Note:

- argv[0] represents Name of Program. But not first Command Line Argument.
- argv[1] represent First Command Line Argument.

In [28]:

```
from sys import argv
print(type(argv))
```

```
<class 'list'>
```

Eg 1: (Executed in Atom Editor)

```
from sys import argv

print(“The Number of Command Line Arguments:”, len(argv))

print(“The List of Command Line Arguments:”, argv)

print(“Command Line Arguments one by one:”)

for x in argv:

    print(x)
```

Output :

```
D:\Python_classes>py test.py 10 20 30

The Number of Command Line Arguments: 4

The List of Command Line Arguments: [‘test.py’, ‘10’,’20’,’30’]

Command Line Arguments one by one:

test.py

10

20

30
```

Eg 2: (Executed in Atom Editor)

```
from sys import argv

sum=0

args=argv[1:]

for x in args :

    n=int(x)

    sum=sum+n

print("The Sum:",sum)
```

Output:

```
D:\Python_classes>py test.py 10 20 30 40

The Sum: 100
```

Note 1:

- Usually space is seperator between command line arguments. If our command line argument itself contains space then we should enclose within double quotes(but not single quotes)

(Executed in Edit Plus)

```
from sys import argv

print(argv[1])
```

Output:

```
D:\Python_classes>py test.py Karthi Keya

Karthi

D:\Python_classes>py test.py 'Karthi Keya'

'Karthi'

D:\Python_classes>py test.py "Karthi Keya"

Karthi Keya
```

Note 2:

- Within the Python program command line arguments are available in the String form. Based on our requirement,we can convert into corresponding type by using typecasting methods.

(Executed in Edit Plus)

```
from sys import argv

print(argv[1]+argv[2])

print(int(argv[1])+int(argv[2]))
```

Output :

```
D:\Python_classes>py test.py 10 20

1020

30
```

Note 3:

- If we are trying to access command line arguments with out of range index then we will get Error.

(Executed in Edit Plus)

```
from sys import argv

print(argv[100])

D:\Python_classes>py test.py 10 20

IndexError: list index out of range
```

5. Output Statements : 'print()' function

- We can use **print()** function to display output to the console for end user sake.
- Multiple forms are there related to **print()** function.

Form-1:print() without any argument

- Just it prints new line character (i.e.,\n)

In [1]:

```
print('karthi')
print()         # prints new line character
print('sahasra')
```

```
karthi

sahasra
```

see the difference in below code:

In [2]:

```
print('karthi')
#print()         # prints new line character
print('sahasra')
```

```
karthi
sahasra
```

Form-2: print() function to print of string argument

In [5]:

```
 print("Hello World")
```

```
Hello World
```

- We can use escape characters also.

In [6]:

```
print("Hello \n World")
print("Hello\tWorld")
```

```
Hello
 World
Hello	World
```

- We can use repetetion operator (*) in the string.

In [7]:

```
print(10*"Hello")
print("Hello"*10)
```

```
HelloHelloHelloHelloHelloHelloHelloHelloHelloHello
HelloHelloHelloHelloHelloHelloHelloHelloHelloHello
```

- We can use + operator also.

In [8]:

```
print("Hello"+"World")
```

```
HelloWorld
```

Note:

- If both arguments are string type then + operator acts as concatenation operator.
- If one argument is string type and second is any other type like int then we will get Error
- If both arguments are number type then + operator acts as arithmetic addition operator.

```
**Form-3: print() with variable number of arguments:**
```

In [10]:

```
a,b,c=10,20,30
print("The Values are :",a,b,c)  # here, we are passing 4 arguments to the print function.
```

```
The Values are : 10 20 30
```

6. 'sep' attribute:

Form-4: print() with 'sep' attribute:

- By default output values are seperated by space. If we want we can specify seperator by using **"sep"** attribute.
- 'sep' means seperator.

In [14]:

```
a,b,c=10,20,30
print(a,b,c)              # 10 20 30
print(a,b,c,sep=',')      # 10,20,30
print(a,b,c,sep=':')     # 10:20:30
print(a,b,c,sep='-')    # 10-20-30
```

```
10 20 30
10,20,30
10:20:30
10-20-30
```

7. 'end' attribute:

Form-5: print() with 'end' attribute:

In [15]:

```
print("Hello")
print("Karthi")
print("Sahasra")
```

```
Hello
Karthi
Sahasra
```

- If we want output in the same line with space, we need to use **end** attribute.
- **default value of 'end' attribute is newline character.** (That means, if there is no end attribute, automatically newline character will be printed)

In [21]:

```
print("Hello",end=' ')
print("Karthi",end=' ')   # if end is space character
print("Sahasra")
```

```
Hello Karthi Sahasra
```

In [22]:

```
print("Hello",end='')
print("Karthi",end='')  # if end is nothing
print("Sahasra")
```

```
HelloKarthiSahasra
```

In [23]:

```
print('hello',end = '::')
print('karthi',end = '****')
print('sahasra')
```

```
hello::karthi****sahasra
```

Eg: Program to demonstrate both 'sep' and 'end' attributes.

```
In [26]:
```

```
print(10,20,30,sep = ':', end = '***')
print(40,50,60,sep = ':')       # default value of 'end' attribute is '\n'
print(70,80,sep = '**',end = '$$')
print(90,100)
```

```
10:20:30***40:50:60
70**80$$90 100
```

```
**Eg :** Consider the following case,
```

```
In [37]:
```

```
print('karthi' + 'sahasra')   # Concatanation
print('karthi','sahasra')    # ',' means space is the seperator
print(10,20,30)
```

```
karthisahasra
karthi sahasra
10 20 30
```

Form-6: print(object) statement:

- We can pass any object (like list,tuple,set etc)as argument to the print() statement.

Eg:

```
In [27]:
```

```
l=[10,20,30,40]
t=(10,20,30,40)
print(l)
print(t)
```

```
[10, 20, 30, 40]
(10, 20, 30, 40)
```

Form-7: print(String,variable list):

- We can use print() statement with String and any number of arguments.

```
In [38]:
```

```
s="Karthi"
a=6
s1="java"
s2="Python"
print("Hello",s,"Your Age is",a)
print("You are learning",s1,"and",s2)
```

```
Hello Karthi Your Age is 6
You are learning java and Python
```

8.Printing formatted string

Form-8: print(formatted string):

%i ====>int

%d ====>int

%f =====>float

%s ======>String type

Syntax:

- **print("formatted string" %(variable list))**

In [43]:

```
a=10
b=20
c=30
print("a value is %i" %a)
print("b value is %d and c value is %d" %(b,c))
```

```
a value is 10
b value is 20 and c value is 30
```

In [57]:

```
s="Karthi"
list=[10,20,30,40]
print("Hello %s ...The List of Items are %s" %(s,list))
```

```
Hello Karthi ...The List of Items are [10, 20, 30, 40]
```

In [60]:

```
price = 70.56789
print('Price value = {}'.format(price))
print('Price value = %f'%price)
print('Price value = %.2f'%price)      # only two digits after decimal  point, we can't do
                                    #This type of customization is possible with formatted s
```

```
Price value = 70.56789
Price value = 70.567890
Price value = 70.57
```

9. Replacement Operator ({ }):

Form-9: print() with replacement operator { }

Eg: 1

In [53]:

```
name = "Karthi"
salary = 100000
sister = "Sahasra"

print("Hello {} your salary is {} and Your Sister {} is waiting".format(name,salary,sister)
print("Hello {0} your salary is {1} and Your Sister {2} is waiting".format(name,salary,sist
print("Hello {1} your salary is {2} and Your Sister {0} is waiting".format(name,salary,sist
print("Hello {2} your salary is {0} and Your Sister {1} is waiting".format(salary,sister,na
print("Hello {x} your salary is {y} and Your Sister {z} is waiting".format(x=name,y=salary,
```

```
Hello Karthi your salary is 100000 and Your Sister Sahasra is waiting
Hello Karthi your salary is 100000 and Your Sister Sahasra is waiting
Hello 100000 your salary is Sahasra and Your Sister Karthi is waiting
Hello Karthi your salary is 100000 and Your Sister Sahasra is waiting
Hello Karthi your salary is 100000 and Your Sister Sahasra is waiting
```

Eg: 2

In [56]:

```
a,b,c,d = 10,20,30,40   # print a=10,b=20,c=30,d=40
print('a = {},b = {},c = {},d = {}'.format(a,b,c,d))
```

```
a = 10,b = 20,c = 30,d = 40
```

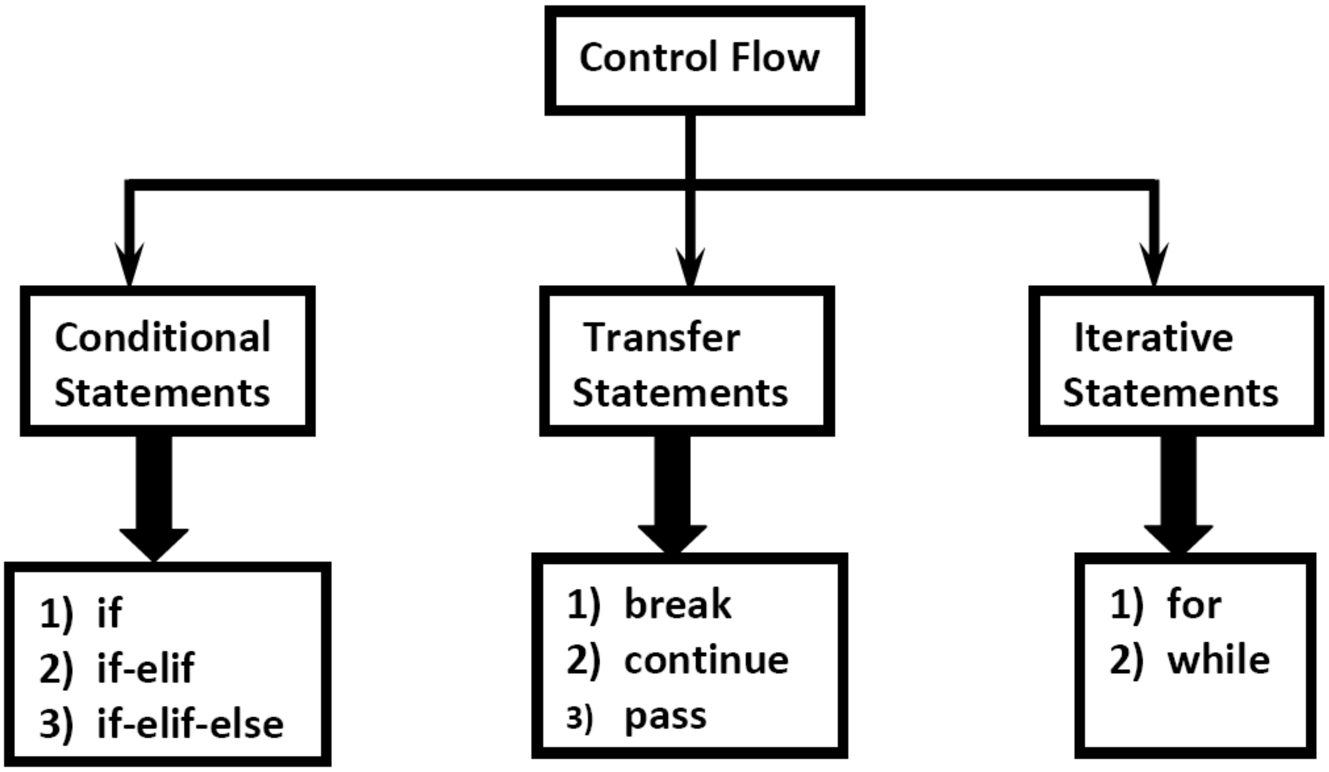
Good Luck

Flow Control Statements:

Date: 22-04-2020 Day-1

Flow control describes the order in which statements will be executed at runtime.

- Flow control statements are divided into three categoriesin Python.



1. Conditional Statements (or) Selection Statements

- Based on some condition result, some group of statements will be executed and some group of statements will not be executed.

Note:

- There is no **switch** statement in Python. (Which is available in C and Java)
- There is no do-while loop in Python.(Which is available in C and Java)
- goto statement is also not available in Python. (Which is available in C)

i) if Statement :

Before going to discuss about if statement syntax and examples, we need to know about an important concept known as **indentation**.

In C/Java language, How to define if statement?

if(condition)

{

body

}

statements

Here, by using curly braces, we can define a block of statements.

But in Python, curly braces style is not there. Then how can we define a particular statement is under if statement (i.e, How we can define if block?).

In Python:

if condition: (**Note:** In Python any where we are using colon(:) means we are defining block)

```
statement 1

statement 2     (These statements are said to be same indentation)

statement 3
```

statement 4 (This staement is not under if statement)

If you are not followed indentation, you will get indentation error.

- In Python enabled editors indention is automatically maintained.

In [ ]:

```
if condition:
    statement 1
    statement 2
    statement 3
statement
```

In this way we need to follow indentation in Python.

In [2]:

```
if 10<20:
    print('10 is less than 20')
print('End of Program')
```

```
10 is less than 20
End of Program
```

In [3]:

```
if 10<20:
print('10 is less than 20')
print('End of Program')
```

```
  File "<ipython-input-3-f2d3b9a6180e>", line 2
    print('10 is less than 20')
        ^
IndentationError: expected an indented block
```

Eg:

In [7]:

```
name=input("Enter Name:")
if name=="Karthi":
    print("Hello Karthi Good Morning")
print("How are you!!!")
```

```
Enter Name:Karthi
Hello Karthi Good Morning
How are you!!!
```

In [8]:

```
name=input("Enter Name:")
if name=="Karthi":
    print("Hello Karthi Good Morning")
print("How are you!!!")
```

```
Enter Name:ravi
How are you!!!
```

ii) if - else Statement:

Syntax:

if condition:

```
Action 1
```

else:

```
Action 2
```

- if condition is true then Action-1 will be executed otherwise Action-2 will be executed.

Eg:

In [10]:

```
name = input('Enter Name : ')
if name == 'Karthi':
    print('Hello Karthi! Good Morning')
else:
    print('Hello Guest! Good MOrning')
print('How are you?')
```

```
Enter Name : Karthi
Hello Karthi! Good Morning
How are you?
```

In [11]:

```
name = input('Enter Name : ')
if name == 'Karthi':
    print('Hello Karthi! Good Morning')
else:
    print('Hello Guest! Good MOrning')
print('How are you?')
```

```
Enter Name : ram
Hello Guest! Good MOrning
How are you?
```

iii) if-elif-else Statement:

Syntax:

if condition1:

```
Action-1
```

elif condition2:

```
Action-2
```

elif condition3:

```
Action-3
```

elif condition4:

```
Action-4
```

...

else:

```
Default Action
```

Based condition the corresponding action will be executed.

Eg :

In [12]:

```
brand=input("Enter Your Favourite Brand:")
if brand=="RC":
    print("It is childrens brand")
elif brand=="KF":
    print("It is not that much kick")
elif brand=="FO":
    print("Buy one get Free One")
else :
    print("Other Brands are not recommended")
```

```
Enter Your Favourite Brand:RC
It is childrens brand
```

In [13]:

```
brand=input("Enter Your Favourite Brand:")
if brand=="RC":
    print("It is childrens brand")
elif brand=="KF":
    print("It is not that much kick")
elif brand=="FO":
    print("Buy one get Free One")
else :
    print("Other Brands are not recommended")
```

```
Enter Your Favourite Brand:FO
Buy one get Free One
```

In [14]:

```
brand=input("Enter Your Favourite Brand:")
if brand=="RC":
    print("It is childrens brand")
elif brand=="KF":
    print("It is not that much kick")
elif brand=="FO":
    print("Buy one get Free One")
else :
    print("Other Brands are not recommended")
```

```
Enter Your Favourite Brand:abc
Other Brands are not recommended
```

Points to Ponder:

1. else part is always optional

- Hence the following are various possible syntaxes.

```
    1. if

    2. if - else

    3. if-elif-else

    4. if-elif
```

2. There is no switch statement in Python

Example Programs

Q 1. Write a program to find biggest of given 2 numbers.

In [17]:

```
n1=int(input("Enter First Number:"))
n2=int(input("Enter Second Number:"))
if n1>n2:
    print("Biggest Number is:",n1)
else :
    print("Biggest Number is:",n2)
```

```
Enter First Number:10
Enter Second Number:20
Biggest Number is: 20
```

Q 2. Write a program to find biggest of given 3 numbers.

In [18]:

```
n1=int(input("Enter First Number:"))
n2=int(input("Enter Second Number:"))
n3=int(input("Enter Third Number:"))
if n1>n2 and n1>n3:
    print("Biggest Number is:",n1)
elif n2>n3:
    print("Biggest Number is:",n2)
else :
    print("Biggest Number is:",n3)
```

```
Enter First Number:10
Enter Second Number:20
Enter Third Number:35
Biggest Number is: 35
```

Q 3. Write a program to find smallest of given 2 numbers?

In [19]:

```
n1=int(input("Enter First Number:"))
n2=int(input("Enter Second Number:"))
if n1>n2:
    print("Smallest Number is:",n2)
else :
    print("Smallest Number is:",n1)
```

```
Enter First Number:10
Enter Second Number:20
Smallest Number is: 10
```

Q 4. Write a program to find smallest of given 3 numbers?

In [20]:

```
n1=int(input("Enter First Number:"))
n2=int(input("Enter Second Number:"))
n3=int(input("Enter Third Number:"))
if n1<n2 and n1<n3:
    print("Smallest Number is:",n1)
elif n2<n3:
    print("Smallest Number is:",n2)
else :
    print("Smallest Number is:",n3)
```

```
Enter First Number:10
Enter Second Number:20
Enter Third Number:30
Smallest Number is: 10
```

Q 5. Write a program to check whether the given number is even or odd?

In [21]:

```
n1=int(input("Enter First Number:"))
rem = n1 % 2
if rem == 0:
    print('Entered Number is an Even Number')
else:
    print('Entered Number is an Odd Number')
```

```
Enter First Number:56
Entered Number is an Even Number
```

In [22]:

```
n1=int(input("Enter First Number:"))
rem = n1 % 2
if rem == 0:
    print('Entered Number is an Even Number')
else:
    print('Entered Number is an Odd Number')
```

```
Enter First Number:55
Entered Number is an Odd Number
```

Q 6. Write a program to check whether the given number is in between 1 and 100?

In [23]:

```
n=int(input("Enter Number:"))
if n>=1 and n<=100 :
    print("The number",n,"is in between 1 to 100")
else:
    print("The number",n,"is not in between 1 to 100")
```

```
Enter Number:78
The number 78 is in between 1 to 100
```

In [24]:

```
n=int(input("Enter Number:"))
if n>=1 and n<=100 :
    print("The number",n,"is in between 1 to 100")
else:
    print("The number",n,"is not in between 1 to 100")
```

```
Enter Number:101
The number 101 is not in between 1 to 100
```

Q 7. Write a program to take a single digit number from the key board and print it's value in English word?

In [25]:

```
n=int(input("Enter a digit from o to 9:"))
if n==0 :
    print("ZERO")
elif n==1:
    print("ONE")
elif n==2:
    print("TWO")
elif n==3:
    print("THREE")
elif n==4:
    print("FOUR")
elif n==5:
    print("FIVE")
elif n==6:
    print("SIX")
elif n==7:
    print("SEVEN")
elif n==8:
    print("EIGHT")
elif n==9:
    print("NINE")
else:
    print("PLEASE ENTER A DIGIT FROM 0 TO 9")
```

```
Enter a digit from o to 9:8
EIGHT
```

In [26]:

```
n=int(input("Enter a digit from o to 9:"))
if n==0 :
    print("ZERO")
elif n==1:
    print("ONE")
elif n==2:
    print("TWO")
elif n==3:
    print("THREE")
elif n==4:
    print("FOUR")
elif n==5:
    print("FIVE")
elif n==6:
    print("SIX")
elif n==7:
    print("SEVEN")
elif n==8:
    print("EIGHT")
elif n==9:
    print("NINE")
else:
    print("PLEASE ENTER A DIGIT FROM 0 TO 9")
```

```
Enter a digit from o to 9:10
PLEASE ENTER A DIGIT FROM 0 TO 9
```

Another Way of writing program for the same requirement

In [16]:

```
list1 = ['ZERO','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIGHT','NINE']
n =int(input('Enter a digit from 0 to 9 :'))
print(list1[n])
```

```
Enter a digit from 0 to 9 :7
SEVEN
```

In [17]:

```
list1 = ['ZERO','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIGHT','NINE']
n =int(input('Enter a digit from 0 to 9 :'))
print(list1[n])
```

```
Enter a digit from 0 to 9 :15

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-17-7dee7e007a8f> in <module>
      1 list1 = ['ZERO','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIG
HT','NINE']
      2 n =int(input('Enter a digit from 0 to 9 :'))
----> 3 print(list1[n])

IndexError: list index out of range
```

How can you extend the above program from 0 to 99?

In [19]:

```
words_upto_19 = ['','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIGHT','NINE','TEN','E
                 'TWELVE','THIRTEEN','FOURTEEN','FIFTEEN','SIXTEEN','SEVENTEEN','EIGHTEEN',

words_for_tens = ['','','TWENTY','THIRTY','FORTY','FIFTY','SIXTY','SEVENTY','EIGHTY','NINET

n = int(input('Enter a number from 0 to 99 : '))

output = ''

if n == 0:
    output = 'ZERO'
elif n <= 19:
    output = words_upto_19[n]
elif n<=99:
    output = words_for_tens[n//10]+' '+words_upto_19[n%10]
else:
    output = 'Pleae Enter a value fron 0 to 99 only'
print(output)
```

```
Enter a number from 0 to 99 : 0
ZERO
```

In [20]:

```
words_upto_19 = ['','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIGHT','NINE','TEN','E
                 'TWELVE','THIRTEEN','FOURTEEN','FIFTEEN','SIXTEEN','SEVENTEEN','EIGHTEEN',

words_for_tens = ['','','TWENTY','THIRTY','FORTY','FIFTY','SIXTY','SEVENTY','EIGHTY','NINET

n = int(input('Enter a number from 0 to 99 : '))

output = ''

if n == 0:
    output = 'ZERO'
elif n <= 19:
    output = words_upto_19[n]
elif n<=99:
    output = words_for_tens[n//10]+' '+words_upto_19[n%10]
else:
    output = 'Pleae Enter a value fron 0 to 99 only'
print(output)
```

```
Enter a number from 0 to 99 : 9
NINE
```

In [21]:

```
words_upto_19 = ['','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIGHT','NINE','TEN','E
                 'TWELVE','THIRTEEN','FOURTEEN','FIFTEEN','SIXTEEN','SEVENTEEN','EIGHTEEN',

words_for_tens = ['','','TWENTY','THIRTY','FORTY','FIFTY','SIXTY','SEVENTY','EIGHTY','NINET

n = int(input('Enter a number from 0 to 99 : '))

output = ''

if n == 0:
    output = 'ZERO'
elif n <= 19:
    output = words_upto_19[n]
elif n<=99:
    output = words_for_tens[n//10]+' '+words_upto_19[n%10]
else:
    output = 'Pleae Enter a value fron 0 to 99 only'
print(output)
```

```
Enter a number from 0 to 99 : 19
NINETEEN
```

In [22]:

```
words_upto_19 = ['','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIGHT','NINE','TEN','E
                 'TWELVE','THIRTEEN','FOURTEEN','FIFTEEN','SIXTEEN','SEVENTEEN','EIGHTEEN',

words_for_tens = ['','','TWENTY','THIRTY','FORTY','FIFTY','SIXTY','SEVENTY','EIGHTY','NINET

n = int(input('Enter a number from 0 to 99 : '))

output = ''

if n == 0:
    output = 'ZERO'
elif n <= 19:
    output = words_upto_19[n]
elif n<=99:
    output = words_for_tens[n//10]+' '+words_upto_19[n%10]
else:
    output = 'Pleae Enter a value fron 0 to 99 only'
print(output)
```

```
Enter a number from 0 to 99 : 25
TWENTY FIVE
```

In [25]:

```
words_upto_19 = ['','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIGHT','NINE','TEN','E
                 'TWELVE','THIRTEEN','FOURTEEN','FIFTEEN','SIXTEEN','SEVENTEEN','EIGHTEEN',

words_for_tens = ['','','TWENTY','THIRTY','FORTY','FIFTY','SIXTY','SEVENTY','EIGHTY','NINET

n = int(input('Enter a number from 0 to 99 : '))

output = ''

if n == 0:
    output = 'ZERO'
elif n <= 19:
    output = words_upto_19[n]
elif n<=99:
    output = words_for_tens[n//10]+' '+words_upto_19[n%10]
else:
    output = 'Pleae Enter a value fron 0 to 99 only'
print(output)
```

```
Enter a number from 0 to 99 : 40
FORTY
```

In [24]:

```
words_upto_19 = ['','ONE','TWO','THREE','FOUR','FIVE','SIX','SEVEN','EIGHT','NINE','TEN','E
                 'TWELVE','THIRTEEN','FOURTEEN','FIFTEEN','SIXTEEN','SEVENTEEN','EIGHTEEN',

words_for_tens = ['','','TWENTY','THIRTY','FORTY','FIFTY','SIXTY','SEVENTY','EIGHTY','NINET

n = int(input('Enter a number from 0 to 99 : '))

output = ''

if n == 0:
    output = 'ZERO'
elif n <= 19:
    output = words_upto_19[n]
elif n<=99:
    output = words_for_tens[n//10]+' '+words_upto_19[n%10]
else:
    output = 'Pleae Enter a value fron 0 to 99 only'
print(output)
```

```
Enter a number from 0 to 99 : 123
Pleae Enter a value fron 0 to 99 only
```

Assignment: Extend the above example to the range 0 to 999

2. Iterative Statements

If we want to execute a group of statements multiple times then we should go for Iterative statements.

Python supports 2 types of iterative statements.

```
i. for loop

ii. while loop
```

i) for loop:

- If we want to execute some action for every element present in some sequence (it may be string or collection) then we should go for **for loop**.

Syntax:

for x in sequence:

```
body
```

where 'sequence' can be string or any collection.

Body will be executed for every element present in the sequence.

Eg 1: Write a Program to print characters present in the given string.

In [27]:

```python
s="Sahasra"
for x in s :
    print(x)
```

```
S
a
h
a
s
r
a
```

Eg 2: To print characters present in string index wise.

In [28]:

```
s=input("Enter some String: ")
i=0
for x in s :
    print("The character present at ",i,"index is :",x)
    i=i+1
```

```
Enter some String: Karthikeya
The character present at  0 index is : K
The character present at  1 index is : a
The character present at  2 index is : r
The character present at  3 index is : t
The character present at  4 index is : h
The character present at  5 index is : i
The character present at  6 index is : k
The character present at  7 index is : e
The character present at  8 index is : y
The character present at  9 index is : a
```

Eg 3: To print Hello 10 times.

In [31]:

```
s = 'Hello'
for i in range(1,11):
    print(s)
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

In [32]:

```
s = 'Hello'
for i in range(10):
    print(s)
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

Eg 4: To display numbers from 0 to 10

In [33]:

```
for i in range(0,11):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

Eg 5: To display odd numbers from 0 to 20

In [34]:

```
for i in range(21):
    if(i%2!=0):
        print(i)
```

```
1
3
5
7
9
11
13
15
17
19
```

Eg 6: To display numbers from 10 to 1 in descending order.

In [35]:

```
for i in range(10,0,-1):
    print(i)
```

```
10
9
8
7
6
5
4
3
2
1
```

Eg 7: To print sum of numbers presenst inside list.

In [36]:

```
list=eval(input("Enter List:"))
sum=0;
for x in list:
    sum=sum+x;
print("The Sum=",sum)
```

```
Enter List:10,20,30,40
The Sum= 100
```

In [42]:

```
list=eval(input("Enter List:"))
sum=0;
for x in list:
    sum=sum+ x;
print("The Sum=",sum)
```

```
Enter List:45,67,89
The Sum= 201
```

ii) while loop:

If we want to execute a group of statements iteratively until some condition false,then we should go for while loop.

Syntax:

while condition:

```
    body
```

Eg 1: To print numbers from 1 to 10 by using while loop

In [43]:

```
x=1
while x <=10:
    print(x)
    x=x+1
```

```
1
2
3
4
5
6
7
8
9
10
```

Eg 2: To display the sum of first n numbers.

In [44]:

```
n=int(input("Enter number:"))
sum=0
i=1
while i<=n:
    sum=sum+i
    i=i+1
print("The sum of first",n,"numbers is :",sum)
```

```
Enter number:10
The sum of first 10 numbers is : 55
```

Date: 23-04-2020 Day-2

Eg 3: write a program to prompt user to enter some name until entering Karthi.

In [1]:

```
name=""
while name!="Karthi":
    name=input("Enter Name:")
print("Thanks for confirmation")
```

```
Enter Name:ramu
Enter Name:raju
Enter Name:Karthi
Thanks for confirmation
```

Infinite Loops

Some times a loop can execute infinite number of times without stopping also.

Eg :

In [ ]:

```
i = 1
while True:        # The body of this while Loop keep on execuing because condition is alwa
    print('Hello', i) # This program never going to terminates
    i=i+1
```

Note: By pressing **Ctrl + C** we can stop this program.

By mistake, if our program entered into an infinite loop, how we can solve this prolem, where we have the above problem requirement.

while True:

```
    body


    if our required condition satisfied
    break
```

If you are using **break** statement, you will come out from the loop.

Nested Loops

Sometimes we can take a loop inside another loop,which are also known as **nested loops.**

Eg 1:

In [1]:

```
for i in range(3):
    for j in range(2):
        print('Hello')
```

```
Hello
Hello
Hello
Hello
Hello
Hello
```

Eg 2:

In [6]:

```
for i in range(4):
    for j in range(4):
        #print("i=",i," j=",j)
        print('i = {}   j = {}'.format(i,j))
```

```
i = 0   j = 0
i = 0   j = 1
i = 0   j = 2
i = 0   j = 3
i = 1   j = 0
i = 1   j = 1
i = 1   j = 2
i = 1   j = 3
i = 2   j = 0
i = 2   j = 1
i = 2   j = 2
i = 2   j = 3
i = 3   j = 0
i = 3   j = 1
i = 3   j = 2
i = 3   j = 3
```

Eg :3

Q. Write a program to dispaly *'s in Right angled triangled form.

In [3]:

```
n = int(input("Enter number of rows:"))
for i in range(1,n+1):
    for j in range(1,i+1):
        print("*",end=" ")
    print()
```

```
Enter number of rows:7
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
```

Alternative way:

In [7]:

```
n = int(input("Enter number of rows:"))
for i in range(1,n+1):
    print("* " * i)
```

```
Enter number of rows:7
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
```

Eg 4:

Q. Write a program to display *'s in pyramid style (also known as equivalent triangle)

In [9]:

```
n = int(input("Enter number of rows:"))
for i in range(1,n+1):
    print(" " * (n),end="")                    # Righ angle Triangle form
    print("* "*i)
```

```
Enter number of rows:7
       *
       * *
       * * *
       * * * *
       * * * * *
       * * * * * *
       * * * * * * *
```

In [13]:

```
n = int(input("Enter number of rows:"))
for i in range(1,n+1):
    print(" " * (n-i),end="")          # Equivalent Triangle form
    print("* "*i)
```

```
Enter number of rows:7
      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * * *
* * * * * * *
```

Transfer Statements

i) break:

- We can use break statement inside loops to break loop execution based on some condition.

Eg :

In [27]:

```
for i in range(10):
    if i==7:
        print("processing is enough..plz break")
        break
    print(i)
```

```
0
1
2
3
4
5
6
processing is enough..plz break
```

Eg :

In [28]:

```
cart=[10,20,600,60,70]
for item in cart:
    if item>500:
        print("To place this order insurence must be required")
        break
    print(item)
```

```
10
20
To place this order insurence must be required
```

ii) continue:

- We can use continue statement to skip current iteration and continue next iteration.

Eg 1: To print odd numbers in the range 0 to 9.

In [29]:

```
for i in range(10):
    if i%2==0:
        continue
    print(i)
```

```
1
3
5
7
9
```

Eg 2:

In [31]:

```
cart=[10,20,500,700,50,60]
for item in cart:
    if item >= 500:
        print("We cannot process this item :",item)
        continue
    print(item)
```

```
10
20
We cannot process this item : 500
We cannot process this item : 700
50
60
```

Eg 3:

In [33]:

```
numbers=[10,20,0,5,0,30]
for n in numbers:
    if n==0:
        print("Hey how we can divide with zero..just skipping")
        continue
    print("100/{} = {}".format(n,100/n))
```

```
100/10 = 10.0
100/20 = 5.0
Hey how we can divide with zero..just skipping
100/5 = 20.0
Hey how we can divide with zero..just skipping
100/30 = 3.3333333333333335
```

Loops with else block:

- Inside loop execution,if break statement not executed ,then only else part will be executed.
- else means loop without break

Eg 1:

In [35]:

```
cart=[10,20,30,40,50]
for item in cart:
    if item>=500:
        print("We cannot process this order")
        break
    print(item)
else:
    print("Congrats ...all items processed successfully")
```

```
10
20
30
40
50
Congrats ...all items processed successfully
```

Eg 2:

In [36]:

```
cart=[10,20,600,30,40,50]
for item in cart:
    if item>=500:
        print("We cannot process this order")
        break
    print(item)
else:
    print("Congrats ...all items processed successfully")
```

```
10
20
We cannot process this order
```

Questions:

Q 1. What is the difference between for loop and while loop in Python?

- We can use loops to repeat code execution
- Repeat code for every item in sequence ==>for loop
- Repeat code as long as condition is true ==>while loop

Q 2. How to exit from the loop?

- by using break statement

Q 3. How to skip some iterations inside loop?

- by using continue statement.

Q 4. When else part will be executed wrt loops?

- If loop executed without break

iii) pass statement:

- pass is a keyword in Python.
- In our programming syntactically if block is required which won't do anything then we can define that empty block with pass keyword.
- pass statement --

```
    - It is an empty statement


    - It is null statement


    - It won't do anything
```

Eg :

In [37]:

```
if True:              # It is invalid
```

```
  File "<ipython-input-37-2c8a33b52dfb>", line 1
    if True:
            ^
SyntaxError: unexpected EOF while parsing
```

In [39]:

```
if True:
    pass           # It is valid
```

In [40]:

```
def m1():       # It is invalid
```

```
  File "<ipython-input-40-e96db426b315>", line 1
    def m1():
             ^
SyntaxError: unexpected EOF while parsing
```

In [41]:

```
def m1():
    pass            # It is valid
```

use case of pass:

- Sometimes in the parent class we have to declare a function with empty body and child class responsible to provide proper implementation. Such type of empty body we can define by using pass keyword. (It is something like abstract method in java).

Eg :

In [42]:

```
for i in range(100):
    if i%9==0:
        print(i)
    else:
        pass
```

```
0
9
18
27
36
45
54
63
72
81
90
99
```

iv) del statement:

- del is a keyword in Python.
- After using a variable, it is highly recommended to delete that variable if it is no longer required,so that the corresponding object is eligible for Garbage Collection. We can delete variable by using 'del' keyword.

Eg :

In [44]:

```
x=10
print(x)
del x
```

```
10
```

After deleting a variable we cannot access that variable otherwise we will get NameError.

In [45]:

```
x = 10
del(x)
print(x)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-45-f85867067c4c> in <module>
      1 x = 10
      2 del(x)
----> 3 print(x)

NameError: name 'x' is not defined
```

Note:

- We can delete variables which are pointing to immutable objects.But we cannot delete the elements present inside immutable object.

Eg :

In [46]:

```
s="karthi"
print(s)
del s           #valid
del s[0]        #TypeError: 'str' object doesn't support item deletion
```

```
karthi

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-46-7771d7c31337> in <module>
      2 print(s)
      3 del s           #valid
----> 4 del s[0]        #TypeError: 'str' object doesn't support item delet
ion

NameError: name 's' is not defined
```

Difference between 'del' and 'None':

In the case del, the variable will be removed and we cannot access that variable(unbind operation).

In [47]:

```
s="karthi"
del s
print(s)     # NameError: name 's' is not defined.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-47-43859941d54d> in <module>
      1 s="durga"
      2 del s
----> 3 print(s)     # NameError: name 's' is not defined.

NameError: name 's' is not defined
```

But in the case of None assignment the variable won't be removed but the corresponding object is eligible for Garbage Collection(re bind operation). Hence after assigning with None value,we can access that variable.

In [48]:

```
s="karthi"
s=None
print(s)
```

```
None
```

In [ ]:

```

```

UNIT - 3

String Data Type

Date: 24-04-2020 Day-1

The most commonly used object in any project and in any programming language is String only. Hence we should aware complete information about String data type.

What is String?

Any sequence of characters within either single quotes or double quotes is considered as a String.

Syntax:

s='karthi'

s="karthi"

Note:

In most of other languges like C, C++,Java, a single character with in single quotes is treated as char data type value. But in Python we are not having char data type.Hence it is treated as String only.

In [19]:

```
ch = 'a'
print(type(ch))
```

```
<class 'str'>
```

How to define multi-line String literals?

1. We can define multi-line String literals by using triple single or double quotes.

In [20]:

```
s = '''karthi
sahasra
sri'''
print(s)                    # Multi line strings
```

```
karthi
sahasra
sri
```

In [21]:

```
s = """karthi
sahasra
sri"""
print(s)
```

```
karthi
sahasra
sri
```

2. We can also use triple quotes to use single quotes or double quotes as symbol inside String literal.

In [22]:

```
s='This is ' single quote symbol'
```

```
  File "<ipython-input-22-8b8100e80f54>", line 1
    s='This is ' single quote symbol'
                      ^
SyntaxError: invalid syntax
```

In [23]:

```
s='This is \' single quote symbol'
print(s)
```

```
This is ' single quote symbol
```

In [24]:

```
s="This is ' single quote symbol"
print(s)
```

```
This is ' single quote symbol
```

In [25]:

```
s='This is " double quotes symbol'
print(s)
```

```
This is " double quotes symbol
```

In [26]:

```
s='The "Python Notes" by 'durga' is very helpful'
```

```
  File "<ipython-input-26-476c982681ef>", line 1
    s='The "Python Notes" by 'durga' is very helpful'
                                  ^
SyntaxError: invalid syntax
```

In [27]:

```
s="The "Python Notes" by 'durga' is very helpful"
```

```
  File "<ipython-input-27-182ab0922e87>", line 1
    s="The "Python Notes" by 'durga' is very helpful"
                 ^
SyntaxError: invalid syntax
```

In [28]:

```
s='The \"Python Notes\" by \'durga\' is very helpful'
print(s)
```

```
The "Python Notes" by 'durga' is very helpful
```

In [29]:

```
s='''The "Python Notes" by 'durga' is very helpful'''
print(s)
```

```
The "Python Notes" by 'durga' is very helpful
```

How to access characters of a String?

We can access characters of a string by using the following ways.

1. By using index
2. By using slice operator

1. By using index:

- Python supports both +ve and -ve index.
- +ve index means left to right(Forward direction)
- -ve index means right to left(Backward direction)

In [30]:

```
s = 'Karthi'
print(s[0])
print(s[5])
print(s[-1])
print(s[19])
```

```
K
i
i

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-30-d39a6b459de8> in <module>
      3 print(s[5])
      4 print(s[-1])
----> 5 print(s[19])

IndexError: string index out of range
```

Note:

- If we are trying to access characters of a string with out of range index then we will get error saying : IndexError

Eg: Q 1. Write a program to accept some string from the keyboard and display its characters by index wise(both positive and negative index)

In [31]:

```
s=input("Enter Some String:")
i=0
for x in s:
    print("The character present at positive index {} and at negative index {} is {}".forma
    i=i+1
```

```
Enter Some String:karthikeya
The character present at positive index 0 and at negative index -10 is k
The character present at positive index 1 and at negative index -9 is a
The character present at positive index 2 and at negative index -8 is r
The character present at positive index 3 and at negative index -7 is t
The character present at positive index 4 and at negative index -6 is h
The character present at positive index 5 and at negative index -5 is i
The character present at positive index 6 and at negative index -4 is k
The character present at positive index 7 and at negative index -3 is e
The character present at positive index 8 and at negative index -2 is y
The character present at positive index 9 and at negative index -1 is a
```

Date: 26-04-2020 Day-2

2. Accessing characters by using slice operator:

- string slice means a part of the string (i.e, Sub string).

Syntax:

string_Name [beginindex:endindex:step]

Here,

- **beginindex:** From where we have to consider slice(substring)
- **endindex:** We have to terminate the slice(substring) at endindex-1
- **step:** incremented / decremented value

Note :

- **Slicing operator returns the sub string form beginindex to endindex - 1**

Note:

- If we are not specifying begin index then it will consider from beginning of the string.
- If we are not specifying end index then it will consider up to end of the string.
- The default value for step is 1

Eg 1:

In [32]:

```
s = 'abcdefghijk'
print(s[2:7])
```

```
cdefg
```

In [33]:

```
s = 'abcdefghijk'
print(s[:7])
```

```
abcdefg
```

In [34]:

```
s = 'abcdefghijk'
print(s[2:])
```

```
cdefghijk
```

In [35]:

```
s = 'abcdefghijk'
print(s[:])
```

```
abcdefghijk
```

In [36]:

```
s = 'abcdefghijk'
print(s[2:7:1])
```

```
cdefg
```

In [37]:

```
s = 'abcdefghijk'
print(s[2:7:2])
```

```
ceg
```

In [38]:

```
s = 'abcdefghijk'
print(s[2:7:3])
```

```
cf
```

In [39]:

```
s = 'abcdefghijk'
print(s[::1])
```

```
abcdefghijk
```

In [40]:

```
s = 'abcdefghijk'
print(s[::2])
```

```
acegik
```

In [41]:

```
s = 'abcdefghijk'
print(s[::3])
```

```
adgj
```

Eg 2:

In [11]:

```
s="Learning Python is very very easy!!!"
s[1:7:1]
```

Out[11]:

```
'earnin'
```

In [12]:

```
s="Learning Python is very very easy!!!"
s[1:7]
```

Out[12]:

```
'earnin'
```

In [13]:

```
s="Learning Python is very very easy!!!"
s[1:7:2]
```

Out[13]:

```
'eri'
```

In [14]:

```
s="Learning Python is very very easy!!!"
s[:7]
```

Out[14]:

```
'Learnin'
```

In [15]:

```
s="Learning Python is very very easy!!!"
s[7:]
```

Out[15]:

```
'g Python is very very easy!!!'
```

In [16]:

```
s="Learning Python is very very easy!!!"
s[::]
```

Out[16]:

```
'Learning Python is very very easy!!!'
```

In [17]:

```
s="Learning Python is very very easy!!!"
s[:]
```

Out[17]:

```
'Learning Python is very very easy!!!'
```

In [18]:

```
s="Learning Python is very very easy!!!"
s[::-1]                                         # Reverse of the string
```

Out[18]:

```
'!!!ysae yrev yrev si nohtyP gninraeL'
```

Behaviour of slice operator:

s[begin:end:step]

- Here, step value can be either +ve or –ve
- if +ve then it should be forward direction(left to right) and we have to consider begin to end-1
- if -ve then it should be backward direction(right to left) and we have to consider begin to end+1

Note:

- In the backward direction if end value is -1 then result is always empty.
- In the forward direction if end value is 0 then result is always empty.

In forward direction:

- default value for begin: 0
- default value for end: length of string
- default value for step: +1

In backward direction:

- default value for begin: -1
- default value for end: -(length of string + 1)

Note:

- Either forward or backward direction, we can take both +ve and -ve values for begin and end index.

Mathematical Operators for String:

We can apply the following mathematical operators for Strings.

1. • operator for concatenation
2. • operator for repetition

In [10]:

```
print("karthi" + "sahasra")

print("karthi"*2)
```

```
karthisahasra
karthikarthi
```

Note:

1. To use + operator for Strings, compulsory both arguments should be str type.
2. To use * operator for Strings, compulsory one argument should be str and other argument should be int.

len() in-built function:

We can use len() function to find the number of characters present in the string.

Eg:

In [2]:

```
s='karthi'
print(len(s)) #5
```

```
6
```

Eg : Q. Write a Python program to access each character of string in forward and backward direction by using while loop.

In [3]:

```
s="Learning Python is very easy !!!"
n=len(s)
i=0

print("Forward direction")
print()
while i<n:
    print(s[i],end=' ')
    i +=1
print('')
print('')
print("Backward direction")
print()
i=-1
while i>=-n:
    print(s[i],end=' ')
    i=i-1
```

```
Forward direction

L e a r n i n g   P y t h o n   i s   v e r y   e a s y   ! ! !

Backward direction

! ! !   y s a e   y r e v   s i   n o h t y P   g n i n r a e L
```

Alternative way [Using slice operator]:

In [4]:

```
s="Learning Python is very easy !!!"

print("Forward direction")
print('')
for i in s:
    print(i,end=' ')
print('')
print('')

print("Forward direction")
print('')
for i in s[::]:
    print(i,end=' ')
print('')
print('')
print('Backward Direction')
print('')
for i in s[::-1]:
    print(i,end=' ')
```

```
Forward direction

L e a r n i n g   P y t h o n   i s   v e r y   e a s y   ! ! !

Forward direction

L e a r n i n g   P y t h o n   i s   v e r y   e a s y   ! ! !

Backward Direction

! ! !   y s a e   y r e v   s i   n o h t y P   g n i n r a e L
```

Eg 2:

In [5]:

```
s = input('Enter the string')
print('Data in Farward Direction')
for i in s:
    print(i,end='')
print()
print('Data in Backward Direction')
for i in s[::-1]:
    print(i,end='')
```

```
Enter the stringkarthi
Data in Farward Direction
karthi
Data in Backward Direction
ihtrak
```

Alternative Way :

In [7]:

```
s = input('Enter the string : ')
print('Data in Farward Direction')
print(s[::1])
print()
print('Data in Backward Direction')
print(s[::-1])
```

```
Enter the string : karthi
Data in Farward Direction
karthi

Data in Backward Direction
ihtrak
```

In [8]:

```
s = 'karthikeya'
print(s[-1:-1:-1])      # It results an empty string
```

In [9]:

```
s = 'karthikeya'
print(s[-1:0:1])      # It results an empty string
```

Date: 27-04-2020 Day 3

Membership Operators:

We can check whether the character or string is the member of another string or not by using following membership operators:

1. in

2. not in

In [42]:

```
s = 'karthi'
print('r' in s)
```

```
True
```

In [43]:

```
s = 'karthi'
print('p' in s)
```

```
False
```

In [44]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
if subs in s:
     print(subs,"is found in main string")
else:
    print(subs,"is not found in main string")
```

```
Enter main string:karthikeya
Enter sub string:thi
thi is found in main string
```

In [45]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
if subs in s:
     print(subs,"is found in main string")
else:
    print(subs,"is not found in main string")
```

```
Enter main string:karthi
Enter sub string:saha
saha is not found in main string
```

Comparison of Strings:

- We can use comparison operators (<,<=,>,>=) and equality operators(==,!=) for strings.
- Comparison will be performed based on alphabetical order.

Eg :

In [46]:

```
s1=input("Enter first string:")
s2=input("Enter Second string:")
if s1==s2:
    print("Both strings are equal")
elif s1<s2:
     print("First String is less than Second String")
else:
     print("First String is greater than Second String")
```

```
Enter first string:karthi
Enter Second string:karthi
Both strings are equal
```

In [47]:

```
s1=input("Enter first string:")
s2=input("Enter Second string:")
if s1==s2:
    print("Both strings are equal")
elif s1<s2:
     print("First String is less than Second String")
else:
     print("First String is greater than Second String")
```

```
Enter first string:karthi
Enter Second string:sahasra
First String is less than Second String
```

In [48]:

```
s1=input("Enter first string:")
s2=input("Enter Second string:")
if s1==s2:
    print("Both strings are equal")
elif s1<s2:
     print("First String is less than Second String")
else:
     print("First String is greater than Second String")
```

```
Enter first string:sahasra
Enter Second string:karthi
First String is greater than Second String
```

Note :

- s1 == s2 ====> Content Comparison
- s1 is s2 ====> Reference Comparison

Removing spaces from the string:

To remove the blank spaces present at either beginning and end of the string, we can use the following 3 methods:

1.rstrip() ===>To remove blank spaces present at end of the string (i.e.,right hand side)

2. lstrip()===>To remove blank spaces present at the beginning of the string (i.e.,left hand side)

3. strip() ==>To remove spaces both sides

Eg :

In [49]:

```
city=input("Enter your city Name:")
scity=city.strip()
if scity=='Hyderabad':
    print("Hello Hyderbadi..Adab")
elif scity=='Chennai':
    print("Hello Madrasi...Vanakkam")
elif scity=="Bangalore":
    print("Hello Kannadiga...Shubhodaya")
else:
    print("your entered city is invalid")
```

```
Enter your city Name:Hyderabad
Hello Hyderbadi..Adab
```

In [50]:

```
city=input("Enter your city Name:")
scity=city.strip()
if scity=='Hyderabad':
    print("Hello Hyderbadi..Adab")
elif scity=='Chennai':
    print("Hello Madrasi...Vanakkam")
elif scity=="Bangalore":
    print("Hello Kannadiga...Shubhodaya")
else:
    print("your entered city is invalid")
```

```
Enter your city Name:Chennai
Hello Madrasi...Vanakkam
```

In [51]:

```
city=input("Enter your city Name:")
scity=city.strip()
if scity=='Hyderabad':
    print("Hello Hyderbadi..Adab")
elif scity=='Chennai':
    print("Hello Madrasi...Vanakkam")
elif scity=="Bangalore":
    print("Hello Kannadiga...Shubhodaya")
else:
    print("your entered city is invalid")
```

```
Enter your city Name:Bangalore
Hello Kannadiga...Shubhodaya
```

In [52]:

```
city=input("Enter your city Name:")
scity=city.strip()
if scity=='Hyderabad':
    print("Hello Hyderbadi..Adab")
elif scity=='Chennai':
    print("Hello Madrasi...Vanakkam")
elif scity=="Bangalore":
    print("Hello Kannadiga...Shubhodaya")
else:
    print("your entered city is invalid")
```

```
Enter your city Name:nandyal
your entered city is invalid
```

In [53]:

```
city=input("Enter your city Name:")
#scity=city.strip()
if city=='Hyderabad':
    print("Hello Hyderbadi..Adab")
elif scity=='Chennai':
    print("Hello Madrasi...Vanakkam")
elif scity=="Bangalore":
    print("Hello Kannadiga...Shubhodaya")
else:
    print("your entered city is invalid")
```

```
Enter your city Name: Hyderabad
your entered city is invalid
```

In [54]:

```
city=input("Enter your city Name:")
scity=city.strip()
if scity=='Hyderabad':
    print("Hello Hyderbadi..Adab")
elif scity=='Chennai':
    print("Hello Madrasi...Vanakkam")
elif scity=="Bangalore":
    print("Hello Kannadiga...Shubhodaya")
else:
    print("your entered city is invalid")
```

```
Enter your city Name:     Hyderabad
Hello Hyderbadi..Adab
```

In [55]:

```
city=input("Enter your city Name:")
scity=city.strip()
if scity=='Hyderabad':
    print("Hello Hyderbadi..Adab")
elif scity=='Chennai':
    print("Hello Madrasi...Vanakkam")
elif scity=="Bangalore":
    print("Hello Kannadiga...Shubhodaya")
else:
    print("your entered city is invalid")
```

```
Enter your city Name:   Hyderabad
Hello Hyderbadi..Adab
```

Note :

- In the middle of the string, if the blank spaces are present then the above specified functions can't do anything.

Finding Substrings:

If you want to find whether the substring is available in the given string or not in Python, we have 4 methods.

For forward direction:

1. find()
2. index()

For backward direction:

1. rfind()

2. rindex()

1. find() :

s.find(substring) (Without Boundary)

- Returns index of first occurrence of the given substring. If it is not available then we will get -1

In [56]:

```
s="Learning Python is very easy"
print(s.find("Python"))      # 9
print(s.find("Java"))        # -1
print(s.find("r"))           #  3
print(s.rfind("r"))          # 21
```

```
9
-1
3
21
```

- By default **find()** method can search total string. We can also specify the boundaries to search.

Syntax:

s.find(substring,begin,end) (With Boundary)

- It will always search from begin index to end-1 index.

Eg :

In [57]:

```
s="karthikeyasahasra"
print(s.find('a'))        #1
print(s.find('a',7,15))   #9
print(s.find('z',7,15))   #-1
```

```
1
9
-1
```

2. index() method:

- index() method is exactly same as find() method except that if the specified substring is not available then we will get ValueError.

In [58]:

```
s = 'abbaaaaaaaaaaaaaaaaabbababa'
print(s.index('bb',2,15))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-58-136a30f266f9> in <module>
      1 s = 'abbaaaaaaaaaaaaaaaaabbababa'
----> 2 print(s.index('bb',2,15))

ValueError: substring not found
```

In [59]:

```
s = 'abbaaaaaaaaaaaaaaaaabbababa'
print(s.index('bb'))
```

```
1
```

In [60]:

```
s = 'abbaaaaaaaaaaaaaaaaabbababa'
print(s.rindex('bb'))
```

```
20
```

Eg :

In [61]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
try:
    n=s.index(subs)
except ValueError:
    print("substring not found")
else:
    print("substring found")
```

```
Enter main string:karthikeya
Enter sub string:thi
substring found
```

In [62]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
try:
    n=s.index(subs)
except ValueError:
    print("substring not found")
else:
    print("substring found")
```

```
Enter main string:karthi
Enter sub string:saha
substring not found
```

Counting substring in the given String:

We can find the number of occurrences of substring present in the given string by using **count()** method.

1. s.count(substring) ==> It will search through out the string

2. s.count(substring, begin, end) ===> It will search from begin index to end-1 index

In [63]:

```
s="abcabcabcabcadda"
print(s.count('a'))      #6
print(s.count('ab'))     #4
print(s.count('a',3,7))  #2
```

```
6
4
2
```

In [64]:

```
s = 'abcdcdckk'
print(s.count('cdc'))
```

```
1
```

Date: 27-04-2020 Day 4

Eg : Q. Write a Python Program to display all positions of substring in a given main string.

In [65]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
flag=False
pos=-1
n=len(s)
c = 0
while True:
    pos=s.find(subs,pos+1,n)
    if pos==-1:
        break
    c = c+1
    print("Found at position",pos)
    flag=True
if flag==False:
    print("Not Found")
print('The number of occurrences : ',c)
```

```
Enter main string:abcabcabcaaa
Enter sub string:abc
Found at position 0
Found at position 3
Found at position 6
The number of occurrences :  3
```

In [66]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
flag=False
pos=-1
n=len(s)
c = 0
while True:
    pos=s.find(subs,pos+1,n)
    if pos==-1:
        break
    c = c+1
    print("Found at position",pos)
    flag=True
if flag==False:
    print("Not Found")
print('The number of occurrences : ',c)
```

```
Enter main string:abcabcabcaa
Enter sub string:bb
Not Found
The number of occurrences :  0
```

In [67]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
flag=False
pos=-1
n=len(s)
c = 0
while True:
    pos=s.find(subs,pos+1,n)
    if pos==-1:
        break
    c = c+1
    print("Found at position",pos)
    flag=True
if flag==False:
    print("Not Found")
print('The number of occurrences : ',c)
```

```
Enter main string:karthikeya
Enter sub string:k
Found at position 0
Found at position 6
The number of occurrences :  2
```

Alternate Way

In [68]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
i = s.find(subs)
if i == -1:
    print('Specified Substring is not found')
c = 0
while i !=- 1:
    c = c + 1
    print('{} is present at index: {}'.format(subs,i))
    i = s.find(subs,i+len(subs),len(s))
print('The number of occurrences : ',c)
```

```
Enter main string:karthikeya
Enter sub string:k
k is present at index: 0
k is present at index: 6
The number of occurrences :  2
```

In [70]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
i = s.find(subs)
if i == -1:
    print('Specified Substring is not found')
c = 0
while i !=- 1:
    c = c + 1
    print('{} is present at index: {}'.format(subs,i))
    i = s.find(subs,i+len(subs),len(s))
print('The number of occurrences : ',c)
```

```
Enter main string:abcabcabcaaa
Enter sub string:abc
abc is present at index: 0
abc is present at index: 3
abc is present at index: 6
The number of occurrences :  3
```

In [71]:

```
s=input("Enter main string:")
subs=input("Enter sub string:")
i = s.find(subs)
if i == -1:
    print('Specified Substring is not found')
c = 0
while i !=- 1:
    c = c + 1
    print('{} is present at index: {}'.format(subs,i))
    i = s.find(subs,i+len(subs),len(s))
print('The number of occurrences : ',c)
```

```
Enter main string:karthi
Enter sub string:saha
Specified Substring is not found
The number of occurrences :  0
```

In [ ]:

```
```

Date 28-04-2020 Day 4

Replacing a string with another string:

We can repalce a string with another string in python using a library function **replace()**.

Syntax:

```
s.replace(oldstring,newstring)
```

Here, inside **'s'**, every occurrence of oldstring will be replaced with newstring.

Eg :

In [1]:

```
s="Learning Python is very difficult"
s1=s.replace("difficult","easy")
print(s1)
```

```
Learning Python is very easy
```

Eg : All occurrences will be replaced

In [6]:

```
s="ababababababab"
print(id(s))
s1=s.replace("a","b")
print(id(s))
print(s1)
```

```
2168146994672
2168146994672
bbbbbbbbbbbbbb
```

In [5]:

```
s="ababababababab"
print(id(s))
s=s.replace("a","b")    # two objcets are created
print(id(s))
print(s)
```

```
2168149958000
2168149958128
bbbbbbbbbbbbbb
```

Q. String objects are immutable then how we can change the content by using replace() method.

- Once we creates string object, we cannot change the content.This non changeable behaviour is nothing but immutability. If we are trying to change the content by using any method, then with those changes a new object will be created and changes won't be happend in existing object.

- Hence with replace() method also a new object got created but existing object won't be changed.

Eg :

In [3]:

```
s="abab"
s1=s.replace("a","b")
print(s,"is available at :",id(s))
print(s1,"is available at :",id(s1))
```

```
abab is available at : 2168149950512
bbbb is available at : 2168149953312
```

In the above example, original object is available and we can see new object which was created because of replace() method.

Eg : Consider the string : Python is easy but Java is difficult.

How can you replace the string 'difficult' with 'easy' and 'easy' with 'difficult'.

In [7]:

```
s = 'Python is easy but Java is difficult'
s = s.replace('difficult','easy')
s = s.replace('easy','difficult')
print(s)                                    # it is not giving correct output
```

```
Python is difficult but Java is difficult
```

In [8]:

```
s = 'Python is easy but Java is difficult'
s = s.replace('difficult','d1')
s = s.replace('easy','e1')
print(s)
```

```
Python is e1 but Java is d1
```

In [10]:

```
s = 'Python is easy but Java is difficult'
s = s.replace('difficult','d1')
s = s.replace('easy','e1')
s = s.replace('d1','easy')
s = s.replace('e1','difficult')
print(s)
```

```
Python is difficult but Java is easy
```

Splitting of Strings:

- We can split the given string according to specified seperator by using **split() method**.
- We can split the given string according to specified seperator in reverse direction by using **rsplit() method**.

Syntax :

l=s.split(seperator, Maximum splits)

- Here, Both parameters are optional
- The default seperator is space.
- Maximum split defines maximum number of splits
- The return type of split() method is List.

rsplit() breaks the string at the seperator staring from the right and returns a list of strings.

Eg 1:

In [11]:

```
s="karthi sahasra sri"
l=s.split()
for x in l:
    print(x)
```

```
karthi
sahasra
sri
```

Eg 2:

In [12]:

```
s="22-02-2018"
l=s.split('-')
for x in l:
    print(x)
```

```
22
02
2018
```

In [14]:

```
s="22-02-2018"
l=s.split()      # no space in the string , so output is same as the given string
for x in l:
    print(x)
```

```
22-02-2018
```

In [1]:

```
s = 'karthi sahasra sri nandyal india'
l=s.split()
for x in l:
    print(x)
```

```
karthi
sahasra
sri
nandyal
india
```

In [10]:

```
s = 'karthi sahasra sri nandyal india'
l=s.rsplit(' ',3)       # rsplit(): from revesre direction it considers the given seperator
for x in l:
    print(x)
```

```
karthi sahasra
sri
nandyal
india
```

In [9]:

```
s = 'karthi sahasra sri nandyal ap india'
l=s.rsplit(' ',3)
for x in l:
    print(x)
```

```
karthi sahasra sri
nandyal
ap
india
```

In [7]:

```
s = 'karthi sahasra sri nandyal india'
l=s.lsplit(' ',3)              # there is no lsplit() methon in python
for x in l:
    print(x)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-7-0862072ad106> in <module>
      1 s = 'karthi sahasra sri nandyal india'
----> 2 l=s.lsplit(' ',3)
      3 for x in l:
      4     print(x)

AttributeError: 'str' object has no attribute 'lsplit'
```

In [11]:

```
s = '10,20,30,40,50,60,70,80'
l = s.split(',',3)
for x in l:
    print(x)
```

```
10
20
30
40,50,60,70,80
```

In [12]:

```
s = '10,20,30,40,50,60,70,80'
l = s.rsplit(',',3)
for x in l:
    print(x)
```

```
10,20,30,40,50
60
70
80
```

In [13]:

```
s = '10,20,30,40,50,60,70,80'
l = s.split(',',-1)
for x in l:
    print(x)
```

```
10
20
30
40
50
60
70
80
```

Joining of Strings:

We can join a group of strings(list or tuple) with respect to the given seperator.

Syntax :

s=seperator.join(group of strings)

Eg 1:

In [15]:

```
t=('sunny','bunny','chinny')
s='-'.join(t)
print(s)
```

```
sunny-bunny-chinny
```

Eg 2:

In [16]:

```
l=['hyderabad','singapore','london','dubai']
s=':'.join(l)
print(s)
```

```
hyderabad:singapore:london:dubai
```

In [17]:

```
l=['hyderabad','singapore','london','dubai']
s=''.join(l)
print(s)
```

```
hyderabadsingaporelondondubai
```

In [18]:

```
l=['hyderabad','singapore','london','dubai']
s=' '.join(l)
print(s)
```

```
hyderabad singapore london dubai
```

Changing case of a String:

We can change case of a string by using the following methods.

1. upper()===>To convert all characters to upper case

2. lower() ===>To convert all characters to lower case

3. swapcase()===>converts all lower case characters to upper case and all upper case characters to lower case

4. title() ===>To convert all characters to title case. i.e first character in every word should be upper case and all remaining characters should be in lower case.

5. capitalize() ==>Only first character will be converted to upper case and all remaining characters can be converted to lower case.

Eg :

In [19]:

```
s='learning Python is very Easy'
print(s.upper())
print(s.lower())
print(s.swapcase())
print(s.title())
print(s.capitalize())
```

```
LEARNING PYTHON IS VERY EASY
learning python is very easy
LEARNING pYTHON IS VERY eASY
Learning Python Is Very Easy
Learning python is very easy
```

Eg : Convert the uppercase chatacters into lowercase and remove spaces.

In [23]:

```
s='Learning Python Is Very Easy'
s = s.lower().replace(' ','')
print(s)
```

```
learningpythonisveryeasy
```

In [24]:

```
# Above example with join() & split() functions

s='Learning Python Is Very Easy'
s = s.lower()
s1 = s.split()
s = ''.join(s1)
print(s)
```

```
learningpythonisveryeasy
```

Checking starting and ending part of the string:

Python contains the following methods for this purpose

1. s.startswith(substring)
2. s.endswith(substring)

Eg :

In [25]:

```
s='learning Python is very easy'
print(s.startswith('learning'))
print(s.endswith('learning'))
print(s.endswith('easy'))
```

```
True
False
True
```

To check type of characters present in a string:

Python contains the following methods for this purpose.

1) isalnum(): Returns True if all characters are alphanumeric(a to z , A to Z ,0 to9)

2) isalpha(): Returns True if all characters are only alphabet symbols(a to z,A to Z)

3) isdigit(): Returns True if all characters are digits only(0 to 9)

4) islower(): Returns True if all characters are lower case alphabet symbols

5) isupper(): Returns True if all characters are upper case aplhabet symbols

6) istitle(): Returns True if string is in title case

7) isspace(): Returns True if string contains only spaces

Note : We can't pass any arguments to these functions.

In [28]:

```
print('Karthidurga786'.isalnum())  # True
print('Karthidurga786'.isalpha()) #False
print('Karthi'.isalpha()) #True
print('karthi'.isdigit()) #False
print('786786'.isdigit()) #True
print('abc'.islower()) #True
print('Abc'.islower()) #False
print('abc123'.islower()) #True
print('ABC'.isupper()) #True
print('Learning python is Easy'.istitle()) #False
print('Learning Python Is Easy'.istitle()) #True
print('             '.isspace()) #True
```

```
True
False
True
False
True
True
False
True
True
False
True
True
```

Date: 01-05-2020 Day 5

Demo Program:

In [14]:

```
s=input("Enter any character:")
if s.isalnum():
    print("Alpha Numeric Character")
    if s.isalpha():
        print("Alphabet character")
        if s.islower():
            print("Lower case alphabet character")
        else:
            print("Upper case alphabet character")
    else:
        print("it is a digit")
elif s.isspace():
    print("It is space character")
else:
    print("Non Space Special Character")
```

```
Enter any character:7
Alpha Numeric Character
it is a digit
```

In [15]:

```
s=input("Enter any character:")
if s.isalnum():
    print("Alpha Numeric Character")
    if s.isalpha():
        print("Alphabet character")
        if s.islower():
            print("Lower case alphabet character")
        else:
            print("Upper case alphabet character")
    else:
        print("it is a digit")
elif s.isspace():
    print("It is space character")
else:
    print("Non Space Special Character")
```

```
Enter any character:a
Alpha Numeric Character
Alphabet character
Lower case alphabet character
```

In [16]:

```
s=input("Enter any character:")
if s.isalnum():
    print("Alpha Numeric Character")
    if s.isalpha():
        print("Alphabet character")
        if s.islower():
            print("Lower case alphabet character")
        else:
            print("Upper case alphabet character")
    else:
        print("it is a digit")
elif s.isspace():
    print("It is space character")
else:
    print("Non Space Special Character")
```

```
Enter any character:A
Alpha Numeric Character
Alphabet character
Upper case alphabet character
```

In [18]:

```
s=input("Enter any character:")
if s.isalnum():
    print("Alpha Numeric Character")
    if s.isalpha():
        print("Alphabet character")
        if s.islower():
            print("Lower case alphabet character")
        else:
            print("Upper case alphabet character")
    else:
        print("it is a digit")
elif s.isspace():
    print("It is space character")
else:
    print("Non Space Special Character")
```

```
Enter any character: 
It is space character
```

In [17]:

```
s=input("Enter any character:")
if s.isalnum():
    print("Alpha Numeric Character")
    if s.isalpha():
        print("Alphabet character")
        if s.islower():
            print("Lower case alphabet character")
        else:
            print("Upper case alphabet character")
    else:
        print("it is a digit")
elif s.isspace():
    print("It is space character")
else:
    print("Non Space Special Character")
```

```
Enter any character:$
Non Space Special Character
```

Formatting the Strings:

We can format the strings with variable values by using **replacement operator {}** and **format() method.**

In [19]:

```
name='karthi'
salary=100000
age=6
print("{} 's salary is {} and his age is {}".format(name,salary,age))
print("{0} 's salary is {1} and his age is {2}".format(name,salary,age))
print("{x} 's salary is {y} and his age is {z}".format(z=age,y=salary,x=name))
```

```
karthi 's salary is 100000 and his age is 6
karthi 's salary is 100000 and his age is 6
karthi 's salary is 100000 and his age is 6
```

Important Programs regarding String Concept:

1. Write a program to reverse the given String.

Way 1:

In [20]:

```
s=input("Enter Some String:")
print(s[::-1])
```

```
Enter Some String:karthi
ihtrak
```

Way 2:

reversed():

In [22]:

```
s=input("Enter Some String:")
print((reversed(s)))
```

```
Enter Some String:karthi
<reversed object at 0x0000018959D1CD68>
```

In [24]:

```
s=input("Enter Some String:")
for x in (reversed(s)):
          print(x)
```

```
Enter Some String:karthi
i
h
t
r
a
k
```

In [21]:

```
s=input("Enter Some String:")
print(''.join(reversed(s)))
```

```
Enter Some String:karthi
ihtrak
```

Way 3:

In [25]:

```
s=input("Enter Some String:")
i=len(s)-1
target=''
while i>=0:
    target=target+s[i]
    i=i-1
print(target)
```

```
Enter Some String:karthi
ihtrak
```

Way 4:

In [27]:

```
s=input("Enter Some String:")
i=len(s)-1
target=''
for x in s:
    target=target+s[i]
    i=i-1
print(target)
```

```
Enter Some String:karthi
ihtrak
```

Way 5:

In [32]:

```
s=input("Enter Some String:")
i=len(s)-1
target=''
for x in range(len(s)-1,-1,-1):
    target = target + s[i]
    i = i-1
print(target)
```

```
Enter Some String:karthi
ihtrak
```

Q 2. Program to reverse order of words.

input: Learning Python is Very Easy

output: Easy Very is Python Learning

In [34]:

```
s=input("Enter Some String:")
l=s.split()
print(l)
print(len(l))
```

```
Enter Some String:karthi sahasra sri
['karthi', 'sahasra', 'sri']
3
```

In [35]:

```
s=input("Enter Some String:")
l=s.split()
l1=[]
i=len(l)-1
while i>=0:
    l1.append(l[i])
    i=i-1
output=' '.join(l1)
print(output)
```

```
Enter Some String:karthi sahasra sri
sri sahasra karthi
```

Q 3. Program to reverse internal content of each word.

input: Learning Python is Very Easy

output: gninreaL nohtyP si yreV ysaE

Way 1:

In [41]:

```
s=input("Enter Some String:")
l=s.split()
l1=[]
i = 0
while i<len(l):
    l1.append(l[i][::-1])
    i=i+1
output=' '.join(l1)
print(output)
```

```
Enter Some String:Learning Python Is Very Easy
gninraeL nohtyP sI yreV ysaE
```

Way 2:

In [44]:

```
s=input("Enter Some String:")
l=s.split()
l1=[]
for i in range(len(l)):
    s1 =l[i]
    l1.append(s1)
print(l1)
```

```
Enter Some String:Learning Python Is Very Easy
['Learning', 'Python', 'Is', 'Very', 'Easy']
```

In [45]:

```
s=input("Enter Some String:")
l=s.split()
l1=[]
for i in range(len(l)):
    s1 =l[i]
    l1.append(s1[::-1])
print(l1)
```

```
Enter Some String:Learning Python Is Very Easy
['gninraeL', 'nohtyP', 'sI', 'yreV', 'ysaE']
```

In [46]:

```
s=input("Enter Some String:")
l=s.split()
l1=[]
for i in range(len(l)):
    s1 =l[i]
    l1.append(s1[::-1])     # Appending reverse of s1 to the list l1
output=' '.join(l1)
print(output)
```

```
Enter Some String:Learning Python Is Very Easy
gninraeL nohtyP sI yreV ysaE
```

Way 3:

In [47]:

```
s=input("Enter Some String:")
l=s.split()
l1=[]
for x in l:
    l1.append(x[::-1])     # Appending reverse of s1 to the list l1
output=' '.join(l1)
print(output)
```

```
Enter Some String:Learning Python Is Very Easy
gninraeL nohtyP sI yreV ysaE
```

Q 4. Write a program to print characters at odd position and even position for the given String?

Way 1:

In [48]:

```
s=input("Enter Some String:")
print("Characters at Even Position:",s[0::2])
print("Characters at Odd Position:",s[1::2])
```

```
Enter Some String:karthikeya
Characters at Even Position: krhky
Characters at Odd Position: atiea
```

Way 2:

In [49]:

```
s=input("Enter Some String:")
i=0
print("Characters at Even Position:")
while i< len(s):
    print(s[i],end=',')
    i=i+2
print()
print("Characters at Odd Position:")
i=1
while i< len(s):
    print(s[i],end=',')
    i=i+2
```

```
Enter Some String:karthikeya
Characters at Even Position:
k,r,h,k,y,
Characters at Odd Position:
a,t,i,e,a,
```

Q 5. Program to merge characters of 2 strings into a single string by taking characters alternatively.

In [53]:

```
s1=input("Enter First String:")
s2=input("Enter Second String:")
output=''
i,j=0,0
while i<len(s1) or j<len(s2):
    output = output + s1[i]
    i = i + 1
    output = output + s2[j]
    j = j + 1
print(output)
```

```
Enter First String:ravi
Enter Second String:ramu
rraavmiu
```

Eventhough above code is working, there is some flaw in the code. This code works for the strings with same length. see the below code.

In [55]:

```
s1=input("Enter First String:")
s2=input("Enter Second String:")
output=''
i,j=0,0
while i<len(s1) or j<len(s2):
    output = output + s1[i]
    i = i + 1
    output = output + s2[j]
    j = j + 1
print(output)
```

```
Enter First String:karthi
Enter Second String:rgm

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-55-5dfcfbfd7737> in <module>
      6     output = output + s1[i]
      7     i = i + 1
----> 8     output = output + s2[j]
      9     j = j + 1
     10 print(output)

IndexError: string index out of range
```

Modified Code :

In [51]:

```
s1=input("Enter First String:")
s2=input("Enter Second String:")
output=''
i,j=0,0
while i<len(s1) or j<len(s2):
    if i<len(s1):
        output=output+s1[i]
        i+=1
    if j<len(s2):
        output=output+s2[j]
        j+=1
print(output)
```

```
Enter First String:karthi
Enter Second String:sahasra
ksaarhtahsira
```

In [52]:

```
s1=input("Enter First String:")
s2=input("Enter Second String:")
output=''
i,j=0,0
while i<len(s1) or j<len(s2):
    if i<len(s1):
        output=output+s1[i]
        i+=1
    if j<len(s2):
        output=output+s2[j]
        j+=1
print(output)
```

```
Enter First String:karthisahasra
Enter Second String:rgm
kragrmthisahasra
```

Q 6. Write a program to sort the characters of the string and first alphabet symbols followed by numeric values.

input: B4A1D3

Output: ABD134

sorted() function is used to sort the present content in side the string, which is given as an argument.

In [56]:

```
s=input("Enter Some String:")
s1=s2=output=''
for x in s:
    if x.isalpha():
        s1=s1+x               # All alphabets in s1
    else:
        s2=s2+x               # All digits in s2
for x in sorted(s1):          # In s1, whatever the content is there which will be sorted.
     output=output+x
for x in sorted(s2):          # # In s2, whatever the content is there which will be sortec
     output=output+x
print(output)
```

```
Enter Some String:b4a1d3
abd134
```

Optimized version of the above code:

In [59]:

```
s=input("Enter Some String:")
s1=s2=output=''
for x in s:
    if x.isalpha():
        s1=s1+x               # All alphabets in s1
    else:
        s2=s2+x               # All digits in s2
print(''.join(sorted(s1))+''.join(sorted(s2)))
```

```
Enter Some String:b4a1d3
abd134
```

Q 7. Write a program for the following requirement:

input: a4b3c2

output: aaaabbbcc

In [60]:

```
s=input("Enter Some String:")
output=''
for x in s:
    if x.isalpha():
        output=output+x
        previous=x
    else:
        output = output + previous*(int(x)-1)
print(output)
```

```
Enter Some String:a5b4c2
aaaaabbbbcc
```

Note:

- **chr(unicode)**===>The corresponding character
- **ord(character)**===>The corresponding unicode value

Q 8. Write a program to perform the following activity :

input: a4k3b2

output:aeknbd

In [1]:

```
s=input("Enter Some String:")
output=''
for x in s:
    if x.isalpha():
        output=output+x
        previous=x
    else:
        output=output+chr(ord(previous)+int(x))
print(output)
```

```
Enter Some String:a4k3b2
aeknbd
```

Q 9. Write a program to remove duplicate characters from the given input string.

Input: ABCDABBCDABBBCCCDDEEEF

Output: ABCDEF

In [2]:

```
s=input("Enter Some String:")
l=[]
for x in s:
    if x not in l:
        l.append(x)
    output=''.join(l)
print(output)
```

```
Enter Some String:abcdffaaannncd
abcdfn
```

Q 10. Write a program to find the number of occurrences of each character present in the given String.

Input: ABCABCABBCDE

Output: A-3,B-4,C-3,D-1,E-1

In [3]:

```
s=input("Enter the Some String:")
d={}
for x in s:
    if x in d.keys():
        d[x]=d[x]+1
    else:
        d[x]=1
for k,v in d.items():
    print("{} = {} Times".format(k,v))
```

```
Enter the Some String:ABCABCABBCDE
A = 3 Times
B = 4 Times
C = 3 Times
D = 1 Times
E = 1 Times
```

In [ ]:

```

```

UNIT - 4

List Data Type

Topics Covered:

1. Introduction

2. Creation of List Objects

3. Accessing elements of List

4. Traversing the elements of List

5. Important functions of List :

```
    i) To get information about list


    ii) Manipulating elements of List


    iii) Ordering elements of List
```

6. Aliasing and Cloning of List objects

7. Using Mathematical operators for List Objects

8. Comparing List objects

9. Membership operators

10. clear() function

11. Nested Lists

12. List Comprehensions

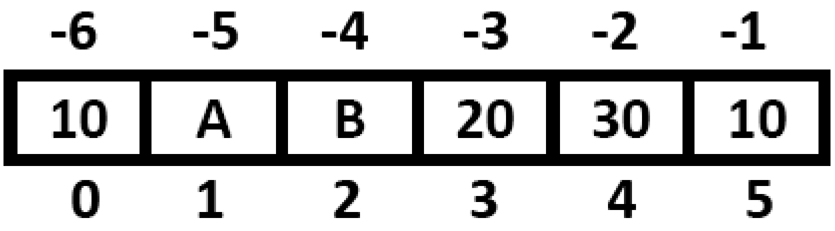
1. Introduction:

If we want to represent a **group of individual objects as a single entity** where insertion order is preserved and duplicates are allowed, then we should go for List.

- Insertion order preserved.
- Duplicate objects are allowed
- Heterogeneous objects are allowed.
- List is dynamic because based on our requirement we can increase the size and decrease the size.
- In List the elements will be placed within square brackets and with comma seperator.

- We can differentiate duplicate elements by using index and we can preserve insertion order by using index. Hence index will play very important role.
- Python supports both positive and negative indexes. +ve index means from left to right where as negative index means right to left.

Eg: [10,"A","B",20, 30, 10]



- List objects are mutable.i.e we can change the content.

2. Creation of List Objects:

1. We can create empty list object as follows...

In [2]:

```
list=[]
print(list)
print(type(list))
```

```
[]
<class 'list'>
```

2. If we know elements already then we can create list as follows

In [4]:

```
list = [10,20,30,40]
print(list)
print(type(list))
```

```
[10, 20, 30, 40]
<class 'list'>
```

3. With dynamic input:

In [5]:

```
list=(input("Enter List:")) # Entire input is considered as string
print(list)
print(type(list))
```

```
Enter List:10,20,30,40
10,20,30,40
<class 'str'>
```

In [6]:

```
list=eval(input("Enter List:"))
print(list)
print(type(list))
```

```
Enter List:[10,20,30,40]
[10, 20, 30, 40]
<class 'list'>
```

In [8]:

```
list=eval(input("Enter List:"))
print(list)
print(type(list))
```

```
Enter List:[ram,raj]

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-ac0b44db1317> in <module>
----> 1 list=eval(input("Enter List:"))
      2 print(list)
      3 print(type(list))

<string> in <module>

NameError: name 'ram' is not defined
```

In [9]:

```
list=eval(input("Enter List:"))
print(list)
print(type(list))
```

```
Enter List:['ram','raj']
['ram', 'raj']
<class 'list'>
```

4. With list() function:

In [ ]:

```
l=list(range(0,10,2))
print(l)                        # Not working in jupyter notebook but executed in Editplus
```

In [12]:

```
[0, 2, 4, 6, 8]
```

Out[12]:

```
[0, 2, 4, 6, 8]
```

In [ ]:

```
s="durga"
l=list(s)
print(l)            # Not working in jupyter notebook but executed in editplus
```

In [ ]:

```
['d', 'u', 'r', 'g', 'a']
```

5. With split() function:

In [16]:

```
s="Learning Python is very very easy !!!"
l=s.split()
print(l)
print(type(l))
```

```
['Learning', 'Python', 'is', 'very', 'very', 'easy', '!!!']
<class 'list'>
```

Note:

- Sometimes we can take list inside another list,such type of lists are called **nested lists.** [10,20,[30,40]]

3. Accessing elements of List:

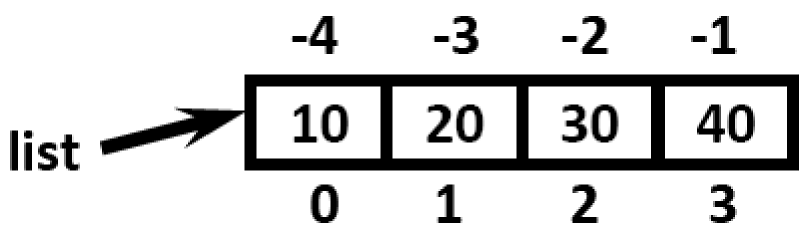
- We can access elements of the list either by using **index** or by using **slice operator**(:)

1. By using index:

- List follows zero based index. ie index of first element is zero.
- List supports both +ve and -ve indexes.
- +ve index meant for Left to Right.
- -ve index meant for Right to Left.

Eg :

list=[10,20,30,40]



In [20]:

```
list=[10,20,30,40]
print(list[0]) #10
print(list[-1]) #40
print(list[-4])  #10
print(list[10]) #IndexError: list index out of range
```

```
10
40
10

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-20-c9b501127143> in <module>
      3 print(list[-1]) #40
      4 print(list[-4])  #10
----> 5 print(list[10]) #IndexError: list index out of range

IndexError: list index out of range
```

In [23]:

```
list = [10,20,[30,40]]
print(list[2])
print(list[2][1])
```

```
[30, 40]
40
```

2. By using slice operator:

Syntax:

list2= list1[start:stop:step]

- start ==>it indicates the index where slice has to start default value is 0
- stop ===>It indicates the index where slice has to end default value is max allowed index of list ie length of the list
- step ==>increment value
- step default value is 1

In [24]:

```
l = [10,20,30,40,50,60]
print(l[::])
```

```
[10, 20, 30, 40, 50, 60]
```

In [25]:

```
l = [10,20,30,40,50,60]
print(l[::2])
```

```
[10, 30, 50]
```

In [26]:

```
l = [10,20,30,40,50,60]
print(l[::-1])
```

```
[60, 50, 40, 30, 20, 10]
```

In [27]:

```
l = [10,20,[30,40],50,60]
print(l[0:3:])
```

```
[10, 20, [30, 40]]
```

In [29]:

```
n=[1,2,3,4,5,6,7,8,9,10]
print(n[2:7:2])    #3,5,7
print(n[4::2])     # 5,7,9
print(n[3:7])      #4,5,6,7
print(n[8:2:-2])   # 9,7,5
print(n[4:100])    # 5,6,7,8,9,10
```

```
[3, 5, 7]
[5, 7, 9]
[4, 5, 6, 7]
[9, 7, 5]
[5, 6, 7, 8, 9, 10]
```

List vs mutability:

Once we creates a List object,we can modify its content. Hence List objects are mutable.

In [30]:

```
n=[10,20,30,40]
print(n)
n[1]=777
print(n)
```

```
[10, 20, 30, 40]
[10, 777, 30, 40]
```

Date: 02-05-2020 Day 2

4. Traversing the elements of List:

The sequential access of each element in the list is called traversal.

1. By using while loop:

In [2]:

```
n=[0,1,2,3,4,5,6,7,8,9,10]
i=0
while i<len(n):
    print(n[i])
    i=i+1
```

```
0
1
2
3
4
5
6
7
8
9
10
```

2. By using for loop:

In [3]:

```
n=[0,1,2,3,4,5,6,7,8,9,10]
for n1 in n:
    print(n1)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

3. To display only even numbers:

In [4]:

```
n=[0,1,2,3,4,5,6,7,8,9,10]
for n1 in n:
    if n1%2==0:
        print(n1)
```

```
0
2
4
6
8
10
```

4. To display elements by index wise:

In [5]:

```
l=["A","B","C"]
x=len(l)
for i in range(x):
    print(l[i],"is available at positive index: ",i,"and at negative index: ",i-x)
```

```
A is available at positive index:  0 and at negative index:  -3
B is available at positive index:  1 and at negative index:  -2
C is available at positive index:  2 and at negative index:  -1
```

5. Important functions of List:

What is the difference between function and method?

- In Python you can use both these terms interchangeably.

-Function:

- Function by default considered as method also.
- If a function is declaring outside a class is called as function.

- Method :

- If you are declaring a function inside a class is called as a method.
- In other words, if you are calling any function with object reference is called as method.

Note:

- Python is both functional oriented as well as object oriented programming language.

I. To get information about list:

1. len():

- returns the number of elements present in the list

In [7]:

```
n=[10,20,30,40]
print(len(n))
```

```
4
```

2. count():

- It returns the number of occurrences of specified item in the list.

In [8]:

```
n=[1,2,2,2,2,3,3]
print(n.count(1))
print(n.count(2))
print(n.count(3))
print(n.count(4))
```

```
1
4
2
0
```

3. index() function:

- returns the index of first occurrence of the specified item.

In [9]:

```
n=[1,2,2,2,2,3,3]
print(n.index(1)) # 0
print(n.index(2)) # 1
print(n.index(3)) # 5
print(n.index(4))
```

```
0
1
5

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-6eef43f6d8a4> in <module>
      3 print(n.index(2)) # 1
      4 print(n.index(3)) # 5
----> 5 print(n.index(4))

ValueError: 4 is not in list
```

Note:

- If the specified element not present in the list then we will get **ValueError**.
- Hence before index() method we have to check whether item present in the list or not by using **in** operator.

In [10]:

```
print( 4 in n)   #False
```

```
False
```

Eg:

In [14]:

```
l = [10,20,30,40,10,20,10,10]
target = int(input('Enter value to search : '))
if target in l:
    print(target,'available and its first occurrence is at ',l.index(target))
else:
    print(target,' is not available')b
```

```
Enter value to search : 50
50  is not available
```

In [15]:

```
l = [10,20,30,40,10,20,10,10]
target = int(input('Enter value to search : '))
if target in l:
    print(target,'available and its first occurrence is at ',l.index(target))
else:
    print(target,' is not available')
```

```
Enter value to search : 20
20 available and its first occurrence is at  1
```

In [16]:

```
l = [10,20,30,40,10,20,10,10]
target = int(input('Enter value to search : '))
if target in l:
    print(target,'available and its first occurrence is at ',l.index(target))
else:
    print(target,' is not available')
```

```
Enter value to search : 10
10 available and its first occurrence is at  0
```

II. Manipulating Elements of List:

1. append() function:

- We can use append() function to add item at the end of the list.
- By using this append function, we always add an element at last position.

In [11]:

```
list=[]
list.append("A")
list.append("B")
list.append("C")
print(list)
```

```
['A', 'B', 'C']
```

Eg: To add all elements to list upto 100 which are divisible by 10

In [17]:

```
list=[]
for i in range(101):
    if i%10==0:
        list.append(i)
print(list)
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Another Way:

In [19]:

```
list= []
for i in range(0,101,10):
    list.append(i)
print(list)
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

2. insert() function:

- To insert item at specified index position.

In [18]:

```
n=[1,2,3,4,5]
n.insert(1,888)
print(n)
```

```
[1, 888, 2, 3, 4, 5]
```

In [23]:

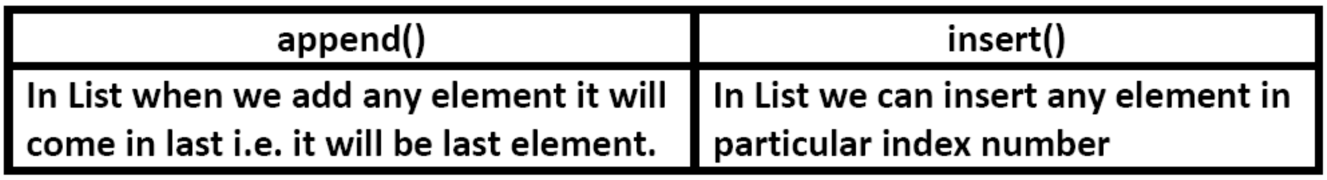
```
n=[1,2,3,4,5]
n.insert(10,777)
n.insert(-10,999)
print(n)
print(n.index(777))
print(n.index(999))
```

```
[999, 1, 2, 3, 4, 5, 777]
6
0
```

Note:

- If the specified index is greater than max index then element will be inserted at last position. If the specified index is smaller than min index then element will be inserted at first position.

Differences between append() and insert()



append()	insert()
In List when we add any element it will come in last i.e. it will be last element.	In List we can insert any element in particular index number

3. extend() function:

- To add all items of one list to another list,we use **extend()** method.

Eg:

l1.extend(l2)

- all items present in l2 will be added to l1

Eg 1:

In [24]:

```
order1=["Chicken","Mutton","Fish"]
order2=["RC","KF","FO"]
order1.extend(order2)
print(order1)
print(order2)
```

```
['Chicken', 'Mutton', 'Fish', 'RC', 'KF', 'FO']
['RC', 'KF', 'FO']
```

In [26]:

```
order1=["Chicken","Mutton","Fish"]
order2=["RC","KF","FO"]
order3 = order1 + order2
print(order1)
print(order2)
print(order3)
```

```
['Chicken', 'Mutton', 'Fish']
['RC', 'KF', 'FO']
['Chicken', 'Mutton', 'Fish', 'RC', 'KF', 'FO']
```

Eg 2:

In [1]:

```
l1 = [10,20,30]
l2 = [40,50,60]
l1.extend(l2)
print(l1)
```

```
[10, 20, 30, 40, 50, 60]
```

Eg 3:

In [2]:

```
order=["Chicken","Mutton","Fish"]
order.extend("Mushroom")     # It adds every character as a single element to the list
print(order)
```

```
['Chicken', 'Mutton', 'Fish', 'M', 'u', 's', 'h', 'r', 'o', 'o', 'm']
```

Explanation :

- Here, 'Mushroom' is a string type, in this string 8 elements are there. These elements are added seperately.

In [3]:

```
order=["Chicken","Mutton","Fish"]
order.append("Mushroom")      # It adds this string as a single element to the list
print(order)
```

```
['Chicken', 'Mutton', 'Fish', 'Mushroom']
```

Date: 03-05-2020 Day 3

4. remove() function:

- We can use this function to remove specified item from the list.
- If the item present multiple times then only first occurrence will be removed.

In [4]:

```
n=[10,20,10,30]
n.remove(10)
print(n)
```

```
[20, 10, 30]
```

If the specified item not present in list then we will get **ValueError**

In [5]:

```
n=[10,20,10,30]
n.remove(40)
print(n)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-75e98f1b4fac> in <module>
      1 n=[10,20,10,30]
----> 2 n.remove(40)
      3 print(n)

ValueError: list.remove(x): x not in list
```

Note:

- Hence before using remove() method first we have to check specified element present in the list or not by using in operator.

In [6]:

```
l1= [10,20,30,40,50,60,70]
x = int(input('Enter the element to be removed : '))
if x in l1:
    l1.remove(x)
    print('Element removed Successfully ')
    print(l1)
else:
    print('Specified element is not available ')
```

```
Enter the element to be removed : 10
Element removed Successfully
[20, 30, 40, 50, 60, 70]
```

In [7]:

```
l1= [10,20,30,40,50,60,70]
x = int(input('Enter the element to be removed : '))
if x in l1:
    l1.remove(x)
    print('Element removed Successfully ')
    print(l1)
else:
    print('Specified element is not available ')
```

```
Enter the element to be removed : 80
Specified element is not available
```

5. pop() function:

- It removes and returns the last element of the list.
- This is only function which manipulates list and returns some element.

Eg:

In [8]:

```
n=[10,20,30,40]
print(n.pop())
print(n.pop())
print(n)
```

```
40
30
[10, 20]
```

- If the list is empty then pop() function raises IndexError

In [9]:

```
n=[]
print(n.pop())
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-9-5146d826acc3> in <module>
      1 n=[]
----> 2 print(n.pop())

IndexError: pop from empty list
```

Note:

- 1. pop() is the only function which manipulates the list and returns some value
- 2. In general we can use append() and pop() functions to implement stack datastructure by using list,which follows LIFO(Last In First Out) order.
- 3. In general we can use pop() function to remove last element of the list. But we can use to remove elements based on index.

We can use pop() function in following ways:

- **n.pop(index)**===>To remove and return element present at specified index.
- **n.pop()**==>To remove and return last element of the list

In [11]:

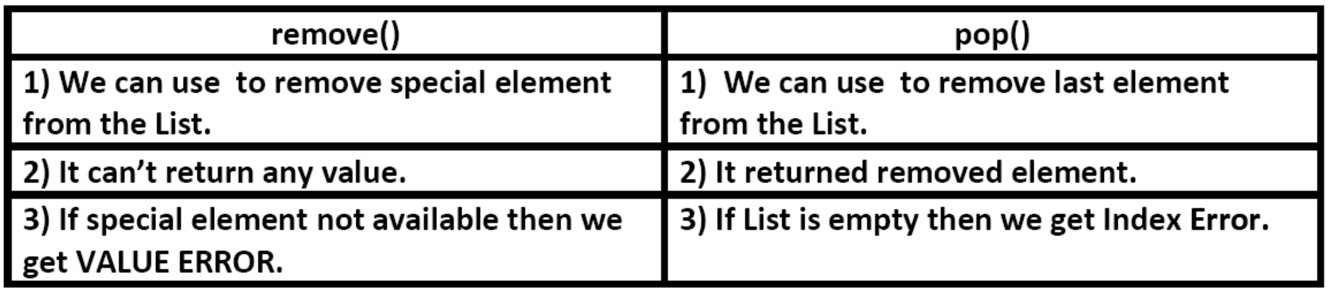
```
n=[10,20,30,40,50,60]
print(n.pop())          #60
print(n.pop(1))         #20
print(n.pop(10))        # IndexError: pop index out of range
```

```
60
20

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-11-c0a703a9cc2f> in <module>
      2 print(n.pop())          #60
      3 print(n.pop(1))           #20
----> 4 print(n.pop(10))        # IndexError: pop index out of range

IndexError: pop index out of range
```

Differences between remove() and pop()



remove()	pop()
1) We can use to remove special element from the List.	1) We can use to remove last element from the List.
2) It can't return any value.	2) It returned removed element.
3) If special element not available then we get VALUE ERROR.	3) If List is empty then we get Index Error.

Note:

List objects are dynamic. i.e based on our requirement we can increase and decrease the size.

- append(),insert() ,extend() ===>for increasing the size/growable nature
- remove(), pop() ======>for decreasing the size /shrinking nature

III. Ordering elements of List:

1. reverse():

- We can use to reverse() order of elements of list.

In [12]:

```
n=[10,20,30,40]
n.reverse()
print(n)
```

```
[40, 30, 20, 10]
```

2. sort() function:

- In list by default insertion order is preserved. If you want to sort the elements of list according to default natural sorting order then we should go for sort() method.

```
    -For numbers ==> default natural sorting order is Ascending Order


    -For Strings ==> default natural sorting order is Alphabetical Order
```

Eg 1:

In [13]:

```
n=[20,5,15,10,0]
n.sort()
print(n)
```

```
[0, 5, 10, 15, 20]
```

Eg 2:

In [14]:

```
s=["Dog","Banana","Cat","Apple"]
s.sort()
print(s)
```

```
['Apple', 'Banana', 'Cat', 'Dog']
```

In [17]:

```
s=["Dog","Banana","Cat","apple"]
s.sort()                                 # Unicode values are used during comparison of alp
print(s)
```

```
['Banana', 'Cat', 'Dog', 'apple']
```

Note:

- To use sort() function, **compulsory list should contain only homogeneous elements**, otherwise we will get **TypeError.**

Eg 3:

In [15]:

```
n=[20,10,"A","B"]
n.sort()
print(n)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-41c38805e086> in <module>
      1 n=[20,10,"A","B"]
----> 2 n.sort()
      3 print(n)

TypeError: '<' not supported between instances of 'str' and 'int'
```

Note:

- In Python 2 if List contains both numbers and Strings then sort() function first sort numbers followed by strings.

In [16]:

```
n=[20,"B",10,"A"]
n.sort()
print(n)             # [10,20,'A','B']   It is valid in Python 2, but in Python 3 it is inv
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-bda452934197> in <module>
      1 n=[20,"B",10,"A"]
----> 2 n.sort()
      3 print(n)             # [10,20,'A','B']   But in Python 3 it is inval
id.

TypeError: '<' not supported between instances of 'str' and 'int'
```

Eg 4:

In [19]:

```
n=['20',"B",'10',"A"]
n.sort()
print(n)
```

```
['10', '20', 'A', 'B']
```

How to sort the elements of list in reverse of default natural sorting order:

One Simple Way

In [25]:

```
n=[40,10,30,20]
n.sort()
n.reverse()
print(n)
```

```
[40, 30, 20, 10]
```

Alternate Way :

- We can sort according to reverse of default natural sorting order by using **reverse = True** argument.

In [27]:

```
n=[40,10,30,20]
n.sort()
print(n)            #[10,20,30,40]
n.sort(reverse=True)
print(n)            #[40,30,20,10]
n.sort(reverse=False)
print(n)           #[10,20,30,40]
```

```
[10, 20, 30, 40]
[40, 30, 20, 10]
[10, 20, 30, 40]
```

In [28]:

```
s=["Dog","Banana","Cat","Apple"]
s.sort(reverse= True)                      # reverse of Alphabetical order
print(s)
```

```
['Dog', 'Cat', 'Banana', 'Apple']
```

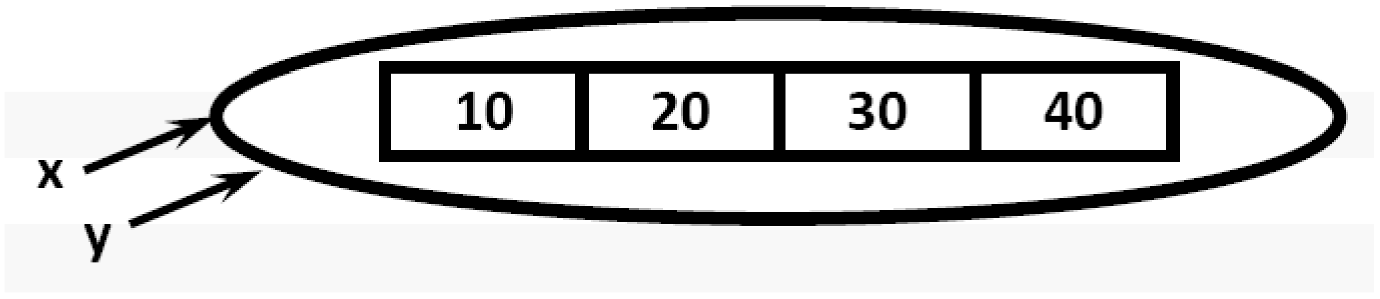
6 . Aliasing and Cloning of List objects:

- The process of giving another reference variable to the existing list is called aliasing.

In [29]:

```
x=[10,20,30,40]
y=x
print(id(x))
print(id(y))
```

```
1709842944648
1709842944648
```

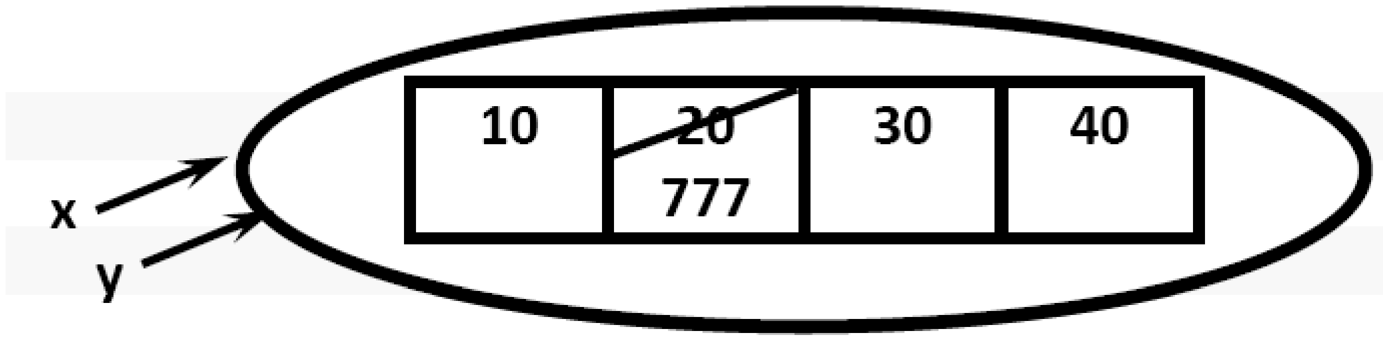


The problem in this approach is by using one reference variable if we are changing content,then those changes will be reflected to the other reference variable.

In [30]:

```
x=[10,20,30,40]
y=x
y[1]=777
print(x)              #[10,777,30,40]
```

```
[10, 777, 30, 40]
```



To overcome this problem we should go for **cloning**.

Cloning : The process of creating exactly duplicate independent object is called cloning.

We can implement cloning by using the following ways:

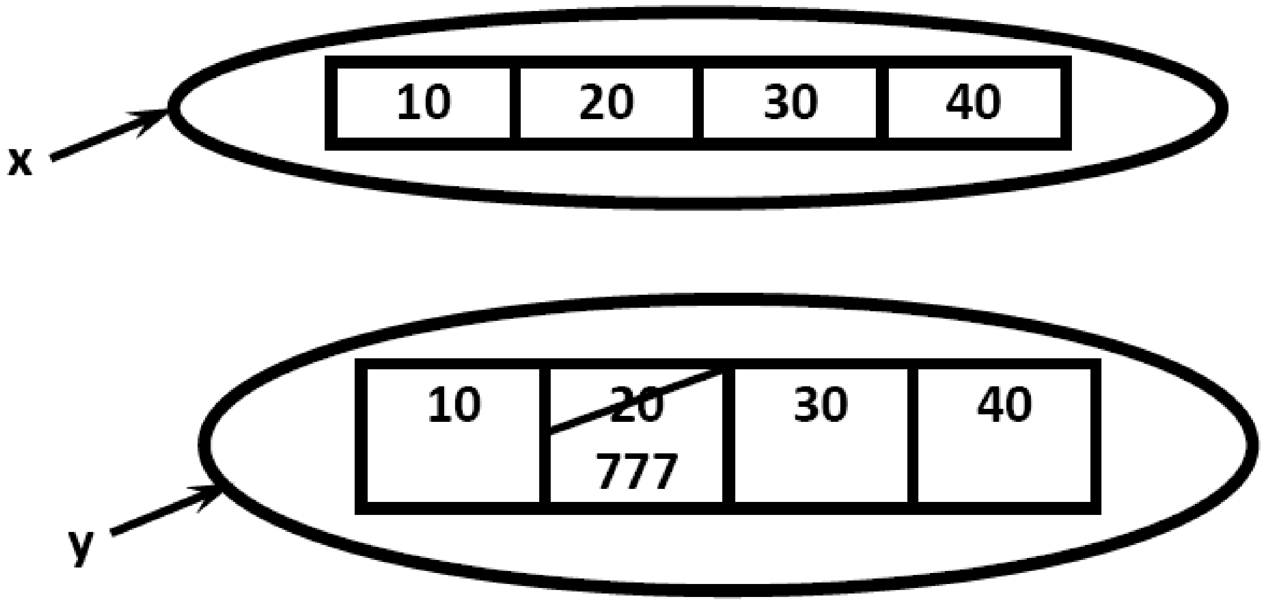
1. slice operator
2. copy() function

1. By using slice operator:

In [31]:

```
x=[10,20,30,40]
y=x[:]
y[1]=777
print(x) #[10,20,30,40]
print(y) #[10,777,30,40]
```

```
[10, 20, 30, 40]
[10, 777, 30, 40]
```

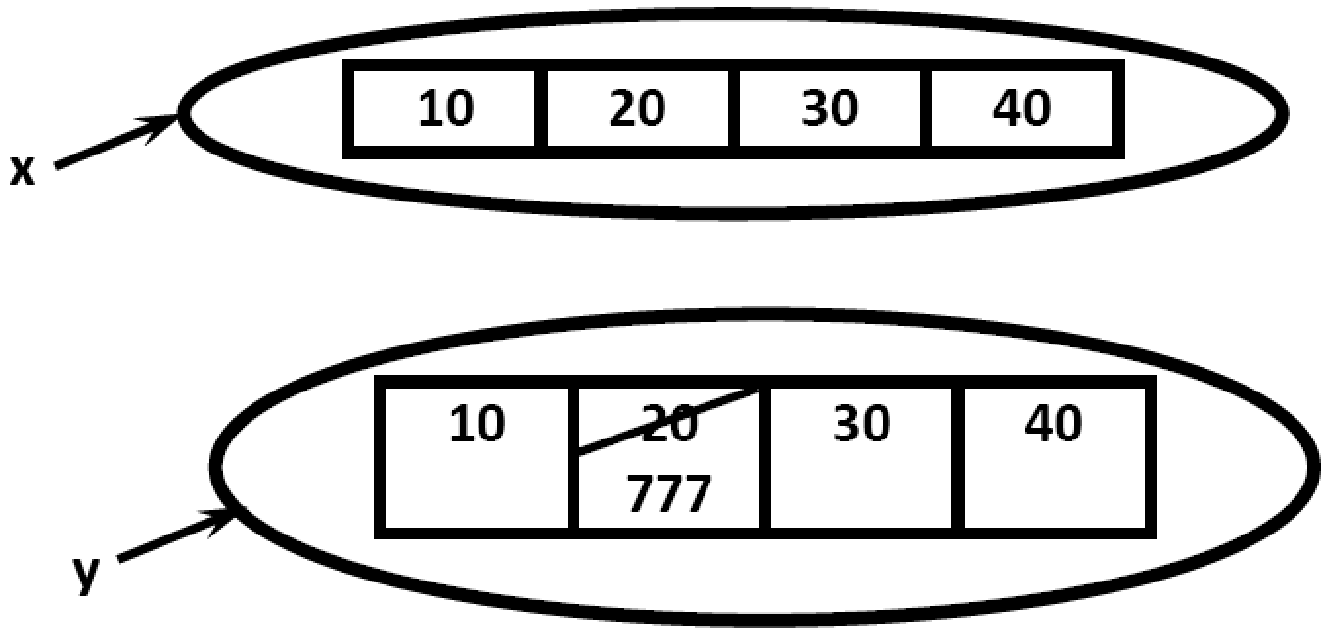


2. By using copy() function:

In [32]:

```
x=[10,20,30,40]
y=x.copy()
y[1]=777
print(x) # [10,20,30,40]
print(y) # [10,777,30,40]
```

```
[10, 20, 30, 40]
[10, 777, 30, 40]
```



Q. What is the difference between = operator and copy() function?

- = operator meant for aliasing copy() function meant for cloning

7. Using Mathematical operators for List Objects:

- We can use + and * operators for List objects.

1. Concatenation operator(+):

- We can use + to concatenate 2 lists into a single list.

In [38]:

```
a=[10,20,30]
b=[40,50,60]
c=a+b                        # concatenation
print(' a : ',a)
print(' b : ',b)
print(' c : ',c)
```

```
 a :  [10, 20, 30]
 b :  [40, 50, 60]
 c :  [10, 20, 30, 40, 50, 60]
```

In [41]:

```
a=[10,20,30]
b=[40,50,60]
c=a.extend(b)   # extend() methodwon't return anything. it adds the content of 'b' to 'a'.
print(' a : ',a)
print(' b : ',b)
print(' c : ',c)
```

```
 a :  [10, 20, 30, 40, 50, 60]
 b :  [40, 50, 60]
 c :  None
```

In [ ]:

```

```

Note:

- To use + operator compulsory both arguments should be list objects,otherwise we will get TypeError.

Eg:

In [42]:

```
c=a+40        #TypeError: can only concatenate list (not "int") to list
print(c)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-42-aae8ea9a4fe9> in <module>
----> 1 c=a+40        #TypeError: can only concatenate list (not "int") to l
ist
      2 print(c)

TypeError: can only concatenate list (not "int") to list
```

In [43]:

```
c=a+[40]      #valid
print(c)
```

```
[10, 20, 30, 40, 50, 60, 40]
```

2. Repetition Operator(*):

- We can use repetition operator * to repeat elements of list specified number of times

In [44]:

```
x=[10,20,30]
y=x*3
print(y)         #[10,20,30,10,20,30,10,20,30]
```

```
[10, 20, 30, 10, 20, 30, 10, 20, 30]
```

8. Comparing List objects

- We can use comparison operators for List objects.

Eg :

In [45]:

```
x=["Dog","Cat","Rat"]
y=["Dog","Cat","Rat"]
z=["DOG","CAT","RAT"]
print(x==y)                      #True
print(x==z)                     #False
print(x != z)                    #True
```

```
True
False
True
```

Note:

- Whenever we are using comparison operators(==,!=) for List objects then the following should be considered:

  ```
  1. The number of elements
  
  
  2. The order of elements
  
  
  3. The content of elements (case sensitive)
  ```

Note:

- When ever we are using relatational operators(<,<=,>,>=) between List objects,only first element comparison will be performed.

Eg :

In [46]:

```
x=[50,20,30]
y=[40,50,60,100,200]
print(x>y)         #True
print(x>=y)        #True
print(x<y)         #False
print(x<=y)        #False
```

```
True
True
False
False
```

Eg :

In [47]:

```
x=["Dog","Cat","Rat"]
y=["Rat","Cat","Dog"]
print(x>y)      #False
print(x>=y)     #False
print(x<y)      #True
print(x<=y)     #True
```

```
False
False
True
True
```

In [48]:

```
x=["Dog","Cat","Rat"]
y=["Dat","Cat","Dog"]
print(x>y)
```

```
True
```

9. Membership operators:

We can check whether element is a member of the list or not by using memebership operators.

1. in operator

2. not in operator

In [49]:

```
n=[10,20,30,40]
print (10 in n)
print (10 not in n)
print (100 not in n)
print (50 in n)
print (50 not in n)
```

```
True
False
True
False
True
```

10. clear() function:

- We can use clear() function to remove all elements of List.

In [50]:

```
n=[10,20,30,40]
print(n)
n.clear()
print(n)
```

```
[10, 20, 30, 40]
[]
```

Date: 04-05-2020 Day 4

11. Nested Lists:

- Sometimes we can take one list inside another list. Such type of lists are called nested lists.
- Consider the follwoing example:

In [1]:

```
n=[10,20,[30,40]]
print(n)
print(n[0])
print(n[2])
print(n[2][0])
print(n[2][1])
```

```
[10, 20, [30, 40]]
10
[30, 40]
30
40
```

Note:

- We can access nested list elements by using index just like accessing multi dimensional array elements.

Nested List as Matrix

- In Python we can represent matrix by using nested lists.

In [2]:

```
n=[[10,20,30],[40,50,60],[70,80,90]]
print(n)
print("Elements by Row wise:")
for r in n:
    print(r)
print("Elements by Matrix style:")
for i in range(len(n)):
    for j in range(len(n[i])):
        print(n[i][j],end=' ')
    print()
```

```
[[10, 20, 30], [40, 50, 60], [70, 80, 90]]
Elements by Row wise:
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]
Elements by Matrix style:
10 20 30
40 50 60
70 80 90
```

12. List Comprehensions

- It is very easy and compact way of creating list objects from any iterable objects(like list,tuple,dictionary,range etc) based on some condition.

Syntax:

list=[expression for item in list if condition]

Consider an example, If you want to store squares of numbers form 1 to 10 in a list,

In [12]:

```
l1=[]
for x in range(1,11):
    l1.append(x*x)
print(l1)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In the above case, the program consisting 4 lines of code. Now for the same purpose we will write the following code in more concised way.

In [13]:

```
l1 = [x*x for x in range(1,21)]
l2 = [x for x in l1 if x % 2 == 0]
print(l1)
print(l2)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 32
4, 361, 400]
[4, 16, 36, 64, 100, 144, 196, 256, 324, 400]
```

Few more examples on List comprehensions :

In [10]:

```
l1 = [x*x for x in range(1,11)]
print(l1)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [14]:

```
l =[2**x for x in range(1,11)]
print(l)
```

```
[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

In [8]:

```
l = [x for x in range(1,11) if x%2==0]
print(l)
```

```
[2, 4, 6, 8, 10]
```

In [15]:

```
l = [x for x in range(1,11) if x%2==1]
print(l)
```

```
[1, 3, 5, 7, 9]
```

In [17]:

```
l = [x**2 for x in range(1,11) if (x**2)%2==1]
print(l)
```

```
[1, 9, 25, 49, 81]
```

In [16]:

```
words=["Balaiah","Nag","Venkatesh","Chiranjeevi"]
l=[w[0] for w in words]
print(l)
```

```
['B', 'N', 'V', 'C']
```

In [21]:

```
words=["Balaiah","Nag","Venkatesh","Chiranjeevi"]
l=[w for w in words if len(w)>6]
print(l)
```

```
['Balaiah', 'Venkatesh', 'Chiranjeevi']
```

In [18]:

```
num1=[10,20,30,40]
num2=[30,40,50,60]
num3=[ i for i in num1 if i not in num2]
print(num3)                               #[10,20]
```

```
[10, 20]
```

In [22]:

```
words="the quick brown fox jumps over the lazy dog".split()    # All 26 alphabets used in t
print(words)
l=[[w.upper(),len(w)] for w in words]
print(l)
```

```
['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
[['THE', 3], ['QUICK', 5], ['BROWN', 5], ['FOX', 3], ['JUMPS', 5], ['OVER',
4], ['THE', 3], ['LAZY', 4], ['DOG', 3]]
```

Example Program

Q. Write a program to display unique vowels present in the given word.

In [23]:

```
vowels=['a','e','i','o','u']
word=input("Enter the word to search for vowels: ")
found=[]
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
print(found)
print("The number of different vowels present in",word,"is",len(found))
```

```
Enter the word to search for vowels: Quality Education is useful
['u', 'a', 'i', 'o', 'e']
The number of different vowels present in Quality Education is useful is 5
```

Suppose if you want lower case and upper case vowels, what you can do is as follows:

In [2]:

```
vowels=['a','e','i','o','u','A','E','I','O','U']
word=input("Enter the word to search for vowels: ")
found=[]
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
print(found)
print("The number of different vowels present in",word,"is",len(found))
```

```
Enter the word to search for vowels: karthi Abc
['a', 'i', 'A']
The number of different vowels present in karthi Abc is 3
```

See the above code in another simplified way:

In [3]:

```
vowels=['a','e','i','o','u']
word=input("Enter the word to search for vowels: ")
found=[]
for letter in word:
    if letter.lower() in vowels:
        if letter.lower() not in found:
            found.append(letter)
print(found)
print("The number of different vowels present in",word,"is",len(found))
```

```
Enter the word to search for vowels: KARTHIKEYA
['A', 'I', 'E', 'A']
The number of different vowels present in KARTHIKEYA is 4
```

In [4]:

```
vowels=['a','e','i','o','u']
word=input("Enter the word to search for vowels: ")
found=[]
for letter in word:
    if letter.lower() in vowels:
        if letter.lower() not in found:
            found.append(letter.lower())
print(found)
print("The number of different vowels present in",word,"is",len(found))
```

```
Enter the word to search for vowels: KARTHIKEYA
['a', 'i', 'e']
The number of different vowels present in KARTHIKEYA is 3
```

In [ ]:

```

```

Tuple Data Type

Date: 04-05-2020 Day 1

Topics Covered:

1. Introduction

2. Creation of Tuple objects

3. Accessing elements of tuple

4. Tuple vs immutability

5. Mathematical operators for tuple

6. Important functions of Tuple

```
  i) len()

 ii) count()

iii) index()

 iv) sorted()

  v) min()

  vi) max()

  vii) cmp()
```

7. Tuple Packing and Unpacking

8. Tuple Comprehension

9. Differences between List and Tuple

1. Introduction:

1. **Tuple is exactly same as List except that it is immutable.** i.e., once we creates Tuple object,we cannot perform any changes in that object. Hence **Tuple is Read Only Version of List.**

2. If our data is fixed and never changes then we should go for Tuple.

3. Insertion Order is preserved.

4. Duplicates are allowed.
5. Heterogeneous objects are allowed.
6. We can preserve insertion order and we can differentiate duplicate objects by using index. Hence index will play very important role in Tuple also.
7. Tuple support both +ve and -ve index. +ve index means forward direction(from left to right) and -ve index means backward direction(from right to left).
8. We can represent Tuple elements within Parenthesis and with comma seperator.

Note :

- Parenethesis are optional but recommended to use.

Eg :

In [1]:

```
t=10,20,30,40
print(t)
print(type(t))
```

```
(10, 20, 30, 40)
<class 'tuple'>
```

In [2]:

```
t=(10,20,30,40)
print(t)
print(type(t))
```

```
(10, 20, 30, 40)
<class 'tuple'>
```

In [3]:

```
t = ()
print(type(t))
```

```
<class 'tuple'>
```

Note:

- We have to take special care about single valued tuple.compulsary the value should ends with comma,otherwise it is not treated as tuple.

Eg:

In [4]:

```
t=(10)
print(t)
print(type(t))
```

```
10
<class 'int'>
```

In [5]:

```
t=(10,)
print(t)
print(type(t))
```

```
(10,)
<class 'tuple'>
```

Q. Which of the following are valid/Invalid tuples?

In [7]:

```
t=()                            # valid
t=10,20,30,40                   # valid
t=10                            # not valid
t=10,                           # valid
t=(10)                          # notvalid
t=(10,)                         # valid
t=(10,20,30,40)                 # valid
t= (10,20,30,)                  # valid
```

In [1]:

```
t = (10,20,30,)
print(t)
print(type(t))
```

```
(10, 20, 30)
<class 'tuple'>
```

Date: 05-05-2020 Day 2

2. Creation of Tuple objects

1. t=()

- creation of empty tuple

In [2]:

```
t=()
print(t)
print(type(t))
```

```
()
<class 'tuple'>
```

2. t=(10,)

t=10, ====> creation of single valued tuple ,parenthesis are optional,but it should ends with comma.

In [3]:

```
t = (10,)
print(t)
print(type(t))
```

```
(10,)
<class 'tuple'>
```

3. t=10,20,30 or **t=(10,20,30)***

- creation of multi values tuples & parenthesis are optional.

In [4]:

```
t = 10,20,30
print(t)
print(type(t))
```

```
(10, 20, 30)
<class 'tuple'>
```

4. By using tuple() function:

- if you have any sequence (i.e., string, list, range etc.,) which can be easily converted into a tuple by using **tuple()** function.

In [5]:

```
list=[10,20,30]
t=tuple(list)
print(t)
print(type(t))
```

```
(10, 20, 30)
<class 'tuple'>
```

In [9]:

```
t=tuple(range(10,20,2))
print(t)
print(type(t))
```

```
(10, 12, 14, 16, 18)
<class 'tuple'>
```

In [10]:

```
t = tuple('karthi')
print(t)
print(type(t))
```

```
('k', 'a', 'r', 't', 'h', 'i')
<class 'tuple'>
```

3. Accessing elements of tuple:

We can access elements of a tuple either by using index or by using slice operator.

1. By using index:

In [7]:

```
t=(10,20,30,40,50,60)
print(t[0])          #10
print(t[-1])         #60
print(t[100])        #IndexError: tuple index out of range
```

```
10
60

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-7-a762027e6505> in <module>
      2 print(t[0])          #10
      3 print(t[-1])         #60
----> 4 print(t[100])        #IndexError: tuple index out of range

IndexError: tuple index out of range
```

2. By using slice operator:

In [8]:

```
t=(10,20,30,40,50,60)
print(t[2:5])             #30,40,50
print(t[2:100])           # 30,40,50,60
print(t[::2])             #10,30,50
```

```
(30, 40, 50)
(30, 40, 50, 60)
(10, 30, 50)
```

Eg:

In [11]:

```
t= tuple('karthikeya')
print(t[0])
print(t[1:5:1])
print(t[-2:-5:-1])
```

```
k
('a', 'r', 't', 'h')
('y', 'e', 'k')
```

4. Tuple vs immutability:

- Once we creates tuple,we cannot change its content. Hence tuple objects are immutable.

Eg :

In [12]:

```
t=(10,20,30,40)
t[1]=70
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-b9dcfa3c846d> in <module>
      1 t=(10,20,30,40)
----> 2 t[1]=70

TypeError: 'tuple' object does not support item assignment
```

5. Mathematical operators for tuple:

- We can apply + and * operators for tuple

1. Concatenation Operator(+):

In [13]:

```
t1=(10,20,30)
t2=(40,50,60)
t3=t1+t2
print(t3)         # (10,20,30,40,50,60)
```

```
(10, 20, 30, 40, 50, 60)
```

In [15]:

```
t1 = 10,20,30,40
t2 = 10,20,30,40
t3 = t1 + t2    # because list and tuple allow duplicates, so you will get all the elements
print(t3)
```

```
(10, 20, 30, 40, 10, 20, 30, 40)
```

2. Multiplication operator (or) repetition operator(*):

In [14]:

```
t1=(10,20,30)
t2=t1*3
print(t2) #(10,20,30,10,20,30,10,20,30)
```

```
(10, 20, 30, 10, 20, 30, 10, 20, 30)
```

6. Important functions of Tuple:

1. len():

- It is an in-built function of Python, if you provide any sequnce (i.e., strings, list,tuple etc.,), in that how many elements are there that will be retuRened this function.

- It is used to return number of elements present in the tuple.

In [16]:

```
t=(10,20,30,40)
print(len(t)) #4
```

```
4
```

2. count():

- To return number of occurrences of given element in the tuple

In [17]:

```
t=(10,20,10,10,20)
print(t.count(10)) #3
```

```
3
```

3. index():

- It returns index of first occurrence of the given element. If the specified element is not available then we will get ValueError.

In [18]:

```
t=(10,20,10,10,20)
print(t.index(10))    # 0
print(t.index(30))    # ValueError: tuple.index(x): x not in tuple
```

```
0

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-e5a94eb1d82a> in <module>
      1 t=(10,20,10,10,20)
      2 print(t.index(10))    # 0
----> 3 print(t.index(30))    # ValueError: tuple.index(x): x not in tuple

ValueError: tuple.index(x): x not in tuple
```

4. sorted():

- It is used to sort elements based on default natural sorting order (Ascending order).

In [30]:

```
t =(10,30,40,20)
print(sorted(t))      # sorted() is going to return list
```

```
[10, 20, 30, 40]
```

In [31]:

```
t =(10,30,40,20)
t.sort()
print(t)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-31-6dd56d99cf24> in <module>
      1 t =(10,30,40,20)
----> 2 t.sort()
      3 print(t)

AttributeError: 'tuple' object has no attribute 'sort'
```

Reason: Tuple is Immutable. We cannot modify the tuple contents.

In [34]:

```
t=(40,10,30,20)
print(id(t))
print(type(t))
t=sorted(t)
print(id(t))
print(type(t))
print(t)      # result is in List form.
```

```
2653757219768
<class 'tuple'>
2653757029192
<class 'list'>
[10, 20, 30, 40]
```

In [36]:

```
t=(40,10,30,20)
t1=sorted(t)
print(type(t1))
print(t1)
print(type(t))
print(t)
```

```
<class 'list'>
[10, 20, 30, 40]
<class 'tuple'>
(40, 10, 30, 20)
```

In [37]:

```
t=(40,10,30,20)
t1=tuple(sorted(t))
print(type(t1))
print(t1)
print(type(t1))
print(t)
```

```
<class 'tuple'>
(10, 20, 30, 40)
<class 'tuple'>
(40, 10, 30, 20)
```

We can sort according to reverse of default natural sorting order is as follows:

In [20]:

```
t1=sorted(t,reverse=True)
print(t1)                    #[40, 30, 20, 10]
```

```
[40, 30, 20, 10]
```

5. min() and max() functions:

- These functions return minimum and maximum values according to default natural sorting order.
- These functions will works on tuple with respect to homogeneous elements only.

In [35]:

```
t=(40,10,30,20)
print(min(t)) #10
print(max(t)) #40
```

```
10
40
```

In [40]:

```
t = ('karthi')       # based on unicode values these functions will work.
print(min(t))
print(max(t))
```

```
a
t
```

In [39]:

```
t = ('kArthi')
print(min(t))
print(max(t))
```

```
A
t
```

6. cmp():

- It compares the elements of both tuples.
- If both tuples are equal then returns 0.
- If the first tuple is less than second tuple then it returns -1.
- If the first tuple is greater than second tuple then it returns +1.

In [41]:

```
t1=(10,20,30)
t2=(40,50,60)
t3=(10,20,30)
print(cmp(t1,t2)) # -1
print(cmp(t1,t3)) # 0
print(cmp(t2,t3)) # +1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-41-558f5c41fd64> in <module>
      2 t2=(40,50,60)
      3 t3=(10,20,30)
----> 4 print(cmp(t1,t2)) # -1
      5 print(cmp(t1,t3)) # 0
      6 print(cmp(t2,t3)) # +1

NameError: name 'cmp' is not defined
```

Note : cmp() function is available only in Python 2 but not in Python 3

In [46]:

```
t1=(10,20,30)
t2=(40,50,60)
t3=(10,20,30)
print(t1==t2)
print(t1==t3)
print(t2==t3)
print(t1<t2)   # true, because it compares only first element.
```

```
False
True
False
True
```

In [47]:

```
t1=(10,20,30)
t2=(5,50,60)
print(t1<t2)
```

```
False
```

7. Tuple Packing and Unpacking:

Tuple packing :

- We can create a tuple by packing a group of variables.

Eg:

In [48]:

```
a=10
b=20
c=30
d=40
t=a,b,c,d
print(t) #(10, 20, 30, 40)
```

```
(10, 20, 30, 40)
```

Here a,b,c,d are packed into a tuple t. This is nothing but **tuple packing.**

Tuple unpacking :

- Tuple unpacking is the reverse process of tuple packing.
- We can unpack a tuple and assign its values to different variables.

Eg :

In [49]:

```
t=(10,20,30,40)
a,b,c,d=t
print("a=",a,"b=",b,"c=",c,"d=",d)
```

```
a= 10 b= 20 c= 30 d= 40
```

Note : This concept is also applicable for any sequence (i.e., string,list,set etc.,) concept also.

Unpacking:

In [1]:

```
t=[10,20,30,40]
a,b,c,d=t
print("a=",a,"b=",b,"c=",c,"d=",d)
```

```
a= 10 b= 20 c= 30 d= 40
```

In [3]:

```
t={10,20,30,40}
a,b,c,d=t
print("a=",a,"b=",b,"c=",c,"d=",d)
```

```
a= 40 b= 10 c= 20 d= 30
```

In [2]:

```
t='abcd'
a,b,c,d=t
print("a=",a,"b=",b,"c=",c,"d=",d)
```

```
a= a b= b c= c d= d
```

Packing:

In [4]:

```
a = 10
b = 20
c = 30
d = 40
t =[a,b,c,d]
print(type(t))
print(t)
```

```
<class 'list'>
[10, 20, 30, 40]
```

In [7]:

```
a = 10
b = 20
c = 30
d = 40
t ={a,b,c,d}    # for 'set' order is not important
print(type(t))
print(t)
```

```
<class 'set'>
{40, 10, 20, 30}
```

In [6]:

```
a = 10
b = 20
c = 30
d = 40
t ='a,b,c,d'
print(type(t))
print(t)
```

```
<class 'str'>
a,b,c,d
```

Note:

- At the time of tuple unpacking the number of variables and number of values should be same. ,otherwise we will get ValueError.

Eg :

In [50]:

```
t=(10,20,30,40)
a,b,c=t                     # ValueError: too many values to unpack (expected 3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-50-11ffc4f6133a> in <module>
      1 t=(10,20,30,40)
----> 2 a,b,c=t                    #ValueError: too many values to unpack
 (expected 3)

ValueError: too many values to unpack (expected 3)
```

8. Tuple Comprehension

Tuple Comprehension is not supported by Python.

t= (x2 for x in range(1,6))**

Here we are not getting tuple object and we are getting **generator** object.

In [9]:

```
t= ( x**2 for x in range(1,6))
print(type(t))
for x in t:
    print(x)
```

```
<class 'generator'>
1
4
9
16
25
```

Eg :

Q. Write a program to take a tuple of numbers from the keyboard and print its sum and average.

In [11]:

```
t=eval(input("Enter Tuple of Numbers:"))
print(type(t))
l=len(t)
sum=0
for x in t:
    sum = sum + x
print("The Sum=",sum)
print("The Average=",sum/l)
```

```
Enter Tuple of Numbers:(10,20,30,40)
<class 'tuple'>
The Sum= 100
The Average= 25.0
```

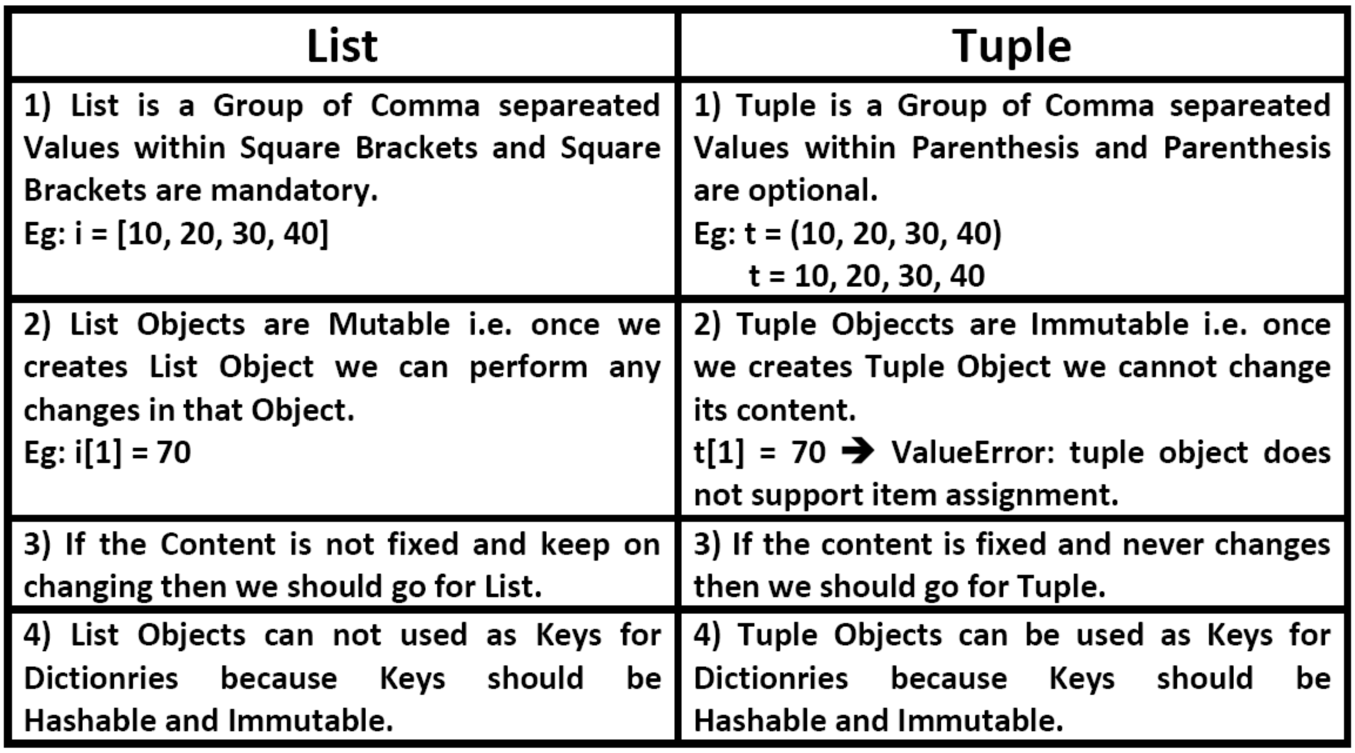
In [12]:

```
t=eval(input("Enter Tuple of Numbers:"))
print(type(t))
l=len(t)
sum=0
for x in t:
    sum = sum + x
print("The Sum=",sum)
print("The Average=",sum/l)
```

```
Enter Tuple of Numbers:100,200,220,300
<class 'tuple'>
The Sum= 820
The Average= 205.0
```

9. Differences between List and Tuple:

- List and Tuple are exactly same except small difference: List objects are mutable where as Tuple objects are immutable.
- In both cases insertion order is preserved, duplicate objects are allowed, heterogenous objects are allowed, index and slicing are supported.



List	Tuple
1) List is a Group of Comma separeated Values within Square Brackets and Square Brackets are mandatory. Eg: i = [10, 20, 30, 40]	1) Tuple is a Group of Comma separeated Values within Parenthesis and Parenthesis are optional. Eg: t = (10, 20, 30, 40) t = 10, 20, 30, 40
2) List Objects are Mutable i.e. once we creates List Object we can perform any changes in that Object. Eg: i[1] = 70	2) Tuple Objeccts are Immutable i.e. once we creates Tuple Object we cannot change its content. t[1] = 70 ➔ ValueError: tuple object does not support item assignment.
3) If the Content is not fixed and keep on changing then we should go for List.	3) If the content is fixed and never changes then we should go for Tuple.
4) List Objects can not used as Keys for Dictionries because Keys should be Hashable and Immutable.	4) Tuple Objects can be used as Keys for Dictionries because Keys should be Hashable and Immutable.

In [ ]:

```

```

UNIT - 5

Set Data Type

Date: 06-05-2020 Day 1

Topics Covered:

1. Introduction

2. Creation of Set Objects

3. Important functions / methods of set

```
1. add()


2. update()


3. copy()


4. pop()


5. remove()


6. discard()


7. clear()
```

4. Mathematical operations on the Set

5. Membership operators

```
i) in


ii) not in
```

6. Set Comprehension

1. Introduction

If we want to represent a group of unique values as a single entity then we should go for set.

Key features of Set Data Type:

1. Duplicates are not allowed.
2. Insertion order is not preserved.But we can sort the elements.
3. Indexing and slicing not allowed for the set.
4. Heterogeneous elements are allowed.
5. Set objects are mutable i.e once we creates set object we can perform any changes in that object based on our requirement.
6. We can represent set elements within curly braces and with comma seperation.
7. We can apply mathematical operations like union,intersection,difference etc on set objects.

2. Creation of Set Objects

i) Creation of set object with single value

In [5]:

```
s = {10}
print(type(s))
print(s)
```

```
<class 'set'>
{10}
```

ii) Creation of set object with multiple values

In [6]:

```
s = {30,40,10,5,20}   # in the output order not preserved
print(type(s))
print(s)
```

```
<class 'set'>
{5, 40, 10, 20, 30}
```

In [10]:

```
s = {30,40,10,5,20}   # in the output order not preserved
print(type(s))
print(s[0])
```

```
<class 'set'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-655c69b5c557> in <module>
      1 s = {30,40,10,5,20}   # in the output order not preserved
      2 print(type(s))
----> 3 print(s[0])

TypeError: 'set' object is not subscriptable
```

In [11]:

```
s = {30,40,10,5,20}   # in the output order not preserved
print(type(s))
print(s[0:6])
```

```
<class 'set'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-05c9c76958c2> in <module>
      1 s = {30,40,10,5,20}   # in the output order not preserved
      2 print(type(s))
----> 3 print(s[0:6])

TypeError: 'set' object is not subscriptable
```

iii) Creation of set objects using set() function

- We can create set objects by using **set()** function.

Syntax:

s=set(any sequence)

Eg 1:

In [7]:

```
l = [10,20,30,40,10,20,10]
s=set(l)
print(s)            # {40, 10, 20, 30} because duplicates are not allowed in set
```

```
{40, 10, 20, 30}
```

Eg 2:

In [9]:

```
s=set(range(5))
print(s) #{0, 1, 2, 3, 4}
```

```
{0, 1, 2, 3, 4}
```

Eg 3:

In [12]:

```
s = set('karthi')
print(s)
```

```
{'a', 'h', 'i', 't', 'k', 'r'}
```

Eg 4:

In [15]:

```
s= set('aaabbbb')
print(s)
```

```
{'a', 'b'}
```

Note:

- While creating empty set we have to take special care. Compulsory we should use set() function.
- s={} ==>It is treated as dictionary but not empty set.

Eg :

In [13]:

```
s = {}
print(type(s))
```

```
<class 'dict'>
```

Eg :

In [14]:

```
s = set()       # set function without any arguments
print(s)
print(type(s))
```

```
set()
<class 'set'>
```

3. Important functions / methods of set:

1. add(x):

- Adds item x to the set

Eg :

In [31]:

```
s={10,20,30}
s.add(40);                          # ';' is optional for python statements
print(s)    #{40, 10, 20, 30}
```

```
{40, 10, 20, 30}
```

In [32]:

```
s={10,20,30}
s.add('karthi');                          # ';' is optional for python statements
print(s)
```

```
{10, 'karthi', 20, 30}
```

2. update(x,y,z):

- This method is used to add multiple items to the set.
- Arguments are not individual elements and these are Iterable objects like List,range etc.
- All elements present in the given Iterable objects will be added to the set.

Eg :

In [33]:

```
s={10,20,30}
s.update('karthi');                          # ';' is optional for python statements
print(s)
```

```
{'a', 10, 'h', 'i', 20, 't', 'k', 'r', 30}
```

In [23]:

```
s={10,20,30}
l=[40,50,60,10]
s.update(l,range(5))
print(s)
```

```
{0, 1, 2, 3, 4, 40, 10, 50, 20, 60, 30}
```

In [24]:

```
s={10,20,30}
l=[40,50,60,10]
s.update(l,range(5),100)
print(s)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-d6e54bc11daa> in <module>
      1 s={10,20,30}
      2 l=[40,50,60,10]
----> 3 s.update(l,range(5),100)
      4 print(s)

TypeError: 'int' object is not iterable
```

In [25]:

```
s={10,20,30}
l=[40,50,60,10]
s.update(l,range(5),'100')
print(s)
```

```
{0, 1, 2, 3, 4, 40, 10, '0', '1', 50, 20, 60, 30}
```

In [26]:

```
s={10,20,30}
l=[40,50,60,10]
s.update(l,range(5),'karthi')
print(s)
```

```
{0, 1, 2, 3, 4, 'a', 40, 10, 'h', 'i', 50, 20, 't', 'k', 'r', 60, 30}
```

In [18]:

```
s =set()
s.update(range(1,10,2),range(0,10,2))
print(s)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Q. What is the difference between add() and update() functions in set?

- We can use add() to add individual item to the Set,where as we can use update() function to add multiple items to Set.
- add() function can take only one argument where as update() function can take any number of arguments but all arguments should be iterable objects.

Q. Which of the following are valid for set s?

1. s.add(10) **Valid**
2. s.add(10,20,30) **TypeError: add() takes exactly one argument (3 given)**

3. s.update(10) **TypeError: 'int' object is not iterable**

4. s.update(range(1,10,2),range(0,10,2)) **Valid**

3. copy():

- Returns copy of the set. It is cloned object (Backup copy).

Eg :

In [20]:

```
s={10,20,30}
s1=s.copy()
print(s1)
print(s)
```

```
{10, 20, 30}
{10, 20, 30}
```

4. pop():

- It removes and returns some random element from the set.

Eg :

In [38]:

```
s={40,10,30,20}
print(s)
print(s.pop())
print(s.pop())
print(s.pop())
print(s)
print(s.pop())
print(s)    # empty set
print(s.pop())
```

```
{40, 10, 20, 30}
40
10
20
{30}
30
set()

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-38-22f4166ffe90> in <module>
      7 print(s.pop())
      8 print(s)    # empty set
----> 9 print(s.pop())

KeyError: 'pop from an empty set'
```

Consider the following case :

In [39]:

```
s={40,10,30,20}
print(s)
print(s.pop())
print(s.pop())
print(s)
```

```
{40, 10, 20, 30}
40
10
{20, 30}
```

In [40]:

```
s={40,10,30,20}
print(s)
print(s.pop())
print(s.pop())
print(s)
```

```
{40, 10, 20, 30}
40
10
{20, 30}
```

In [41]:

```
s={40,10,30,20}
print(s)
print(s.pop())
print(s.pop())
print(s)
```

```
{40, 10, 20, 30}
40
10
{20, 30}
```

Note :

How many times you may execute the code, the elements which are popped from the set in same order. The reason is ---

- All the elements of set are inserted based on some hashcode.If that order is fixed then it is always going to return one by one. But in which order these elements are inserted we don't know.

5. remove(x):

- It removes specified element from the set.
- If the specified element not present in the Set then we will get **KeyError**.

Eg :

In [30]:

```
s={40,10,30,20}
s.remove(30)
print(s)         # {40, 10, 20}
s.remove(50)     # KeyError: 50
```

```
{40, 10, 20}

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-30-fd29f2336f3b> in <module>
      2 s.remove(30)
      3 print(s)         # {40, 10, 20}
----> 4 s.remove(50)     #KeyError: 50

KeyError: 50
```

Date: 07-05-2020 Day 2

6. discard(x):

- It removes the specified element from the set. If the specified element not present in the set then we won't get any error.

In [1]:

```
s={10,20,30}
s.discard(10)
print(s)             #{20, 30}
s.discard(50)
print(s)            #{20, 30}
```

```
{20, 30}
{20, 30}
```

Answer the following :

Q. What is the difference between remove() and discard() functions in Set?

Q. Explain differences between pop(),remove() and discard() functionsin Set?

7.clear():

- To remove all elements from the Set.

In [2]:

```
s={10,20,30}
print(s)
s.clear()
print(s)
```

```
{10, 20, 30}
set()
```

4. Mathematical operations on the Set

1.union():

x.union(y) ==> We can use this function to return all elements present in both x and y sets

wecan perform union operation in two ways:

1. x.union(y) ==> by calling through union() method.

2. x|y ==> by using '|' operator.

This operation returns all elements present in both sets x and y (without duplicate elements).

Eg :

In [3]:

```
x={10,20,30,40}
y={30,40,50,60}
print(x.union(y)) #{10, 20, 30, 40, 50, 60}
print(x|y) #{10, 20, 30, 40, 50, 60}
```

```
{40, 10, 50, 20, 60, 30}
{40, 10, 50, 20, 60, 30}
```

2. intersection():

wecan perform intersection operation in two ways:

1. x.intersection(y) --> by calling through intersection() method.

2. x&y --> by using '&' operator.

This operation returns common elements present in both sets x and y.

Eg :

In [4]:

```
x={10,20,30,40}
y={30,40,50,60}
print(x.intersection(y)) #{40, 30}
print(x&y) #{40, 30}
```

```
{40, 30}
{40, 30}
```

3. difference():

wecan perform difference operation in two ways:

1. x.difference(y) --> by calling through difference() method.

2. x-y --> by using '-' operator.

This operation returns the elements present in x but not in y.

Eg :

In [5]:

```
x={10,20,30,40}
y={30,40,50,60}
print(x.difference(y)) #{10, 20}
print(x-y) #{10, 20}
print(y-x) #{50, 60}
```

```
{10, 20}
{10, 20}
{50, 60}
```

4.symmetric_difference():

wecan perform symmetric_difference operation in two ways:

1. x.symmetric_difference(y) --> by calling through symmetric_difference method.

2. x^y --> by using '^' operator.

This operation returns elements present in either x or y but not in both.

Eg :

In [6]:

```
x={10,20,30,40}
y={30,40,50,60}
print(x.symmetric_difference(y)) #{10, 50, 20, 60}
print(x^y) #{10, 50, 20, 60}
```

```
{10, 50, 20, 60}
{10, 50, 20, 60}
```

5. Membership operators:

Membership operators are used to check whether a particular object is available or not.

For any sequence, we can apply membership operators.

Follwoing are the membership operators:

1. in

2. not in

Eg :

In [8]:

```
s=set("karthi")
print(s)
print('a' in s)
print('z' in s)
```

```
{'r', 'a', 'i', 't', 'k', 'h'}
True
False
```

6. Set Comprehension

Set comprehension is possible.

Syntax:

s = {expression for x in sequence condition}

In [9]:

```
s = {x*x    for x in range(6)}
print(s)
```

```
{0, 1, 4, 9, 16, 25}
```

In [10]:

```
s={2**x for x in range(2,10,2)}
print(s)
```

```
{16, 256, 64, 4}
```

Note :

set objects won't support indexing and slicing.

Eg:

In [11]:

```
s={10,20,30,40}
print(s[0])           #TypeError: 'set' object does not support indexing
print(s[1:3])         #TypeError: 'set' object is not subscriptable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-6f6a7552f39c> in <module>
      1 s={10,20,30,40}
----> 2 print(s[0])           #TypeError: 'set' object does not support in
dexing
      3 print(s[1:3])         #TypeError: 'set' object is not subscriptabl
e

TypeError: 'set' object is not subscriptable
```

Example Programs

Q 1. Write a program to eliminate duplicates present in the list.

Approach-1:

In [12]:

```
l=eval(input("Enter List of values: "))
s=set(l)
print(s)
```

```
Enter List of values: 10,20,30,10,20,40
{40, 10, 20, 30}
```

Approach-2:

In [13]:

```
l=eval(input("Enter List of values: "))
l1=[]
for x in l:
    if x not in l1:
        l1.append(x)
print(l1)
```

```
Enter List of values: 10,20,30,10,20,40
[10, 20, 30, 40]
```

Q. Write a program to print different vowels present in the given word.

In [15]:

```
w=input("Enter word to search for vowels: ")
s=set(w)
v={'a','e','i','o','u'}
d=s.intersection(v)
print("The different vowel present in",w,"are",d)
print('The number of different vowels : ',len(d))
```

```
Enter word to search for vowels: Learning python is very easy
The different vowel present in Learning python is very easy are {'o', 'a',
'i', 'e'}
The number of different vowels :  4
```

In [ ]:

```

```

Dictionary Data Type

Date: 07-05-2020 Day 1

Topics Covered:

1. Introduction

2. Creation of Dictionary objects

3. Accessing data from the Dictionary

4. Updating the Dictionary

5. Deleting the elements from Dictionary

6. Important functions of Dictionary

```
i) dict()

ii) len()

iii) clear()

iv) get()

v) pop()

vi) popitem()

vii) keys()

viii) values()

ix) items()

x) copy()

xi) setdefault()

xii) update()
```

7. Dictionary Comprehension

Example Programs

1. Introduction

- We can use List,Tuple and Set to represent a group of individual objects as a single entity.
- If we want to represent a group of objects as key-value pairs then we should go for Dictionary.

Eg:

rollno----name

phone number--address

ipaddress---domain name

Key features of Dictionary Data type :

1. Duplicate keys are not allowed but values can be duplicated.
2. Hetrogeneous objects are allowed for both key and values.
3. insertion order is not preserved.
4. Dictionaries are mutable.
5. Dictionaries are dynamic.
6. indexing and slicing concepts are not applicable.

Note:

- In C++ and Java Dictionaries are known as **"Map"** where as in Perl and Ruby it is known as **"Hash"**.

2. Creation of Dictionary objects

If you want to create an empty dictionary, we use the following approach:

In [1]:

```
d = {}
print(type(d))
```

```
<class 'dict'>
```

We can create an empty dictionary using **dict()** function also.

In [2]:

```
d = dict()
print(type(d))
```

```
<class 'dict'>
```

We can add entries into a dictionary as follows:

- **d[key] = value**

In [8]:

```
d[100]="karthi"
d[200]="sahasra"
d[300]="sri"
d['rgm'] = 'Nandyal'
print(d) #{100: 'karthi', 200: 'sahasra', 300: 'sri', 'rgm' : 'Nandyal'}
```

```
{100: 'karthi', 200: 'sahasra', 300: 'sri', 'rgm': 'Nandyal'}
```

If we know data in advance then we can create dictionary as follows:

In [4]:

```
d={100:'karthi' ,200:'sahasra', 300:'sri'}
print(d)
```

```
{100: 'karthi', 200: 'sahasra', 300: 'sri'}
```

3. Accessing data from the dictionary

We can access data by using keys.

In [12]:

```
d={'a':'apple' ,'b':'banana', 'c':'cat'}
print(d['b'])
```

```
banana
```

If the specified key is not available then we will get **KeyError**.

In [13]:

```
d={'a':'apple' ,'b':'banana', 'c':'cat'}
print(d['z'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-13-15b56079ad88> in <module>
      1 d={'a':'apple' ,'b':'banana', 'c':'cat'}
----> 2 print(d['z'])

KeyError: 'z'
```

We can prevent this by checking whether key is already available or not by using **has_key()** function (or) by using **in** operator.

d.has_key(400) ==> returns 1 if key is available otherwise returns 0

Note :

- **has_key()** function is available only in Python 2 but not in Python 3.

Hence compulsory we have to use in operator.

In [16]:

```
d={'a':'apple' ,'b':'banana', 'c':'cat'}
if 'b' in d:
    print(d['b'])
```

```
banana
```

In [18]:

```
d={'a':'apple' ,'b':'banana', 'c':'cat'}
if 'z' in d:
    print(d['z'])     # If the key is not there in the dictionary, it wont give any key err
```

Example Program :

Q. Write a program to enter name and percentage marks in a dictionary and display information on the screen.

In [23]:

```
rec={}
n=int(input("Enter number of students: "))
i=1
while i <= n:
    name=input("Enter Student Name: ")
    marks=input("Enter % of Marks of Student: ")
    rec[name]=marks
    i=i+1
print("Name of Student","\t","% of Marks")
for x in rec:
    print("\t",x,"\t",rec[x])         # x ===> key       rec[x] =====> value
```

```
Enter number of students: 3
Enter Student Name: sourav
Enter % of Marks of Student: 89
Enter Student Name: sachin
Enter % of Marks of Student: 77
Enter Student Name: dravid
Enter % of Marks of Student: 77
Name of Student          % of Marks
         sourav          89
         sachin          77
         dravid          77
```

Date: 08-05-2020 Day 2

4. Updating the Dictionary

Syntax:

d[key]=value

- If the key is not available then a new entry will be added to the dictionary with the specified key-value pair.
- If the key is already available then old value will be replaced with new value.

Eg :

In [1]:

```
d={100:"karthi",200:"sahasra",300:"sri"}
print(d)
d[400]="sachin"
print(d)
d[100]="sourav"
print(d)
```

```
{100: 'karthi', 200: 'sahasra', 300: 'sri'}
{100: 'karthi', 200: 'sahasra', 300: 'sri', 400: 'sachin'}
{100: 'sourav', 200: 'sahasra', 300: 'sri', 400: 'sachin'}
```

5. Deleting the elements from Dictionary

Syntax :

del d[key]

- It deletes entry associated with the specified key.
- If the key is not available then we will get **KeyError**.

Eg :

In [2]:

```
d={100:"karthi",200:"sahasra",300:"sri"}
print(d)
del d[100]
print(d)
del d[400]
```

```
{100: 'karthi', 200: 'sahasra', 300: 'sri'}
{200: 'sahasra', 300: 'sri'}

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-2-a42fad35d4cc> in <module>
      3 del d[100]
      4 print(d)
----> 5 del d[400]

KeyError: 400
```

Note : Let us discuss about few more functions related to delete the contents of a dictionary.

1. clear():

- This function is used to remove all entries from the dictionary.

Eg :

In [3]:

```
d={100:"karthi",200:"sahasra",300:"sri"}
print(d)
d.clear()
print(d)
```

```
{100: 'karthi', 200: 'sahasra', 300: 'sri'}
{}
```

2.del:

To delete total dictionary, we can use **del** command .Now we cannot access dictionary **d.**

Eg :

In [4]:

```
d={100:"karthi",200:"sahasra",300:"sri"}
print(d)
del d
print(d)    # d can not access so we will get NameError
```

```
{100: 'karthi', 200: 'sahasra', 300: 'sri'}

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-a93a2726b01d> in <module>
      2 print(d)
      3 del d
----> 4 print(d)

NameError: name 'd' is not defined
```

Example:

In [7]:

```
list = ['sourav','sachin','rahul']
d={100:list}         # here, value is a list which cinsists of multiple objects which are a
print(d)
```

```
{100: ['sourav', 'sachin', 'rahul']}
```

6. Important functions of Dictionary

1. dict():

This function is used to create a dictionary.

In [13]:

```
d=dict()                                            #It creates empty dictionary
print(d)
d=dict({100:"karthi",200:"saha"})                   #It creates dictionary with specifie
print(d)
d=dict([(100,"karthi"),(200,"saha"),(300,"sri")])     #It creates dictionary with the giver
print(d)
d=dict(((100,"karthi"),(200,"saha"),(300,"sri")))     #It creates dictionary with the giver
print(d)
d=dict({(100,"karthi"),(200,"saha"),(300,"sri")})     #It creates dictionary with the giver
print(d)
d=dict({[100,"karthi"],[200,"saha"],[300,"sri"]})     #It creates dictionary with the giver
print(d)
```

```
{}
{100: 'karthi', 200: 'saha'}
{100: 'karthi', 200: 'saha', 300: 'sri'}
{100: 'karthi', 200: 'saha', 300: 'sri'}
{300: 'sri', 200: 'saha', 100: 'karthi'}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-683bd03bcacb> in <module>
      9 d=dict({(100,"karthi"),(200,"saha"),(300,"sri")})     #It creates di
ctionary with the given set of tuple elements
     10 print(d)
---> 11 d=dict({[100,"karthi"],[200,"saha"],[300,"sri"]})     #It creates di
ctionary with the given set of list elements
     12 print(d)

TypeError: unhashable type: 'list'
```

Note :

- Compulsory internally we need to take tuple only is acceptable. If you take list it gives the above specified error.

If the key values are available in the form of tuple, then all those tuple values can be coverted into dictionary by using 'dict()' function.

2. len()

- Returns the number of items in the dictionary.

In [17]:

```
d=dict({100:"karthi",200:"saha"})    #It creates dictionary with specified elements
print(d)
print(len(d))
```

```
{100: 'karthi', 200: 'saha'}
2
```

3. clear():

- To remove all elements from the dictionary.

In [16]:

```
d=dict({100:"karthi",200:"saha"})    #It creates dictionary with specified elements
print(d)
d.clear()
print(d)
```

```
{100: 'karthi', 200: 'saha'}
{}
```

4. get():

- To get the value associated with the key.
- Two forms of **get()** method is available in Python.

i. d.get(key)

- If the key is available then returns the corresponding value otherwise returns None.It wont raise any error.

In [18]:

```
d=dict({100:"karthi",200:"saha"})    #It creates dictionary with specified elements
print(d.get(100))
```

```
karthi
```

In [19]:

```
d=dict({100:"karthi",200:"saha"})    #It creates dictionary with specified elements
print(d.get(500))
```

```
None
```

ii. d.get(key,defaultvalue)

- If the key is available then returns the corresponding value otherwise returns default value.

In [21]:

```
d=dict({100:"karthi",200:"saha"})    #It creates dictionary with specified elements
print(d.get(100,'ravan'))
```

```
karthi
```

In [14]:

```
d=dict({100:"karthi",200:"saha"})    #It creates dictionary with specified elements
print(d.get(500,'ravan'))
print(d)
```

```
ravan
{100: 'karthi', 200: 'saha'}
```

Another Example :

In [23]:

```
d={100:"karthi",200:"saha",300:"sri"}
print(d[100]) #karthi
print(d[400]) #KeyError:400
print(d.get(100)) #karthi
print(d.get(400)) #None
print(d.get(100,"Guest")) #karthi
print(d.get(400,"Guest")) #Guest
```

```
karthi

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-23-2890397dced0> in <module>
      1 d={100:"karthi",200:"saha",300:"sri"}
      2 print(d[100]) #karthi
----> 3 print(d[400]) #KeyError:400
      4 print(d.get(100)) #karthi
      5 print(d.get(400)) #None

KeyError: 400
```

In [24]:

```
d={100:"karthi",200:"saha",300:"sri"}
print(d[100]) #karthi
#print(d[400]) #KeyError:400
print(d.get(100)) #karthi
print(d.get(400)) #None
print(d.get(100,"Guest")) #karthi
print(d.get(400,"Guest")) #Guest
```

```
karthi
karthi
None
karthi
Guest
```

5. pop():

Syntax :

d.pop(key)

- It removes the entry associated with the specified key and returns the corresponding value.
- If the specified key is not available then we will get **KeyError**.

In [26]:

```
d={100:"karthi",200:"saha",300:"sri"}
print(d)
print(d.pop(100))
print(d)
print(d.pop(400))
```

```
{100: 'karthi', 200: 'saha', 300: 'sri'}
karthi
{200: 'saha', 300: 'sri'}

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-26-787766bb18c2> in <module>
      3 print(d.pop(100))
      4 print(d)
----> 5 print(d.pop(400))

KeyError: 400
```

6. popitem():

- It removes an arbitrary item(key-value) from the dictionaty and returns it.

In [29]:

```
d={100:"karthi",200:"saha",300:"sri"}
print(d)
print(d.popitem())
print(d.popitem())
print(d)
print(d.pop(400))          # KeyError
```

```
{100: 'karthi', 200: 'saha', 300: 'sri'}
(300, 'sri')
(200, 'saha')
{100: 'karthi'}

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-29-4185b7c5bad9> in <module>
      4 print(d.popitem())
      5 print(d)
----> 6 print(d.pop(400))          # KeyError

KeyError: 400
```

If the dictionary is empty then we will get **KeyError.**

In [28]:

```
d ={}
print(d.popitem())         #KeyError: 'popitem(): dictionary is empty'
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-28-14f741a4e5d5> in <module>
      1 d ={}
----> 2 print(d.popitem())

KeyError: 'popitem(): dictionary is empty'
```

In [30]:

```
d={100:"karthi",200:"saha",300:"sri"}
print(d)
print(d.popitem())
print(d.popitem())
print(d.popitem())
print(d.popitem())
print(d)
```

```
{100: 'karthi', 200: 'saha', 300: 'sri'}
(300, 'sri')
(200, 'saha')
(100, 'karthi')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-30-17881d89d74e> in <module>
      4 print(d.popitem())
      5 print(d.popitem())
----> 6 print(d.popitem())
      7 print(d)

KeyError: 'popitem(): dictionary is empty'
```

7. keys():

- It returns all keys associated with dictionary.

Eg :

In [33]:

```
d={100:"karthi",200:"saha",300:"sri"}
print(d.keys())
for key in d.keys():
    print(key)
```

```
dict_keys([100, 200, 300])
100
200
300
```

8. values():

- It returns all values associated with the dictionary.

In [34]:

```
d={100:"karthi",200:"saha",300:"sri"}
print(d.values())
for key in d.values():
    print(key)
```

```
dict_values(['karthi', 'saha', 'sri'])
karthi
saha
sri
```

9. items():

- It returns list of tuples representing key-value pairs like as shown below.

[(k,v),(k,v),(k,v)]

In [38]:

```
d={100:"karthi",200:"saha",300:"sri"}
list = d.items()
print(list)
```

```
dict_items([(100, 'karthi'), (200, 'saha'), (300, 'sri')])
```

In [35]:

```
d={100:"karthi",200:"saha",300:"sri"}
for k,v in d.items():
    print(k,"--",v)
```

```
100 -- karthi
200 -- saha
300 -- sri
```

10. copy():

- This method is used to create exactly duplicate dictionary(cloned copy).

In [37]:

```
d={100:"karthi",200:"saha",300:"sri"}
d1=d.copy()
print(d1)
print(d)
```

```
{100: 'karthi', 200: 'saha', 300: 'sri'}
{100: 'karthi', 200: 'saha', 300: 'sri'}
```

11. setdefault():

Syntax :

d.setdefault(k,v)

- If the key is already available then this function returns the corresponding value.
- If the key is not available then the specified key-value will be added as new item to the dictionary.

Eg :

In [39]:

```
d={100:"karthi",200:"saha",300:"sri"}
print(d.setdefault(400,"sourav"))
print(d)
print(d.setdefault(100,"sachin"))
print(d)
```

```
sourav
{100: 'karthi', 200: 'saha', 300: 'sri', 400: 'sourav'}
karthi
{100: 'karthi', 200: 'saha', 300: 'sri', 400: 'sourav'}
```

12. update():

Syntax :

d.update(x)

- All items present in the dictionary x will be added to dictionary **d**.

In [42]:

```
d={100:"karthi",200:"saha",300:"sri"}
d1 ={'a':'apple', 'b':'banana'}
d.update(d1)
print(d)
```

```
{100: 'karthi', 200: 'saha', 300: 'sri', 'a': 'apple', 'b': 'banana'}
```

In [44]:

```
d={100:"karthi",200:"saha",300:"sri"}
d1 ={'a':'apple', 'b':'banana'}
d2 = {777:'A', 888:'B'}
d.update(d1,d2)             # For ipdate method. you need to pass single argument only.
print(d)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-58a2bfd142f6> in <module>
      2 d1 ={'a':'apple', 'b':'banana'}
      3 d2 = {777:'A', 888:'B'}
----> 4 d.update(d1,d2)             # For update method. you need to pass si
ngle argument only.
      5 print(d)

TypeError: update expected at most 1 arguments, got 2
```

In [46]:

```
d={100:"karthi",200:"saha",300:"sri"}
d1 ={'a':'apple', 'b':'banana'}
d2 = {777:'A', 888:'B'}
d.update([(777,'A')])   # For ipdate method. you can pass list of tuple as an argument. i.e
print(d)
```

```
{100: 'karthi', 200: 'saha', 300: 'sri', 777: 'A'}
```

In [47]:

```
d={100:"karthi",200:"saha",300:"sri"}
d1 ={'a':'apple', 'b':'banana'}
d2 = {777:'A', 888:'B'}
d.update([(777,'A'),(888,'B'),(999,'C')])  # you can add any no.of list of tuple elements.
print(d)
```

```
{100: 'karthi', 200: 'saha', 300: 'sri', 777: 'A', 888: 'B', 999: 'C'}
```

Date: 09-05-2020 Day 3

Example Programs

Q 1. Write a program to take dictionary from the keyboard and print the sum of values.

In [3]:

```
d=eval(input("Enter dictionary:"))
s=sum(d.values())
print("Sum= ",s)
```

```
Enter dictionary:{'A':100,'B':200,'c':300}
Sum=  600
```

In [4]:

```
d=eval(input("Enter dictionary:"))
s=sum(d.values())
print("Sum= ",s)
```

```
Enter dictionary:'A':100,'B':200,'c':300

Traceback (most recent call last):

  File "C:\Users\HP\Anaconda3\lib\site-packages\IPython\core\interactiveshel
l.py", line 3296, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  File "<ipython-input-4-7372dea074de>", line 1, in <module>
    d=eval(input("Enter dictionary:"))

  File "<string>", line 1
    'A':100,'B':200,'c':300
       ^
SyntaxError: invalid syntax
```

In [5]:

```
l = [10,20,30,40]
s = sum(l)          # sum() function works on list also
print('Sum is : ',s)
```

```
Sum is :  100
```

In [6]:

```
l = (10,20,30,40)
s = sum(l)          # sum() function works on tuple also
print('Sum is : ',s)
```

```
Sum is :  100
```

In [7]:

```
l = {10,20,30,40}
s = sum(l)          # sum() function works on set also
print('Sum is : ',s)
```

```
Sum is :  100
```

Note : sum() function can work on any sequence.

Q 2. Write a program to find number of occurrences of each letter present in the given string.

In [9]:

```
word=input("Enter any word: ")
d={}
for x in word:
    d[x]=d.get(x,0)+1           # we are creating dictionary with the given word    ====>
for k,v in d.items():
    print(k,"occurred ",v," times")
```

```
Enter any word: mississippi
m occurred  1  times
i occurred  4  times
s occurred  4  times
p occurred  2  times
```

In [10]:

```
word=input("Enter any word: ")
d={}
for x in word:
    d[x]=d.get(x,0)+1           # we are creating dictionary with the given word    ====>
for k,v in sorted(d.items()):   # To sort all the items of the dictionary in alphabetical c
    print(k,"occurred ",v," times")
```

```
Enter any word: mississippi
i occurred  4  times
m occurred  1  times
p occurred  2  times
s occurred  4  times
```

Q 3. Write a program to find number of occurrences of each vowel present in the given string.

In [11]:

```
word=input("Enter any word: ")
vowels={'a','e','i','o','u'}
d={}
for x in word:
    if x in vowels:
        d[x]=d.get(x,0)+1
for k,v in sorted(d.items()):
    print(k,"occurred ",v," times")
```

```
Enter any word: doganimaldoganimal
a occurred  4  times
i occurred  2  times
o occurred  2  times
```

Q 4. Write a program to accept student name and marks from the keyboard and creates a dictionary. Also display student marks by taking student name as input.

In [12]:

```
n=int(input("Enter the number of students: "))
d={}
for i in range(n):
    name=input("Enter Student Name: ")
    marks=input("Enter Student Marks: ")
    d[name]=marks            # assigninng values to the keys of the dictionary 'd'
while True:
    name=input("Enter Student Name to get Marks: ")
    marks=d.get(name,-1)
    if marks== -1:
        print("Student Not Found")
    else:
        print("The Marks of",name,"are",marks)   # print('The marks of {} :{}'.format(name,
    option=input("Do you want to find another student marks[Yes|No]")
    if option=="No":
        break
print("Thanks for using our application")
```

```
Enter the number of students: 5
Enter Student Name: Sourav
Enter Student Marks: 90
Enter Student Name: Sachin
Enter Student Marks: 87
Enter Student Name: Rahul
Enter Student Marks: 86
Enter Student Name: Parthiv
Enter Student Marks: 56
Enter Student Name: Robin
Enter Student Marks: 66
Enter Student Name to get Marks: Sourav
The Marks of Sourav are 90
Do you want to find another student marks[Yes|No]Y
Enter Student Name to get Marks: Robin
The Marks of Robin are 66
Do you want to find another student marks[Yes|No]y
Enter Student Name to get Marks: karthi
Student Not Found
Do you want to find another student marks[Yes|No]No
Thanks for using our application
```

7. Dictionary Comprehension

- Comprehension concept applicable for dictionaries also.

In [20]:

```
squares={x:x*x for x in range(1,6)}
print(squares)
doubles={x:2*x for x in range(1,6)}
print(doubles)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
{1: 2, 2: 4, 3: 6, 4: 8, 5: 10}
```

```
In [ ]:
```

UNIT - 6

Functions

Date: 10-05-2020 Day 1

Topics Covered:

1. Introduction

2. Types of Functions

3. Parameters

```
        i) Types of Parameters


        ii) Return statement


        iii) Returning of mutiple values from a function
```

4. Types of Variables

5. Recursive Functions

6. Anonymous Functions

7. Function Aliasing

8. Nested Functions

1. Introduction

If a group of statements is repeatedly required then it is not recommended to write these statements everytime seperately.We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting. This unit is nothing but **function**.

The main advantage of functions is **code Reusability.**

-

Note:

> In other languages functions are known as methods,procedures,subroutines etc.

Eg :

In [1]:

```
a = 20
b = 10
print('The Sum : ', a + b)
print('The Difference : ', a - b)
print('The Product : ', a * b)
```

```
The Sum :  30
The Difference :  10
The Product :  200
```

In [2]:

```
a = 20
b = 10
print('The Sum : ', a + b)
print('The Difference : ', a - b)
print('The Product : ', a * b)
a = 200
b = 100
print('The Sum : ', a + b)
print('The Difference : ', a - b)
print('The Product : ', a * b)
```

```
The Sum :  30
The Difference :  10
The Product :  200
The Sum :  300
The Difference :  100
The Product :  20000
```

In [3]:

```
a = 20
b = 10
print('The Sum : ', a + b)
print('The Difference : ', a - b)
print('The Product : ', a * b)
a = 200
b = 100
print('The Sum : ', a + b)
print('The Difference : ', a - b)
print('The Product : ', a * b)
a = 2000
b = 1000
print('The Sum : ', a + b)
print('The Difference : ', a - b)
print('The Product : ', a * b)
```

```
The Sum :  30
The Difference :  10
The Product :  200
The Sum :  300
The Difference :  100
The Product :  20000
The Sum :  3000
The Difference :  1000
The Product :  2000000
```

Here, we written 15 lines of code. What is the problem in this code?

Have you observed that the same code (3 lines) is repeating thrice in the code. Generally we never recommended to wrtie the group of statements repeatedly in the program.

Problems of writing the same code repeatedly in the program:

1. Length of the program increases.

2. Readability of the program decreases.

3. No Code Reusabilty.

How can you resolve this problem?

- We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting. This unit is nothing but **function**.

How can you solve this problem by defining function for the above example?

In [4]:

```
def calculate(a,b):
    print('The Sum : ', a + b)
    print('The Difference : ', a - b)
    print('The Product : ', a * b)

a = 20
b = 10
calculate(a,b)
```

```
The Sum :  30
The Difference :  10
The Product :  200
```

In [5]:

```
def calculate(a,b):
    print('The Sum : ', a + b)
    print('The Difference : ', a - b)
    print('The Product : ', a * b)

a = 20
b = 10
calculate(a,b)
a = 200
b = 100
calculate(a,b)
```

```
The Sum :  30
The Difference :  10
The Product :  200
The Sum :  300
The Difference :  100
The Product :  20000
```

In [6]:

```
def calculate(a,b):
    print('The Sum : ', a + b)
    print('The Difference : ', a - b)       # Function 'calculate()' executes 3 times
    print('The Product : ', a * b)

a = 20
b = 10
calculate(a,b)                            # Function call
a = 200
b = 100
calculate(a,b)
a = 2000
b = 1000
calculate(a,b)
```

```
The Sum :  30
The Difference :  10
The Product :  200
The Sum :  300
The Difference :  100
The Product :  20000
The Sum :  3000
The Difference :  1000
The Product :  2000000
```

In [8]:

```
def calculate(a,b):
    print('The Sum : ', a + b)
    print('The Difference : ', a - b)       # Function 'calculate()' executes 3 times
    print('The Product : ', a * b)

calculate(20,10)                            # Function call
calculate(200,100)
calculate(2000,1000)                       # Concise code resulted because of code reusabil
```

```
The Sum :  30
The Difference :  10
The Product :  200
The Sum :  300
The Difference :  100
The Product :  20000
The Sum :  3000
The Difference :  1000
The Product :  2000000
```

Note:

We are writing the function once and calling that function 'n' times.

2. Types of Functions

Python supports 2 types of functions:

1. Built in Functions

2. User Defined Functions

1. Built in Functions

- The functions which are coming along with Python software automatically,are called **built in functions** or **pre defined functions**.

Eg:

id()

type()

input()

eval() etc..

2. User Defined Functions:

- The functions which are developed by programmer explicitly according to business requirements, are called **user defined functions.**

Syntax to create user defined functions:

def function_name(parameters) :

```
Stmt 1


Stmt 2

---

Stmt n
```

return value

Note:

While creating functions we can use 2 keywords:

1. def (mandatory)

2. return (optional)

Eg 1: Write a function to print Hello message

In [9]:

```
def wish():
    print("Hello Good Morning")
wish()
wish()
wish()
```

```
Hello Good Morning
Hello Good Morning
Hello Good Morning
```

3. Parameters:

- **Parameters are inputs to the function.**
- If a function contains parameters,then at the time of calling,compulsory we should provide values,otherwise we will get error.

i. Types of Parameters in Python:

1. Positional Parameters:

- In the case of positional arguments, number of arguments must be same.
- In the case of positional arguments, order of the arguments is important.

2. Keyword (i.e., Parameter name) Parameters:

- In the case of keyword arguments, order of the arguments is not important.
- In the case of keyword arguments, number of arguments must be same.

3. Default Parameters:

- You can define default value for the arguments.
- If you are not passing any argument, then default values by default will be considered.
- After default arguments you should not take normal arguments. (i.e., Default arguments you need to take at last)

4. Variable length Parameters:

- Sometimes we can pass variable number of arguments to our function,such type of arguments are called variable length arguments.
- We can declare a variable length argument with * symbol as follows

```
def f1(*n):
```

- We can call this function by passing any number of arguments including zero number. Internally all these values represented in the form of tuple.

Eg: Write a function to take name of the student as input and print wish message by name.

In [11]:

```
def wish(name):
    print("Hello",name," Good Morning")

wish("Karthi")
wish("Sahasra")
```

```
Hello Karthi  Good Morning
Hello Sahasra  Good Morning
```

Eg: Write a function to take number as input and print its square value.

In [1]:

```
def squareIt(number):
    print("The Square of",number,"is", number*number)
squareIt(4)
squareIt(5)
squareIt(7)
```

```
The Square of 4 is 16
The Square of 5 is 25
The Square of 7 is 49
```

Date: 10-05-2020 Day 2

ii. return statement

- Function can take input values as parameters and executes business logic, and returns output to the caller with **return statement.**
- Python function can return any number of values at a time by using a return statement.
- Default return value of any python function is **None**.

Eg :

In [3]:

```
def wish():
    print('hello')
#print(wish())
wish()
```

```
hello
```

In [4]:

```
def wish():
    print('hello')
print(wish())
#wish()
```

```
hello
None
```

Simple Example Programs:

Q 1. Write a function to accept 2 numbers as input and return sum.

In [43]:

```
def add(x,y):
    return x+y
result=add(10,20)
print("The sum is",result)
print("The sum is",add(100,200))
```

```
The sum is 30
The sum is 300
```

Note : If we are not writing return statement then default return value is None.

Eg :

In [44]:

```
def f1():
    print("Hello")
f1()
print(f1())
```

```
Hello
Hello
None
```

Q 2. Write a function to check whether the given number is even or odd?

In [45]:

```
def even_odd(num):
    if num%2==0:
        print(num,"is Even Number")
    else:
        print(num,"is Odd Number")
even_odd(10)
even_odd(15)
```

```
10 is Even Number
15 is Odd Number
```

Q 3. Write a function to find factorial of given number.

In [47]:

```
def fact(num):
    result=1
    while num>=1:
        result=result*num
        num=num-1
    return result
for i in range(1,5):
    print("The Factorial of",i,"is :",fact(i))
```

```
The Factorial of 1 is : 1
The Factorial of 2 is : 2
The Factorial of 3 is : 6
The Factorial of 4 is : 24
```

iii. Returning multiple values from a function:

- In other languages like C,C++ and Java, function can return atmost one value. But in Python, a function can return any number of values.

Eg : Python program to return multiple values at a time using a return statement.

In [6]:

```
def calc(a,b):         # Here, 'a' & 'b' are called positional arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div

a,b,c,d = calc(100,50)          # Positional arguments
print(a,b,c,d)
```

```
150 50 5000 2.0
```

Alternate Way :

In [7]:

```
def calc(a,b):                       # Positional Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div

t = calc(100,50)
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [8]:

```
def calc(a,b):                          # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div

t = calc(a = 100, b = 50)              # keyword arguments Arguments
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [9]:

```
def calc(a,b):                          # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div

t = calc(b = 50, a = 100)              # keyword arguments Arguments
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

Some more examples on keyword arguments

In [10]:

```
def calc(a,b):                          # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div

t = calc(100, b = 50)              # It is perfectly valid
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [12]:

```
def calc(a,b):                     # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div

t = calc(b = 50, 100)          # It is invalid, because positional argument should follow k
for x in t:                   # first keyword argument then possitional argument is not all
    print(x)
```

```
  File "<ipython-input-12-66d17e37e83f>", line 8
    t = calc(b = 50, 100)          # It is invalid, because positional argum
ent should follow keyword argument
                    ^
SyntaxError: positional argument follows keyword argument
```

In [13]:

```
def calc(a,b):                     # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div

t = calc(50, a = 50)          # It is also invalid
for x in t:
    print(x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-b7af0ee55c3b> in <module>
      6     return sum,sub,mul,div
      7 
----> 8 t = calc(50, a = 50)          # It is also invalid
      9 for x in t:
     10     print(x)

TypeError: calc() got multiple values for argument 'a'
```

Another Example:

In [11]:

```
def wish(name,msg):
    print('Hello',name,msg)

wish(name = 'Karthi',msg = 'Good Morning') #order is not important, but no.of arguments is
wish(msg = 'Good Morning',name = 'Karthi')
```

```
Hello Karthi Good Morning
Hello Karthi Good Morning
```

Eg : Program on default parameters.

In [15]:

```
def wish(name ='Guest',msg):        # After default argument, we should not take non-defaul
    print('Hello',name,msg)
```

```
  File "<ipython-input-15-1e1d7a3c73c0>", line 1
    def wish(name ='Guest',msg):        # After default argument, we should
 not take non-default argument
           ^
SyntaxError: non-default argument follows default argument
```

In [13]:

```
def wish(msg,name ='Guest'):
    print(msg,name)

wish('Hello','Karthi')
```

```
Hello Karthi
```

In [12]:

```
def wish(msg,name ='Guest'):
    print(msg,name)

wish('Hello')
```

```
Hello Guest
```

In [14]:

```
def wish(msg,name ='Guest'):
    print(msg,name)

wish()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-993af2d34958> in <module>
      2     print(msg,name)
      3 
----> 4 wish()

TypeError: wish() missing 1 required positional argument: 'msg'
```

Note:

- You can give any number of default arguments.

Eg :

In [18]:

```
def wish(name ='Guest',msg='Good Morning'):
    print('Hello',name,msg)

wish()
```

```
Hello Guest Good Morning
```

Eg :

In [20]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)

wish(99,48,'Karthi')  # Valid
```

```
Student Name: Karthi
Student Age: 48
Student Marks: 99
Message: Good Morning
```

In [21]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)

wish(age=48,marks = 100)  # valid
```

```
Student Name: Guest
Student Age: 48
Student Marks: 100
Message: Good Morning
```

In [25]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)

wish(100,age=46,msg='Bad Morning',name='Karthi')  # valid
```

```
Student Name: Karthi
Student Age: 46
Student Marks: 100
Message: Bad Morning
```

In [24]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)

wish(marks=100,46,msg='Bad Morning',name = 'Karthi')  # invalid,  You must specify age alsc
```

```
  File "<ipython-input-24-2c5c0c3096c5>", line 7
    wish(marks=100,46,msg='Bad Morning',name = 'Karthi')  # valid
                  ^
SyntaxError: positional argument follows keyword argument
```

In [26]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)

wish(46,marks=100,msg='Bad Morning',name = 'Karthi')  # invalid
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-e767702586d4> in <module>
      5     print('Message:',msg)
      6 
----> 7 wish(46,marks=100,msg='Bad Morning',name = 'Karthi')  # invalid

TypeError: wish() got multiple values for argument 'marks'
```

Eg : Program on variable length parameters.

In [23]:

```
def sum(a,b):
    print(a+b)
sum(10,20)
```

```
30
```

Now it is working correctly. After some time my requiremnet is as follows:

sum(10,20,30)

In [24]:

```
def sum(a,b):
    print(a+b)              # This sum() function we can't use for the new requirement.
sum(10,20,30)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-2918cd889583> in <module>
      1 def sum(a,b):
      2     print(a+b)
----> 3 sum(10,20,30)

TypeError: sum() takes 2 positional arguments but 3 were given
```

In [25]:

```
def sum(a,b,c):
    print(a+b+c)      # we have to go for another sum() function
sum(10,20,30)
```

```
60
```

Now it is working correctly. After some time my requiremnet is as follows:

sum(10,20,30,40)

In [26]:

```
def sum(a,b,c):
    print(a+b+c)
sum(10,20,30,40)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-b7625a84ded9> in <module>
      1 def sum(a,b,c):
      2     print(a+b+c)
----> 3 sum(10,20,30,40)

TypeError: sum() takes 3 positional arguments but 4 were given
```

Once again the same problem. we should go for another sum() function.

In [27]:

```
def sum(a,b,c,d):
    print(a+b+c+d)      # we have to go for another sum() function
sum(10,20,30,40)
```

```
100
```

If you change the number of arguments, then automatically for every change, compusorily we need to go for new function unnecessarily. Because of this length of the code is going to increase.

To overcome this problem we should go for **variable length arguments**.

In [29]:

```
def sum(*n):                 # Here, 'n' is a variable length argument. actually variable l
    result =0
    for x in n:
        result = result + x
    print(result)
sum(10,20,30,40)
```

```
100
```

In [30]:

```
def sum(*n):                 # Here, 'n' is a variable length argument. actually variable l
    result =0
    for x in n:
        result = result + x
    print(result)
sum(10,20,30)
```

```
60
```

In [31]:

```
def sum(*n):                 # Here, 'n' is a variable length argument. actually variable l
    result =0
    for x in n:
        result = result + x
    print(result)
sum(10,20)
```

```
30
```

In [32]:

```
def sum(*n):                 # Here, 'n' is a variable length argument. actually variable l
    result =0
    for x in n:
        result = result + x
    print(result)
sum(10)
```

```
10
```

In [33]:

```
def sum(*n):                 # Here, 'n' is a variable length argument. actually variable l
    result =0
    for x in n:
        result = result + x
    print(result)
sum()
```

```
0
```

In [35]:

```
def sum(*n):                # Here, 'n' is a variable length argument. actually variable l
    result =0
    for x in n:
        result = result + x
    print('The Sum is : ', result)
sum(10,20,30,40)
sum(10,20,30)
sum(10,20)
sum(10)
sum()
```

```
The Sum is :  100
The Sum is :  60
The Sum is :  30
The Sum is :  10
The Sum is :  0
```

Note : Same function is used for variable number of arguments.

Key Point 1:

- We can mix variable length arguments with positional arguments.
- You can take positional arguments and variable length arguments simultaneously.

In [38]:

```
def sum(name,*n):
    result =0
    for x in n:
        result = result + x
    print("The Sum by", name, ": ", result)
sum('Robin',10,20,30,40)
sum('Rahul',10,20,30)
sum('Sachin',10,20)
sum('Sourav',10)
sum('Karthi')
```

```
The Sum by Robin :  100
The Sum by Rahul :  60
The Sum by Sachin :  30
The Sum by Sourav :  10
The Sum by Karthi :  0
```

Note:

- **Rule :** After variable length argumenst,if we are taking any other arguments then we should provide values as keyword arguments.

In [39]:

```
def sum(*n,name):
    result =0
    for x in n:
        result = result + x
    print("The Sum by", name, ": ", result)
sum('Robin',10,20,30,40)
sum('Rahul',10,20,30)
sum('Sachin',10,20)
sum('Sourav',10)
sum('Karthi')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-b8733ebba999> in <module>
      4         result = result + x
      5     print("The Sum by", name, ": ", result)
----> 6 sum('Robin',10,20,30,40)
      7 sum('Rahul',10,20,30)
      8 sum('Sachin',10,20)

TypeError: sum() missing 1 required keyword-only argument: 'name'
```

In [40]:

```
def sum(*n,name):
    result =0
    for x in n:
        result = result + x
    print("The Sum by", name, ": ", result)
sum(name = 'Robin',10,20,30,40)
sum(name = 'Rahul',10,20,30)
sum(name = 'Sachin',10,20)
sum(name = 'Sourav',10)
sum(name ='Karthi')
```

```
  File "<ipython-input-40-19134896e44a>", line 6
    sum(name = 'Robin',10,20,30,40)
                      ^
SyntaxError: positional argument follows keyword argument
```

In [41]:

```
def sum(*n,name):
    result =0
    for x in n:
        result = result + x
    print("The Sum by", name, ": ", result)
sum(10,20,30,40,name = 'Robin')
sum(10,20,30,name = 'Rahul')
sum(10,20,name = 'Sachin')
sum(10,name = 'Sourav')
sum(name ='Karthi')
```

```
The Sum by Robin :  100
The Sum by Rahul :  60
The Sum by Sachin :  30
The Sum by Sourav :  10
The Sum by Karthi :  0
```

Another Example:

In [48]:

```
def f1(n1,*s):
    print(n1)
    for s1 in s:
        print(s1)

f1(10)
f1(10,20,30,40)
f1(10,"A",30,"B")
```

```
10
10
20
30
40
10
A
30
B
```

Conclusion :

- After variable length arguments, if you are taking any other argument, then we have to provide values as key word arguments only.
- If you pass first normal argument and then variable arguments, then there is no rule to follow. It works correctly.

Key Point 2:

Keyword variable length arguments :

- Now, Suppose if we want to pass any number of keyword arguments to a function, compulsorily we have to identify the difference with the above case (i.e., Passing of any number of positional arguments).
- We can declare key word variable length arguments also. For this we have to use **.

- We can call this function by passing any number of keyword arguments. Internally these keyword arguments will be stored inside a dictionary.

Eg :

In [42]:

```
def display(**kwargs):
    for k,v in kwargs.items():
        print(k,"=",v)
display(n1=10,n2=20,n3=30)
display(rno=100,name="Karthi",marks=70,subject="Python")
```

```
n1 = 10
n2 = 20
n3 = 30
rno = 100
name = Karthi
marks = 70
subject = Python
```

Case Study

In [6]:

```
def f(arg1,arg2,arg3=4,arg4=8):
    print(arg1,arg2,arg3,arg4)

f(3,2)       #3 2 4 8

f(10,20,30,40)   # 10,20,30,40

f(25,50,arg4=100)   # 25  50 4 100

f(arg4=2,arg1=3,arg2=4)   # 3 4 4 2

#f()      # TypeError: f() missing 2 required positional arguments: 'arg1' and 'arg2'

#f(arg3=10,arg4=20,30,40)    SyntaxError: positional argument follows keyword argument

#f(4,5,arg2=6)  #TypeError: f() got multiple values for argument 'arg2'

#f(4,5,arg3=5,arg5=6)  #TypeError: f() got an unexpected keyword argument 'arg5'
```

```
3 2 4 8
10 20 30 40
25 50 4 100
3 4 4 2
```

Note:

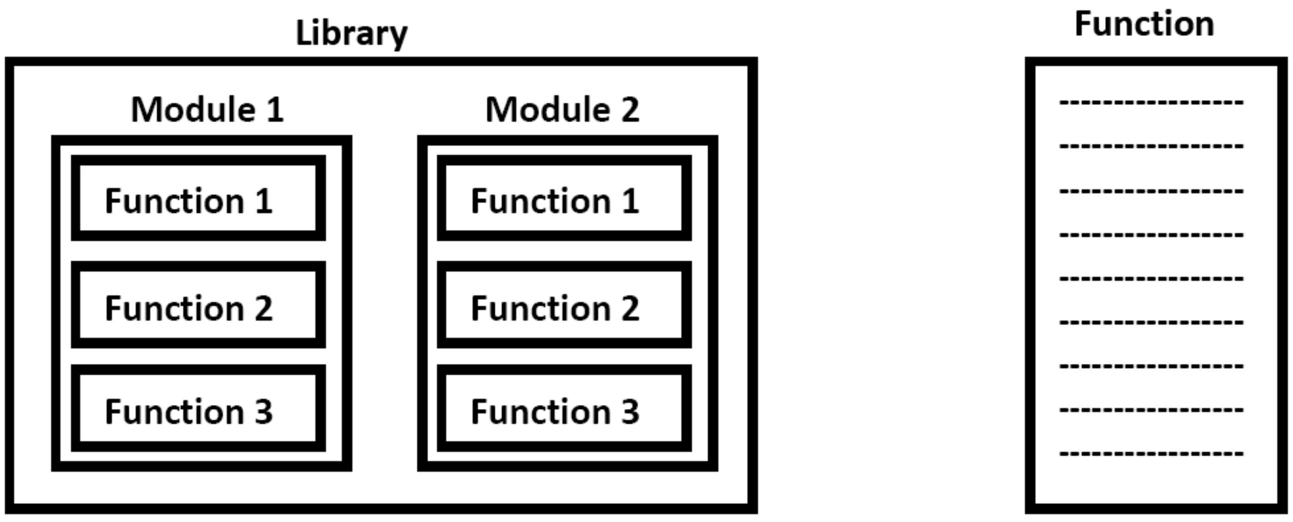
Function vs Module vs Package vs Library:

1. A group of lines with some name is called a function.

2. A group of functions saved to a file , is called Module.

3. A group of Modules is nothing but Package.

4. A group of related packages is nothing but Library.



4. Types of Variables:

Python supports 2 types of variables.

1. Global Variables

2. Local Variables

1. Global Variables

- The variables which are declared outside of function are called global variables.
- These variables can be accessed in all functions of that module.

Consider the following example,

In [1]:

```
a = 10                  # Global Variables

def f1():
    a = 20             # Local variable to the function 'f1'
    print(a)                # 20

def f2():
    print(a)                # 10

f1()
f2()
```

```
20
10
```

Suppose our requirement is, we don't want local variable. Can you please refer the local variable as the global variable only. How you can do that?

- For that, one special keyword is used, called as **global**.

global keyword:

We can use global keyword for the following 2 purposes:

1. To declare global variables explicitly inside function.
2. To make global variable available to the function so that we can perform required modifications.

Eg 1:

In [2]:

```
a=10
def f1():
    a=777
    print(a)
def f2():
    print(a)
f1()
f2()
```

```
777
10
```

Eg 2:

In [3]:

```
a=10
def f1():
    global a                    # To bring global variable to the function for required modi
    a=777                       # we are changing the value of the local variable
    print(a)
def f2():
    print(a)
f1()
f2()
```

```
777
777
```

Eg 3:

In [6]:

```
def f1():
    x = 10   # local variable of 'f1()'
    print(x)

def f2():
    print(x)  # local variable of  'f1()' can not accessed by function 'f2()'

f1()
f2()
```

```
10

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-949ab59188a5> in <module>
      7
      8 f1()
----> 9 f2()

<ipython-input-6-949ab59188a5> in f2()
      4
      5 def f2():
----> 6     print(x)  # local variable of  'f1()' can not accessed by functi
on 'f2()'
      7
      8 f1()

NameError: name 'x' is not defined
```

Here, if you make x of f1() as a global variable, problem will be solved. How can you make 'x' as global variable?

Eg 4:

In [7]:

```
def f1():
    global x
    x=10
    print(x)
def f2():
    print(x)
f1()
f2()
```

```
10
10
```

In [27]:

```
def f1():
    global x
    x=10
    print(x)
def f2():
    print(x)
f2()
f1()
```

```
10
10
```

In [8]:

```
def f1():
    global x = 10      # This syntax is invalid in Python
    print(x)
def f2():
    print(x)
f1()
f2()
```

```
  File "<ipython-input-8-c080f0bbb9d1>", line 2
    global x = 10      # This syntax is invalid in Python
             ^
SyntaxError: invalid syntax
```

Another Example :

In [31]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999                            # global variable 'a' is overrides the old value.
    print('f2 :',a)
f1()
f2()
```

```
f1 : 888
f2 : 999
```

In [33]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
def f3():
    print('f3 :',a)
f1()
f2()
f3()
```

```
f1 : 888
f2 : 999
f3 : 999
```

In [35]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
def f3():
    print('f3 :',a)
f3()
f1()
f2()
```

```
f3 : 888
f1 : 888
f2 : 999
```

In [36]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
def f3():
    print('f3 :',a)
f3()
f2()
f1()
```

```
f3 : 999
f2 : 999
f1 : 888
```

In [37]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
def f3():
    a = 1000
    print('f3 :',a)
f3()
f2()
f1()
```

```
f3 : 1000
f2 : 999
f1 : 888
```

Another Example :

In [ ]:

```
def f1():
    global a
    a = 888                       # global variable 'a' is overrides the old value.
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
f2()
f1()
```

Note:

- If global variable and local variable having the same name, then we can access global variable inside a function using **globals() function**.

Eg :

In [38]:

```
a=10           #global variable
def f1():
    a=777      #local variable
    print(a)
    #
f1()
```

```
777
```

In [40]:

```
a=10            #global variable
def f1():
    a=777        #local variable
    print(a)
    print(globals()['a'])        # globals() function cosisiting all global members related
f1()                             # here, 'a' is the key value.
```

```
777
10
```

Another Example :

In [41]:

```
def f1():
    a = 10         # SyntaxError: name 'a' is assigned to before global declaration
    global a
    a = 50
    print(a)
f1()
```

```
  File "<ipython-input-41-9834772fedac>", line 6
    f1()

^
SyntaxError: name 'a' is assigned to before global declaration
```

In [42]:

```
def f1():
    global a
    a = 10
    a = 50
    print(a)
f1()
```

```
50
```

5. Recursive Functions

A function that calls itself is known as Recursive Function.

Eg:

factorial(3)=3*factorial(2)

```
=3*2*factorial(1)

=3*2*1*factorial(0)

=3*2*1*1 =6
```

factorial(n)= n*factorial(n-1)

The main advantages of recursive functions are:

1. We can reduce length of the code and improves readability.
2. We can solve complex problems very easily. For example, Towers of Hanoi, Ackerman's Problem etc.,

Q 1. Write a Python Function to find factorial of given number with recursion.

In [45]:

```
def factorial(n):
    if n==0:
        result=1
    else:
        result=n*factorial(n-1)
    return result

print("Factorial of 0 is :",factorial(0))
print("Factorial of 4 is :",factorial(4))
print("Factorial of 5 is :",factorial(5))
print("Factorial of 40 is :",factorial(40))
```

```
Factorial of 0 is : 1
Factorial of 4 is : 24
Factorial of 5 is : 120
Factorial of 40 is : 815915283247897734345611269596115894272000000000
```

6. Anonymous Functions

- Sometimes we can declare a function without any name,such type of nameless functions are called **anonymous functions** or **lambda functions**.
- The main purpose of anonymous function is **just for instant use(i.e., for one time usage)** (Eg: Alone train journey).

Normal Function:

We can define by using **def** keyword.

def squareIt(n):

```
return n*n
```

lambda Function:

We can define by using **lambda** keyword

```
lambda n:n*n
```

Syntax of lambda Function:

```
lambda argument_list : expression
```

Note:

- By using Lambda Functions we can write very concise code so that readability of the program will be improved.

Q. Write a program to create a lambda function to find square of given number.

In [46]:

```
s=lambda n:n*n
print("The Square of 4 is :",s(4))
print("The Square of 5 is :",s(5))
```

```
The Square of 4 is : 16
The Square of 5 is : 25
```

Q 2. Write a program to create a Lambda function to find sum of 2 given numbers.

In [48]:

```
s=lambda a,b:a+b
print("The Sum of 10,20 is:",s(10,20))
print("The Sum of 100,200 is:",s(100,200))
```

```
The Sum of 10,20 is: 30
The Sum of 100,200 is: 300
```

Q 3.Write a program to create a Lambda Function to find biggest of given values.

In [49]:

```
s=lambda a,b:a if a>b else b
print("The Biggest of 10,20 is:",s(10,20))
print("The Biggest of 100,200 is:",s(100,200))
```

```
The Biggest of 10,20 is: 20
The Biggest of 100,200 is: 200
```

Note:

- Lambda Function internally returns expression value and we are not required to write return statement explicitly.

Sometimes we can pass a function as argument to another function. In such cases lambda functions are best choice.

- We can use lambda functions very commonly with **filter(),map() and reduce() functions**,because these functions expect function as argument.

1. filter() function:

- We can use **filter()** function **to filter values from the given sequence based on some condition.**
- For example, we have 20 numbers and if we want to retrieve only even numbers from them.

Syntax:

```
filter(function,sequence)
```

Where,

- **function argument** is responsible to perform conditional check.
- **sequence** can be list or tuple or string.

Q 1. Program to filter only even numbers from the list by using filter() function.

Without lambda Function:

In [50]:

```
def isEven(x):
    if x%2==0:
        return True
    else:
        return False
l=[0,5,10,15,20,25,30]
l1=list(filter(isEven,l))
print(l1)              #[0,10,20,30]
```

```
[0, 10, 20, 30]
```

With lambda Function:

In [51]:

```
l=[0,5,10,15,20,25,30]
l1=list(filter(lambda x:x%2==0,l))
print(l1)                                #[0,10,20,30]
l2=list(filter(lambda x:x%2!=0,l))
print(l2)                                #[5,15,25]
```

```
[0, 10, 20, 30]
[5, 15, 25]
```

2. map() function:

- **For every element present in the given sequence,apply some functionality and generate new element with the required modification.** For this requirement we should go for **map()** function.

Syntax:

```
map(function,sequence)
```

The function can be applied on each element of sequence and generates new sequence.

Eg 1: For every element present in the list perform double and generate new list of doubles.

Without lambda

In [52]:

```
l=[1,2,3,4,5]
def doubleIt(x):
    return 2*x

l1=list(map(doubleIt,l))
print(l1)                #[2, 4, 6, 8, 10]
```

```
[2, 4, 6, 8, 10]
```

With lambda

In [53]:

```
l=[1,2,3,4,5]
l1=list(map(lambda x:2*x,l))
print(l1)                       #[2, 4, 6, 8, 10]
```

```
[2, 4, 6, 8, 10]
```

Eg 2: Find square of given numbers using map() function.

In [54]:

```
l=[1,2,3,4,5]
l1=list(map(lambda x:x*x,l))
print(l1)                 #[1, 4, 9, 16, 25]
```

```
[1, 4, 9, 16, 25]
```

We can apply map() function on multiple lists also.But make sure all list should have same length.

Syntax:

```
map(lambda x,y:x*y,l1,l2))


x is from l1 and y is from l2
```

Eg :

In [55]:

```
l1=[1,2,3,4]
l2=[2,3,4,5]
l3=list(map(lambda x,y:x*y,l1,l2))
print(l3)                   #[2, 6, 12, 20]
```

```
[2, 6, 12, 20]
```

In [1]:

```
l1=[1,2,3,4,5,6,7]     # The extra elements will be ignored
l2=[2,3,4,5]
l3=list(map(lambda x,y:x*y,l1,l2))
print(l3)                   #[2, 6, 12, 20]
```

```
[2, 6, 12, 20]
```

Date: 11-05-2020 Day 2

3. reduce() function:

> reduce() function **reduces sequence of elements into a single element by applying the specified function.**

Syntax:

```
reduce(function,sequence)
```

Note :

> reduce() function present in **functools module** and hence we should write import statement.

Eg 1:

In [56]:

```
from functools import *
l=[10,20,30,40,50]
result=reduce(lambda x,y:x+y,l)
print(result) # 150
```

```
150
```

In [9]:

```
from functools import *
l=sum([10,20,30,40,50])
# result=reduce(lambda x,y:x*y,l)
print(l) #150
```

```
150
```

Eg 2:

```
In [2]:

from functools import *
l=[10,20,30,40,50]
result=reduce(lambda x,y:x*y,l)
print(result) #12000000

12000000
```

Eg 3:

```
In [3]:

from functools import *
result=reduce(lambda x,y:x+y,range(1,101))
print(result)        #5050

5050
```

Note:

In Python every thing is treated as object. (Except keywords).

Even functions also internally treated as objects only.

Eg :

```
In [12]:

def f1():
    print("Hello")

print(f1)
print(id(f1))
print(id(print)) # print is also an object
print(id(id))    # id is also an object

<function f1 at 0x000001697E535598>
1552602584472
1552519043040
1552519042104
```

7. Function Aliasing:

For the existing function we can give another name, which is nothing but **function aliasing.**

Eg :

In [5]:

```
def wish(name):
    print("Good Morning:",name)

greeting = wish
print(id(wish))
print(id(greeting))

greeting('Karthi')
wish('Karthi')
```

```
1552601886504
1552601886504
Good Morning: Karthi
Good Morning: Karthi
```

Note:

> In the above example only one function is available but we can call that function by using either wish name or greeting name.

If we delete one name still we can access that function by using alias name.

Eg :

In [6]:

```
def wish(name):
    print("Good Morning:",name)

greeting=wish

greeting('Karthi')
wish('Karthi')

del wish
wish('Karthi') #NameError: name 'wish' is not defined
greeting('Saha')
```

```
Good Morning: Karthi
Good Morning: Karthi

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-f292fb57f669> in <module>
      8 
      9 del wish
---> 10 wish('Karthi') #NameError: name 'wish' is not defined
     11 greeting('Saha')

NameError: name 'wish' is not defined
```

In [14]:

```
def wish(name):
    print("Good Morning:",name)

greeting=wish
rgm = greeting
greeting('Karthi')
wish('Karthi')
rgm('Karthi')

del wish
#wish('Karthi') #NameError: name 'wish' is not defined
greeting('Saha')
```

```
Good Morning: Karthi
Good Morning: Karthi
Good Morning: Karthi
Good Morning: Saha
```

8. Nested Functions

We can declare a function inside another function, such type of functions are called Nested functions.

**Where we have this type of requirement?

```
    If a group of statements inside a function are repeatedly requires, then these
group of statements we will define as inner function and we can call this inner fu
nction whenever need arises.
```

Eg :

In [15]:

```
def outer():
    print("outer function started")
    def inner():
        print("inner function execution")                # It is function declaration
    print("outer function calling inner function")
    inner()                                               # It is a function call
outer()
#inner() ==>NameError: name 'inner' is not defined
```

```
outer function started
outer function calling inner function
inner function execution
```

```
In [16]:

def outer():
    print("outer function started")
    def inner():
        print("inner function execution")
    print("outer function calling inner function")
    inner()
outer()
inner() #NameError: name 'inner' is not defined

outer function started
outer function calling inner function
inner function execution

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-63fb5b235d22> in <module>
      6     inner()
      7 outer()
----> 8 inner() #NameError: name 'inner' is not defined

NameError: name 'inner' is not defined
```

In the above example **inner()** function is local to **outer()** function and hence it is not possible to call **inner()** function directly from outside of **outer()** function.

Another Example :

```
In [17]:

def f1():
    def inner(a,b):
        print('The Sum :',a+b)
        print('The Average :',(a+b)/2)
    inner(10,20)
    inner(20,30)
    inner(40,50)
    inner(100,200)
f1()

The Sum : 30
The Average : 15.0
The Sum : 50
The Average : 25.0
The Sum : 90
The Average : 45.0
The Sum : 300
The Average : 150.0
```

Is it possible to pass a function as an argument to another function?

Yes, a function can take another function as an argument.

For example,

filter(function,Sequence)

map(function,Sequence)

reduce(function,Sequence)

Note:

A function can return another function.

In [13]:

```
def outer():
    print("outer function started")
    def inner():
        print("inner function execution")
    print(id(inner))
    print("outer function returning inner function")
    return inner

f1=outer()       # f1 is pointingto inner function.
print(id(f1))
print(type(f1))
f1()             # Now directly 'inner()' function is calling
f1()
f1()
```

```
outer function started
1581911874824
outer function returning inner function
1581911874824
<class 'function'>
inner function execution
inner function execution
inner function execution
```

In [10]:

```
def outer():
    print("outer function started")
    def inner():
        print("inner function execution")
    print("outer function returning inner function")
    return inner()       # inner() is not returning anything, so you will get 'None'

f1=outer()       # f1 is pointingto inner function.
print(f1)
```

```
outer function started
outer function returning inner function
inner function execution
None
```

Q. What is the differenece between the following lines?

```
f1 = outer
```

```
f1 = outer()
```

In the first case for the **outer()** function we are providing another name **f1**(function aliasing).

But in the second case we calling **outer()** function,which returns **inner()** function. For that **inner()** function we are providing another name **'f1'.**

```
In [ ]:
```

Modules

Date: 15-05-2020 Day 1

1. Introduction

A group of functions, variables and classes saved to a file, which is nothing but **module.**

Every Python file (.py) acts as a module.

Eg:

In [1]:

```
x = 888
y = 999

def add(a,b):                               # Use Editplus editor
    print('The Sum : ',a+b)

def product(a,b):
    print('The produc :', a*b)
```

Let me save this code as **rgm.py**, and itself is a module.

rgm.py module contains two variables and 2 functions.

If we want to use members of module in our program then we should import that module.

Syntax of importing a module

```
import modulename
```

We can access members by using module name.

modulename.variable

modulename.function()

In [ ]:

```
import rgm
print(rgm.x)
rgm.add(10,20)       # Executed in Editplus editor
rgm.product(10,20)
```

Output

888

The Sum : 30

The Product : 200

Note:

- Whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently.This is avaialble at _ _ **pycache_ _** file, which is available at current working directory.

2.Advantages of Modules

1. Code Reusability
2. Readability improved
3. Maintainability improved

3. Renaming a module at the time of import (module aliasing):

We can create alias name for a module. This can be done as follows:

import rgm as r

Here, **rgm** is original module name and **r** is alias name. We can access members by using alias name** **r**.

Eg:

In [ ]:

```
import rgm as r
print(r.x)
r.add(10,20)                    # Executed in Editplus editor
r.product(10,20)
```

Output

888

The Sum : 30

The Product : 200

Eg :

In [ ]:

```
import rgm as r
print(rgm.x)
rgm.add(10,20)                    # Executed in Editplus editor
rgm.product(10,20)
```

Output

Traceback (most recent call last):

File "test.py", line 2, in

```
print(rgm.x)
```

NameError: name 'rgm' is not defined

Note:

Once we define alias name for a module, compulsory you should use alias name only. original names by default will be gone related to that particular file.

4. from ... import

- We can import particular members of module by using **from ... import**.
- The main advantage of this is we can access members directly without using module name.

Eg :

In [ ]:

```
from rgm import x,add
print(x)
add(10,20)                    # Executed in Editplus editor
product(10,20)
```

Output

888

The Sum : 30

Traceback (most recent call last):

File "test.py", line 4, in

```
product(10,20)
```

NameError: name 'product' is not defined

We can import all members of a module as follows

from rgm import

Eg :

In [ ]:

```
from rgm import *
print(x)
add(10,20)                    # Executed in Editplus editor
product(10,20)
```

Output

888

The Sum : 30

The Product : 200

5. member aliasing

Similar to module aliasing, member aliasing also possible in python. This can be done as follows:

from rgm import x as y,add as sum

print(y)

sum(10,20)

Note: Once we defined as alias name,we should use alias name only and we should not use original name.

Eg:

In [ ]:

```
from rgm import x as y
print(x)
```

Output

Traceback (most recent call last):

File "test.py", line 2, in

```
    print(x)
```

NameError: name 'x' is not defined

6. Various possibilties of import

import modulename

import module1,module2,module3

import module1 as m

import module1 as m1,module2 as m2,module3 from module import member

from module import member1,member2,memebr3

from module import memeber1 as x

from module import *

Date: 16-05-2020 Day 2

7. Reloading a Module

By default, module will be loaded only once eventhough we are importing multiple times.

Demo Program for module reloading:

Assume that we created a module named as **module1.py**.

In [ ]:

```
print('This is from module 1')
```

Now, we want to use **module1.py** in another module **test.py**.

In [ ]:

```
import module1
import module1
import module1
import module1
import module1
import module1
print('This is Test Module')    # Executed in Editplus editor
```

Output

This is from module 1

This is Test Module

In [ ]:

```
import module1
'''import module1
import module1
import module1
import module1
import module1'''

print('This is Test Module')            # Executed in Editplus editor
```

Output

This is from module 1

This is Test Module

In the above program test module will be loaded only once eventhough we are importing multiple times.

The problem in this approach is after loading a module if it is updated outside then updated version of **module1** is not available to our program.

In [ ]:

```
import time
import module1     # importing original version of module1
print("Program entering into sleeping state ")
time.sleep(30)   # during this time we want to modify something to module1
import module 1  # After 30 seconds we are importing module1, is it going to import updated
print("This is last line of program")                # Executed in Editplus editor
```

Output

This is from module 1

Program entering into sleeping mode

```
                                               # After 30 seconds
```

This is last line of program # Updated version is not available.

We can solve this problem by reloading module explicitly based on our requirement. We can reload by using **reload()** function of **imp** module.

import importlib

importlib.reload(module1)

In [ ]:

```
import time
from importlib import reload
import module1
print("program entering into sleeping state")
time.sleep(30)                                    # It is not mandatory
reload(module1)
print("This is last line of program")                # Executed in Editplus editor
```

Output

In [ ]:

```
This is from module 1                          # original version of module1


Program entering into sleeping mode
                                               # After 30 seconds

This is from updated module 1                  # during this time module1 is updated (i.e


This is last line of program                   # Updated version of module1 is now availa
```

Note: In the above program, everytime updated version of module1 will be available to our program

- **The main advantage of explicit module reloading is we can ensure that updated version is always available to our program.**

8. Finding members of module by using 'dir()' function

- Python provides inbuilt function **dir()** to list out all members of current module or a specified module.

dir() ===>To list out all members of current module

dir(moduleName)==>To list out all members of specified module

Eg 1: To display members of current module

In [1]:

```
x=10
y=20
def f1():
    print("Hello")
print(dir())            # To print all members of current module
```

```
['In', 'Out', '_', '__', '___', '__builtin__', '__builtins__', '__doc__', '_
_loader__', '__name__', '__package__', '__spec__', '_dh', '_i', '_i1', '_i
h', '_ii', '_iii', '_oh', 'exit', 'f1', 'get_ipython', 'quit', 'x', 'y']
```

Eg 2: To display members of particular module

Consider **rgm.py** module,

In [ ]:

```
x = 888
y = 999

def add(a,b):
    print('The Sum :',a+b)

def product(a,b):
    print('The Product :',a*b)
```

import **rgm** module in another module, called as **test.py**,

In [ ]:

```
import rgm
print(dir(rgm))
```

Output

['**builtins**', '**cached**', '**doc**', '**file**', '**loader**', '**name**', '**package**', '**spec**', 'add', 'product', 'x', 'y']

Note:

- For every module at the time of execution Python interpreter will add some special properties automatically for internal use.

Eg: _ *builtins ,_ _cached ,' doc ,_ _file , _ _loader , _ _name ,_ _package , _ _spec* _

Based on our requirement we can access these properties also in our program.

In [ ]:

```
print(__builtins__ )
print(__cached__ )
print(__doc__)
print(__file__)
print(__loader__)
print(__name__)
print(__package__)
print(__spec__)
```

Output

<module 'builtins' (built-in)>

None

None

test.py

<*frozen*importlib_external.SourceFileLoader object at 0x000001C8488D2640>

main

None

None

9. The Special variable '_ *name* _'

- For every Python program , a special variable _ *name*_ will be added internally. This variable stores information regarding whether the program is executed as an individual program or as a module.
- If the program executed as an individual program then the value of this variable is _ *main*_.
- If the program executed as a module from some other program then the value of this variable is the name of module where it is defined.
- Hence by using this _ *name*_ variable we can identify whether the program executed directly or as a module.

Demo program:

module1.py

In [2]:

```python
def f1():
    if __name__=='__main__':
        print("The code executed directly as a program")
        print("The value of __name__:",__name__)
    else:
        print("The code executed indirectly as a module from some other program")
        print("The value of __name__:",__name__)
f1()
```

```
The code executed directly as a program
The value of __name__: __main__
```

test.py

In [ ]:

```python
import module1
print("From test we are executing module f1()")
module1.f1()                    # Executed in Editplus editor
```

Output

The code executed indirectly as a module from some other program

The value of **name**: module1

From test we are executing module f1()

The code executed indirectly as a module from some other program

The value of **name**: module1

10.Working with math module

- Python provides inbuilt module math.
- This module defines several functions which can be used for mathematical operations.

The main important functions are:

1. sqrt(x)
2. ceil(x)
3. floor(x)
4. fabs(x)
5. log(x)
6. sin(x)
7. tan(x)

Eg :

In [6]:

```
from math import *
print(sqrt(4))
print(ceil(10.1))
print(floor(10.1))
print(fabs(-10.6))       # Ignore sign, consider only value    fabs ---> float absolute fu
print(fabs(10.6))        # Ignore sign, consider only value
```

```
2.0
11
10
10.6
10.6
```

Note:

- We can find help for any module by using **help()** function.

Eg:

In [5]:

```python
import math
help(math)
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x but the sign
of y.

        On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

    cosh(x, /)
        Return the hyperbolic cosine of x.

    degrees(x, /)
        Convert angle x from radians to degrees.

    erf(x, /)
        Error function at x.
    erfc(x, /)
        Complementary error function at x.

    exp(x, /)
        Return e raised to the power of x.

    expm1(x, /)
        Return exp(x)-1.

        This function avoids the loss of precision involved in the direct ev
aluation of exp(x)-1 for small x.

    fabs(x, /)
        Return the absolute value of the float x.

    factorial(x, /)
        Find x!.

        Raise a ValueError if x is negative or non-integral.

    floor(x, /)
        Return the floor of x as an Integral.

        This is the largest integer <= x.

    fmod(x, y, /)
        Return fmod(x, y), according to platform C.

        x % y may differ.

    frexp(x, /)
        Return the mantissa and exponent of x, as pair (m, e).

        m is a float and e is an int, such that x = m * 2.**e.
        If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

    fsum(seq, /)
        Return an accurate floating point sum of values in the iterable seq.

        Assumes IEEE-754 floating point arithmetic.

    gamma(x, /)
        Gamma function at x.

    gcd(x, y, /)
        greatest common divisor of x and y

    hypot(x, y, /)
        Return the Euclidean distance, sqrt(x*x + y*y).

    isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
        Determine whether two floating point numbers are close in value.

          rel_tol
            maximum difference for being considered "close", relative to the
            magnitude of the input values
          abs_tol
            maximum difference for being considered "close", regardless of t
he
            magnitude of the input values
        Return True if a is close in value to b, and False otherwise.

        For the values to be considered close, the difference between them
        must be smaller than at least one of the tolerances.

        -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
        is, NaN is not close to anything, even itself.  inf and -inf are
        only close to themselves.

    isfinite(x, /)
        Return True if x is neither an infinity nor a NaN, and False otherwi
se.

    isinf(x, /)
        Return True if x is a positive or negative infinity, and False other
wise.

    isnan(x, /)
        Return True if x is a NaN (not a number), and False otherwise.

    ldexp(x, i, /)
        Return x * (2**i).

        This is essentially the inverse of frexp().

    lgamma(x, /)
        Natural logarithm of absolute value of Gamma function at x.

    log(...)
        log(x, [base=math.e])
        Return the logarithm of x to the given base.

        If the base not specified, returns the natural logarithm (base e) of
x.

    log10(x, /)
        Return the base 10 logarithm of x.

    log1p(x, /)
        Return the natural logarithm of 1+x (base e).

        The result is computed in a way which is accurate for x near zero.

    log2(x, /)
        Return the base 2 logarithm of x.

    modf(x, /)
        Return the fractional and integer parts of x.

        Both results carry the sign of x and are floats.

    pow(x, y, /)
        Return x**y (x to the power of y).

    radians(x, /)
        Convert angle x from degrees to radians.

    remainder(x, y, /)
        Difference between x and the closest integer multiple of y.
```

```
            Return x - n*y where n*y is the closest integer multiple of y.
            In the case where x is exactly halfway between two multiples of
            y, the nearest even value of n is used. The result is always exact.

        sin(x, /)
            Return the sine of x (measured in radians).

        sinh(x, /)
            Return the hyperbolic sine of x.

        sqrt(x, /)
            Return the square root of x.

        tan(x, /)
            Return the tangent of x (measured in radians).

        tanh(x, /)
            Return the hyperbolic tangent of x.

        trunc(x, /)
            Truncates the Real x to the nearest Integral toward 0.

            Uses the __trunc__ magic method.

    DATA
        e = 2.718281828459045
        inf = inf
        nan = nan
        pi = 3.141592653589793
        tau = 6.283185307179586

    FILE
        (built-in)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-9ef53aca3b77> in <module>
      1 import math
      2 help(math)
----> 3 a

NameError: name 'a' is not defined
```

11. Working with random module

- This module defines several functions to generate random numbers.
- We can use these functions while developing games,in cryptography and to generate random numbers on fly for authentication. (For example, OTPs).

1. random() function:

This function always generate some float value between 0 and 1 (not inclusive).

0 < x < 1

Eg :

In [14]:

```
from random import random        # from random import *
#import random
for i in range(10):
    print(random())
```

```
0.5298687885975731
0.1959883975809048
0.7856765556766167
0.621345638337082
0.0017397062733900404
0.053682389130991326
0.6870134702620266
0.012400503004914687
0.9615995552319757
0.5501802331038093
```

2. randint() function:

This function is used to generate random integers beween two given numbers (inclusive).

Eg:

In [22]:

```
from random import *
for i in range(10):
    print(randint(1,10)) # generate random int value between 1 and 100(inclusive)
```

```
4
5
3
9
4
5
6
10
6
6
```

3. uniform():

- It returns random float values between 2 given numbers (not inclusive).

Eg:

In [17]:

```
from random import *
for i in range(10):
    print(uniform(1,10))
```

```
4.219659431824531
5.058564769299971
7.5266485757042485
4.190749282077976
3.8464174832123033
7.043381178043777
9.30504048393276
4.41712781745301
3.9935721537809465
8.274960415539518
```

Note:

- **random()** ===>in between 0 and 1 (not inclusive) ===> float
- **randint(x,y)** ==>in between x and y (inclusive) ===> int
- **uniform(x,y)** ==> in between x and y (not inclusive) ===> float

4. randrange([start],stop,[step])

- returns a random number from range **start<= x < stop**
- start argument is optional and default value is 0
- step argument is optional and default value is 1

For example,

- randrange(10)-->generates a number from 0 to 9
- randrange(1,11)-->generates a number from 1 to 10
- randrange(1,11,2)-->generates a number from 1,3,5,7,9

Eg :

In [18]:

```
from random import *
for i in range(10):
    print(randrange(10))
```

```
5
8
6
1
2
6
2
9
6
1
```

Eg 2:

In [20]:

```
from random import *
for i in range(10):
    print(randrange(1,11))
```

```
8
5
5
6
6
8
3
10
10
4
```

Eg 3:

In [24]:

```
from random import *
for i in range(10):
    print(randrange(1,11,2))
```

```
1
9
1
3
7
7
7
7
1
5
```

5. choice() function:

- It won't return random number.

- It will return a random object from the given list or tuple.
- Always the argument for this function is any indexable sequence. (i.e., Set is not supported)

Eg :

In [25]:

```
from random import *
list=["Sunny","Bunny","Chinny","Vinny","pinny"]
for i in range(10):
    print(choice(list))
```

```
Chinny
Chinny
pinny
Sunny
Chinny
Vinny
Chinny
Bunny
Sunny
Bunny
```

In [26]:

```
from random import *
list=("Sunny","Bunny","Chinny","Vinny","pinny")
for i in range(10):
    print(choice(list))
```

```
Bunny
Bunny
pinny
Bunny
Sunny
Vinny
Sunny
Chinny
Bunny
Chinny
```

In [27]:

```
from random import *
list={"Sunny","Bunny","Chinny","Vinny","pinny"}
for i in range(10):                    #Set object is not support indexing
    print(choice(list))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-207be8b00172> in <module>
      2 list={"Sunny","Bunny","Chinny","Vinny","pinny"}
      3 for i in range(10):
----> 4     print(choice(list))

~\Anaconda3\lib\random.py in choice(self, seq)
    260         except ValueError:
    261             raise IndexError('Cannot choose from an empty sequence')
from None
--> 262         return seq[i]
    263 
    264     def shuffle(self, x, random=None):

TypeError: 'set' object is not subscriptable
```

In [28]:

```
from random import *
list=["Sunny","Bunny","Chinny","Vinny","pinny"]
for i in range(10):
    print(choice('karthi'))
```

```
r
a
t
t
h
r
a
a
h
r
```

Date: 17-05-2020 Day-3

Example programs

Q 1. Write a Python program to generate a six digit random number as One Time Password (OTP).

Way 1

In [3]:

```
from random import *
for i in range(10):
    print(randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),sep
```

```
792314
871530
780895
157332
796357
502414
826712
785971
309040
015443
```

In [16]:

```
from random import *
for i in range(10):
    print(randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),sep
```

```
161446
197083
758751
996540
851466
666187
700286
719132
372328
136409
```

Way 2

In [7]:

```
from random import *
for i in range(10):
    for x in range(6):
        print(randint(0,9),end='')      # Correct version
    print()
```

```
899095
829710
031807
656661
053638
290606
805589
240823
108752
356289
```

In [15]:

```
from random import *
for i in range(10):
    for x in range(6):
        print(randint(0,9),end='')      # Correct version
    print()
```

```
738483
300653
071471
666878
872723
287566
540940
676690
328498
360878
```

Way 3

In [8]:

```
from random import *
for i in range(10):
    print(randint(000000,999999),sep='')
```

```
962601
889203
384254
393560
633705
103495
985154
107436
149361
240032
```

In [9]:

```
from random import *
for i in range(10):
    print(randint(000000,999999),sep='')
```

```
682096
266601
510098
805968
203852
838713
744178
568925
830862
538721
```

In [10]:

```
from random import *
for i in range(10):
    print(randint(000000,999999),sep='')     # Some times it may give wrong output also, bec
```

```
347289
624504
70817
916648
627988
848795
66449
415174
989153
872602
```

Way 4

In [11]:

```
from random import *
for i in range(10):
    print(randint(100000,999999),sep='')
```

```
869252
909810
596249
585590
346792
498318
326801
788542
835508
960551
```

In [12]:

```
from random import *
for i in range(10):
    print(randint(100000,999999),sep='')
```

```
572571
692732
462218
234897
442399
480218
278091
182737
396578
244616
```

In [13]:

```
from random import *
for i in range(10):
    print(randint(100000,999999),sep='')
```

```
501568
282792
596089
356996
377236
717111
145663
941932
903885
904369
```

In [14]:

```
from random import *
for i in range(10):                              # This code is working properly, ex
    print(randint(100000,999999),sep='')
```

```
749026
924598
217585
709813
503600
640488
464279
508086
521877
834829
```

Flaw: It won't generate OTP which start from 0.

Q 2. Write aPython program to generate a random password of 6 length.

Within the OTP --

- 1,3,5 positions are alphabets.
- 2,4,6 positions are digits.

Way 1

```
In [20]:
```

```
from random import *
for i in range(10):
       print(chr(randint(65,90)),randint(0,9),chr(randint(65,90)),randint(0,9),chr(randint(
```

```
R2I6T5
F4S6I6
W5B8K5
M3U7L6
C3A6M3
O5L7W3
T5W7X4
S3I7W1
L5C9A9
J9H4W4
```

```
In [21]:
```

```
from random import *
for i in range(10):
       print(chr(randint(65,90)),randint(0,9),chr(randint(65,90)),randint(0,9),chr(randint(
```

```
B9D7J9
X0S2M0
S6R7Y6
B2B1B0
O4G4I0
J6U2P2
L6C6B3
C6C5K9
M1Q7N3
I3Y2F7
```

Way 2:

```
In [27]:
```

```
from random import *
for i in range(10):
    for x in range(1,7):
        if x%2 == 1:
            print(chr(randint(65,90)),end='')
        else:
            print(randint(0,9),end='')
    print()
```

```
P3H1U3
V1Q1A7
R4Z2G9
I3Y4H8
B8K7V9
N4X7S0
R5L3S2
O5A3Y8
M0D7J7
H7V2Z0
```

In [28]:

```
from random import *
for i in range(10):
    for x in range(1,7):
        if x%2 == 1:
            print(chr(randint(65,90)),end='')
        else:
            print(randint(0,9),end='')
    print()
```

```
I4O0C6
E1K2K9
K9J6X8
W9I3Z9
Z0A1X9
T6S9M7
L6E8W7
A1X8J2
A7M6A3
R4R1E7
```

In [ ]:

```

```

Regular Expressions

Date: 20-05-2020

Introduction

If we want to represent a group of Strings according to a particular format/pattern then we should go for Regular Expressions. i.e., **Regualr Expressions is a declarative mechanism to represent a group of Strings accroding to particular format/pattern.**

Eg 1: We can write a regular expression to represent all mobile numbers. (i.e., All mobile numbers having a particular format i.e., exactly 10 numbers only)

Eg 2: We can write a regular expression to represent all mail ids.

Eg 3: We can write a regular expression to represent all java/python/C identifiers.

Note: Regular Expressions is language independent concept.

The main important application areas of Regular Expressions are as follows:

1. To develop validation frameworks/validation logic. For example, mail id validation, mobile number validation etc.
2. To develop Pattern matching applications (ctrl-f in windows, grep in UNIX etc).
3. To develop Translators like compilers, interpreters etc. In compiler design, Lexical analysis phase is internally implemented using Regular expressions only.
4. To develop digital circuits. For example, Binary Incrementor, Binary adder, Binary subtractor etc.
5. To develop communication protocols like TCP/IP, UDP etc. (Protocol means set of rules, to follow the rules during communication, we use regular expressions).

NOTE:

3 Mantras, to become no.1 software Engineer:

1. Ctrl + f

2. Ctrl + c

3. Ctrl + v

Then you may get one doubt that, **why we need to learn all these technologies???**

Ans: To get the Job all several things are required, To do the job nothing is

required.

Real Fact: Almost 80% - 90% of the software engineers work depends on the above 3

mantras only.

re module:

We can develop Regular Expression Based applications by using python module known as **re**.

This module contains several in-built functions to use Regular Expressions very easily in our applications.

1. compile():

- re module contains compile() function to compile a pattern into RegexObject.

For example, if you want to find the pattern 'python' in the given string, first you need to convert this pattern into RegexObject form.

In [1]:

```
import re
pattern = re.compile("python")
print(type(pattern))
```

```
<class 're.Pattern'>
```

2. finditer():

- Returns an Iterator object which yields Match object for every Match.

In [ ]:

```
matcher = pattern.finditer("Learning python is very easy...")
```

On Match object we can call the following methods.

1. start() ==> Returns start index of the match
2. end() ==> Returns end+1 index of the match
3. group() ==> Returns the matched string

Eg: Write a python program to find whether the given pattern is available in the given string or not?

In [3]:

```
import re
count=0
pattern=re.compile("python")
matcher=pattern.finditer("Learning python is very easy...") # We are searching for a word i
for match in matcher:
    count+=1
    print(match.start(),"...",match.end(),"...",match.group())
print("The number of occurrences: ",count)
```

```
9 ... 15 ... python
The number of occurrences:  1
```

Note: More Simplified form

- We can pass pattern directly as argument to finditer() function.

In [4]:

```
import re
count=0
matcher=re.finditer("ab","abaababa") # We are searching for a word in one sentence (i.e., p
for match in matcher:
    count+=1
    print(match.start(),"...",match.end(),"...",match.group())
print("The number of occurrences: ",count)
```

```
0 ... 2 ... ab
3 ... 5 ... ab
5 ... 7 ... ab
The number of occurrences:  3
```

In [10]:

```
import re
count=0
matcher=re.finditer("ba","abaababa") # We are searching for a word in one sentence (i.e., p
for match in matcher:
    count+=1
    print(match.start(),"...",match.end(),"...",match.group())
print("The number of occurrences: ",count)
```

```
1 ... 3 ... ba
4 ... 6 ... ba
6 ... 8 ... ba
The number of occurrences:  3
```

In [13]:

```
import re
count=0
matcher=re.finditer("bb","abaababa") # We are searching for a word in one sentence (i.e., p
for match in matcher:
    count+=1
    print("start:{},end:{},group:{}".format(match.start(),match.end(),match.group()))
print("The number of occurrences: ",count)
```

```
The number of occurrences:  0
```

In [14]:

```
import re
count=0
matcher=re.finditer("ab","abaababa") # We are searching for a word in one sentence (i.e., p
for match in matcher:
    count+=1
    print("start:{},end:{},group:{}".format(match.start(),match.end(),match.group()))
print("The number of occurrences: ",count)
```

```
start:0,end:2,group:ab
start:3,end:5,group:ab
start:5,end:7,group:ab
The number of occurrences:  3
```

In [15]:

```
import re
count=0
matcher=re.finditer("ab","abababa") # We are searching for a word in one sentence (i.e., pa
for match in matcher:
    count+=1
    print("start:{},end:{},group:{}".format(match.start(),match.end(),match.group()))
print("The number of occurrences: ",count)
```

```
start:0,end:2,group:ab
start:2,end:4,group:ab
start:4,end:6,group:ab
The number of occurrences:  3
```

Date: 21-05-2020 Day 2

Character classes:

We can use **character classes** to search a group of characters.

1. [abc]===>Either a or b or c
2. [^abc] ===>Except a and b and c
3. [a-z]==>Any Lower case alphabet symbol
4. [A-Z]===>Any upper case alphabet symbol
5. [a-zA-Z]==>Any alphabet symbol

6. [0-9] Any digit from 0 to 9

7. [a-zA-Z0-9]==>Any alphanumeric character

8. [^a-zA-Z0-9]==>Except alphanumeric characters(Special Characters)

In [8]:

```
import re
matcher=re.finditer("[abc]","a7b@k9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
2 ...... b
```

In [9]:

```
import re
matcher=re.finditer("[^abc]","a7b@k9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
1 ...... 7
3 ...... @
4 ...... k
5 ...... 9
6 ...... z
```

In [1]:

```
import re
matcher=re.finditer("[a-z]","a7b@k9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
2 ...... b
4 ...... k
6 ...... z
```

In [3]:

```
import re
matcher=re.finditer("[0-9]","a7b@k9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
1 ...... 7
5 ...... 9
```

In [4]:

```
import re
matcher=re.finditer("[A-Z]","a7b@k9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

In [5]:

```
import re
matcher=re.finditer("[a-zA-Z]","a7b@k9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
2 ...... b
4 ...... k
6 ...... z
```

In [6]:

```
import re
matcher=re.finditer("[a-zA-Z0-9]","a7b@k9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
1 ...... 7
2 ...... b
4 ...... k
5 ...... 9
6 ...... z
```

In [7]:

```
import re
matcher=re.finditer("[^a-zA-Z0-9]","a7b@k9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
3 ...... @
```

In [8]:

```
import re
matcher=re.finditer("[abc]","abcabc")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
1 ...... b
2 ...... c
3 ...... a
4 ...... b
5 ...... c
```

Pre defined Character classes

\s ==> Space character

\S ==> Any character except space character

\d ==> Any digit from 0 to 9

\D ==> Any character except digit

\w ==> Any word character [a-zA-Z0-9]

\W ==> Any character except word character (only Special Characters includes)

. ==> Any character including special characters

In [9]:

```
import re
matcher=re.finditer("\s","a7b k@9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
3 ......
```

In [10]:

```
import re
matcher=re.finditer("\S","a7b k@9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
1 ...... 7
2 ...... b
4 ...... k
5 ...... @
6 ...... 9
7 ...... z
```

In [11]:

```
import re
matcher=re.finditer("\d","a7b k@9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
1 ...... 7
6 ...... 9
```

In [12]:

```
import re
matcher=re.finditer("\D","a7b k@9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
2 ...... b
3 ......
4 ...... k
5 ...... @
7 ...... z
```

In [13]:

```
import re
matcher=re.finditer("\w","a7b k@9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
1 ...... 7
2 ...... b
4 ...... k
6 ...... 9
7 ...... z
```

In [14]:

```
import re
matcher=re.finditer("\W","a7b k@9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
3 ......  
5 ...... @
```

In [15]:

```
import re
matcher=re.finditer(".","a7b k@9z")
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
1 ...... 7
2 ...... b
3 ......  
4 ...... k
5 ...... @
6 ...... 9
7 ...... z
```

Qunatifiers

We can use **quantifiers** to specify the number of occurrences to match.

a ==> Exactly one 'a'

a+ ==> Atleast one 'a'

a* ==> Any number of a's including zero number

a? ==> Atmost one 'a', i.e., either zero number or one number

a{m} ==> Exactly m number of a's

a{m,n} ==> Minimum m number of a's and Maximum n number of a's

In [17]:

```
import re
matcher=re.finditer("a","abaabaaab")                        # Exactly one 'a'
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
2 ...... a
3 ...... a
5 ...... a
6 ...... a
7 ...... a
```

In [18]:

```
import re
matcher=re.finditer("a+","abaabaaab")                       # Atleast one 'a'
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
2 ...... aa
5 ...... aaa
```

In [19]:

```
import re
matcher=re.finditer("a*","abaabaaab")                        # Any no.of 'a's including zero
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
1 ......
2 ...... aa
4 ......
5 ...... aaa
8 ......
9 ......
```

In [20]:

```
import re
matcher=re.finditer("a?","abaabaaab")                           # Atmost one 'a', i.e., either
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
1 ......
2 ...... a
3 ...... a
4 ......
5 ...... a
6 ...... a
7 ...... a
8 ......
9 ......
```

In [21]:

```
import re
matcher=re.finditer("a{3}","abaabaaab")                    # Exactly '3' number of 'a's
for match in matcher:
    print(match.start(),"......",match.group())
```

```
5 ...... aaa
```

In [25]:

```
import re
matcher=re.finditer("a{2,4}","abaabaaab")                    # Minimum '2' 'a's and Maximum
for match in matcher:
    print(match.start(),"......",match.group())
```

```
2 ...... aa
5 ...... aaa
```

In [26]:

```
import re
matcher=re.finditer("a{2,2}","abaabaaab")                    # Minimum '2' 'a's and Maximum
for match in matcher:
    print(match.start(),"......",match.group())
```

```
2 ...... aa
5 ...... aa
```

In [27]:

```
import re
matcher=re.finditer("a{1,4}","abaabaaab")                    # Minimum '1' 'a' and Maximum
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
2 ...... aa
5 ...... aaa
```

In [28]:

```
import re
matcher=re.finditer("a{2}a*","abaabaaab")                    # Exactly'2' 'a's followed by
for match in matcher:
    print(match.start(),"......",match.group())
```

```
2 ...... aa
5 ...... aaa
```

In [29]:

```
import re
matcher=re.finditer("[^a]","abaabaaab")                    # Except 'a'
for match in matcher:
    print(match.start(),"......",match.group())
```

```
1 ...... b
4 ...... b
8 ...... b
```

Note:

^x ==> It will check whether target string starts with x or not

x$ ==> It will check whether target string ends with x or not

In [30]:

```
import re
matcher=re.finditer("^a","abaabaaab")                    # Whether the given string starts
for match in matcher:
    print(match.start(),"......",match.group())
```

```
0 ...... a
```

In [31]:

```
import re
matcher=re.finditer("a$","abaabaaab")                    # Whether the given string  ends w
for match in matcher:
    print(match.start(),"......",match.group())
```

In [32]:

```
import re
matcher=re.finditer("b$","abaabaaab")                    # Whether the given string  ends w
for match in matcher:
    print(match.start(),"......",match.group())
```

```
8 ...... b
```

Important functions of 're' module

1. match()
2. fullmatch()
3. search()
4. findall()
5. finditer()
6. sub()
7. subn()

8. split()

9. compile()

1. match():

- We can use match function to check the given pattern at beginning of target string or not.
- If the match is available then we will get Match object, otherwise we will get None.

In [34]:

```
import re
s=input("Enter pattern to check: ")

m=re.match(s,"abcabdefg")                        # match() function

if m!= None:
    print("Match is available at the beginning of the String")
    print("Start Index:",m.start(), "and End Index:",m.end())
else:
    print("Match is not available at the beginning of the String")
```

```
Enter pattern to check: abc
Match is available at the beginning of the String
Start Index: 0 and End Index: 3
```

In [35]:

```
import re
s=input("Enter pattern to check: ")

m=re.match(s,"abcabdefg")                        # match() function

if m!= None:
    print("Match is available at the beginning of the String")
    print("Start Index:",m.start(), "and End Index:",m.end())
else:
    print("Match is not available at the beginning of the String")
```

```
Enter pattern to check: rgm
Match is not available at the beginning of the String
```

2. fullmatch():

- We can use fullmatch() function to match a pattern to all of target string. i.e., complete string should be matched according to given pattern.
- If complete string matched then this function returns Match object otherwise it returns None.

In [36]:

```
import re
s=input("Enter pattern to check: ")

m=re.fullmatch(s,"ababab")

if m!= None:
    print("Full String Matched")
else:
    print("Full String not Matched")
```

```
Enter pattern to check: ab
Full String not Matched
```

In [37]:

```
import re
s=input("Enter pattern to check: ")

m=re.fullmatch(s,"ababab")

if m!= None:
    print("Full String Matched")
else:
    print("Full String not Matched")
```

```
Enter pattern to check: abababa
Full String not Matched
```

In [38]:

```
import re
s=input("Enter pattern to check: ")

m=re.fullmatch(s,"ababab")

if m!= None:
    print("Full String Matched")
else:
    print("Full String not Matched")
```

```
Enter pattern to check: ababab
Full String Matched
```

3. search():

- We can use search() function to search the given pattern in the target string.
- If the match is available then it returns the Match object which represents first occurrence of the match.
- If the match is not available then it returns None.

In [39]:

```
import re
s=input("Enter pattern to check: ")

m=re.search(s,"abaabaaab")

if m!= None:
    print("Match is available")
    print("First Occurrence of match with start index:",m.start(),"and end index:",m.end())
else:
    print("Match is not available")
```

```
Enter pattern to check: aa
Match is available
First Occurrence of match with start index: 2 and end index: 4
```

In [40]:

```
import re
s=input("Enter pattern to check: ")

m=re.search(s,"abaabaaab")

if m!= None:
    print("Match is available")
    print("First Occurrence of match with start index:",m.start(),"and end index:",m.end())
else:
    print("Match is not available")
```

```
Enter pattern to check: bb
Match is not available
```

4. findall():

- This function is used to find all occurrences of the match.
- This function returns a list object which contains all occurrences.

In [41]:

```
import re
l=re.findall("[0-9]","a7b9c5kz")
print(l)
```

```
['7', '9', '5']
```

5. finditer():

- It returns the iterator yielding a match object for each match.
- On each match object we can call start(), end() and group() functions.

In [42]:

```
import re
itr=re.finditer("[a-z]","a7b9c5k8z")
for m in itr:
    print(m.start(),"...",m.end(),"...",m.group())
```

```
0 ... 1 ... a
2 ... 3 ... b
4 ... 5 ... c
6 ... 7 ... k
8 ... 9 ... z
```

In [48]:

```
import re
itr=re.finditer("\d","a7b9c5k8z")
for m in itr:
    print(type(m))
    print(m.start(),"...",m.end(),"...",m.group())
```

```
<class 're.Match'>
1 ... 2 ... 7
<class 're.Match'>
3 ... 4 ... 9
<class 're.Match'>
5 ... 6 ... 5
<class 're.Match'>
7 ... 8 ... 8
```

6. sub():

- sub means substitution or replacement.

re.sub(regex,replacement,targetstring)

- In the target string every matched pattern will be replaced with provided replacement.

In [43]:

```
import re
s=re.sub("[a-z]","#","a7b9c5k8z")
print(s)
```

```
#7#9#5#8#
```

Here, Every alphabet symbol is replaced with # symbol.

In [49]:

```
import re
s=re.sub("\d","#","a7b9c5k8z")
print(s)
```

```
a#b#c#k#z
```

Here, Every digit is replaced with # symbol.

7. subn():

- It is exactly same as sub except it can also returns the number of replacements.
- This function returns a tuple where first element is result string and second element is number of replacements.

(resultstring, number of replacements)

In [44]:

```
import re
t=re.subn("[a-z]","#","a7b9c5k8z")
print(t)
print("The Result String:",t[0])
print("The number of replacements:",t[1])
```

```
('#7#9#5#8#', 5)
The Result String: #7#9#5#8#
The number of replacements: 5
```

8. split():

- If we want to split the given target string according to a particular pattern then we should go for split() function.
- This function returns list of all tokens.

In [45]:

```
import re
l=re.split(",","sunny,bunny,chinny,vinny,pinny")
print(l)
for t in l:
    print(t)
```

```
['sunny', 'bunny', 'chinny', 'vinny', 'pinny']
sunny
bunny
chinny
vinny
pinny
```

In [46]:

```
import re
l=re.split("\.","www.rgmcet.edu.in")
for t in l:
    print(t)
```

```
www
rgmcet
edu
in
```

In [50]:

```
import re
l=re.split("[.]","www.rgmcet.edu.in")
for t in l:
    print(t)
```

```
www
rgmcet
edu
in
```

Two special symbols used in Regular Expressions

1. ^ symbol:

We can use **^** symbol to check whether the given target string starts with our provided pattern or not.

Eg:

res=re.search("^Learn",s)

- if the target string starts with Learn then it will return Match object,otherwise returns None.

In [52]:

```
import re
s="Learning Python is Very Easy"
res=re.search("^Learn",s)
if res != None:
    print("Target String starts with Learn")
else:
    print("Target String Not starts with Learn")
```

```
Target String starts with Learn
```

In [53]:

```
import re
s="Learning Python is Very Easy"
res=re.search("^Learns",s)
if res != None:
    print("Target String starts with Learn")
else:
    print("Target String Not starts with Learn")
```

```
Target String Not starts with Learn
```

2. '$' symbol:

We can use $ symbol to check whether the given target string ends with our provided pattern or not.

Eg:

res=re.search("Easy$",s)

- If the target string ends with Easy then it will return Match object,otherwise returns None.

In [54]:

```
import re
s="Learning Python is Very Easy"
res=re.search("Easy$",s)
if res != None:
    print("Target String ends with Easy")
else:
    print("Target String Not ends with Easy")
```

```
Target String ends with Easy
```

In [55]:

```
import re
s="Learning Python is Very Easy"
res=re.search("easy$",s)
if res != None:
    print("Target String ends with Easy")
else:
    print("Target String Not ends with Easy")
```

```
Target String Not ends with Easy
```

Note:

- If we want to ignore case then we have to pass 3rd argument re.IGNORECASE for search() function.

Eg:

res = re.search("easy$",s,re.IGNORECASE)

In [56]:

```
import re
s="Learning Python is Very Easy"
res=re.search("easy$",s,re.IGNORECASE)
if res != None:
    print("Target String ends with Easy")
else:
    print("Target String Not ends with Easy")
```

```
Target String ends with Easy
```

In [57]:

```
import re
s="Learning Python is Very Easy"
res=re.search("Easys$",s)
if res != None:
    print("Target String ends with Easy")
else:
    print("Target String Not ends with Easy")
```

```
Target String Not ends with Easy
```

Date: 22-05-2020 Day 3

Example applications using Regular expressions

App 1: Write a Regular Expression to represent all Yava language (My own language) identifiers.

Rules:

1. The allowed characters are a-z,A-Z,0-9,#.
2. The first character should be a lower case alphabet symbol from a to k.
3. The second character should be a digit divisible by 3.
4. The length of identifier should be atleast 2.

Regular Expression

[a-k][3069][a-zA-Z0-9#]*

Write a python program to check whether the given string is Yava language identifier or not?

In [7]:

```
import re
s = input('Enter Identifier to validate :')
m = re.fullmatch('[a-k][3069][a-zA-Z0-9#]*',s)
if m!= None:
    print(s,'is valid Yava Identifier')
else:
    print(s,'is not Yava Identifier')
```

```
Enter Identifier to validate :a3
a3 is valid Yava Identifier
```

In [8]:

```
import re
s = input('Enter Identifier to validate :')
m = re.fullmatch('[a-k][3069][a-zA-Z0-9#]*',s)
if m!= None:
    print(s,'is valid Yava Identifier')
else:
    print(s,'is not Yava Identifier')
```

```
Enter Identifier to validate :z3k5
z3k5 is not Yava Identifier
```

In [9]:

```
import re
s = input('Enter Identifier to validate :')
m = re.fullmatch('[a-k][3069][a-zA-Z0-9#]*',s)
if m!= None:
    print(s,'is valid Yava Identifier')
else:
    print(s,'is not Yava Identifier')
```

```
Enter Identifier to validate :a9@
a9@ is not Yava Identifier
```

In [10]:

```
import re
s = input('Enter Identifier to validate :')
m = re.fullmatch('[a-k][3069][a-zA-Z0-9#]*',s)
if m!= None:
    print(s,'is valid Yava Identifier')
else:
    print(s,'is not Yava Identifier')
```

```
Enter Identifier to validate :f3jkgjidu
f3jkgjidu is valid Yava Identifier
```

In [11]:

```
import re
s = input('Enter Identifier to validate :')
m = re.fullmatch('[a-k][3069][a-zA-Z0-9#]*',s)
if m!= None:
    print(s,'is valid Yava Identifier')
else:
    print(s,'is not Yava Identifier')
```

```
Enter Identifier to validate :a6kk9z##
a6kk9z## is valid Yava Identifier
```

App 2: Write a Regular Expression to represent all 10 digit mobile numbers.

Rules:

1. Every number should contains exactly 10 digits.
2. The first digit should be 7 or 8 or 9

Regular Expression

[7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]

or

[7-9][0-9]{9}

or

[7-9]\d{9}

Write a Python Program to check whether the given number is valid mobile number or not?

In [13]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[7-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :9885768283
9885768283 is valid Mobile number
```

In [15]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[7-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :6754876589
6754876589 is not valid Mobile number
```

In [16]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[7-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :898989
898989 is not valid Mobile number
```

**App 3: Write a regular expression to find the valid mobile number based on the following rules.

The mobile number may contain 10 digit or 11 digit or 12 digit or 13 digit also:

10 : 6 to 9, 9 digits ===> [6-9][0-9]{9}

11: The first digit is 0 ===>0[[6-9][0-9]{9}

12: The first 2 digits should 91 ===>[9][1][6-9][0-9]{9}

13: The first 3 digits should be +91 ===>[+][9][1][6-9][0-9]{9}

Write a Python Program to check whether the given number is valid mobile number or not?

In [17]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[6-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :6098236876
6098236876 is valid Mobile number
```

In [18]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[0][6-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :07435637732
07435637732 is valid Mobile number
```

In [19]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[0][6-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :05903809282
05903809282 is not valid Mobile number
```

In [20]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[9][1][6-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :917543420987
917543420987 is valid Mobile number
```

In [21]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[9][1][6-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :938763425678
938763425678 is not valid Mobile number
```

In [22]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[+][9][1][6-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :+917485920584
+917485920584 is valid Mobile number
```

In [23]:

```
import re
s = input('Enter Number :')
m = re.fullmatch('[+][9][1][6-9][0-9]{9}',s)
if m!= None:
    print(s,'is valid Mobile number')
else:
    print(s,'is not valid Mobile number')
```

```
Enter Number :-919876543287
-919876543287 is not valid Mobile number
```

App 4: Write a python program to extract all mobile numbers present in input.txt file where numbers are mixed with normal text data.

Note: Executed in Editplus Editor

Assume that, **input.txt** file contains the following information.

C:\Pythonpractice>type iput.txt

Hello this is prathap with mobile number 9885768283

hello this is karthi with mobile number 8787878787

hello this is sahasra iwth mobile number 9101919178

999999999999 aand 963374944994

In [ ]:

```
import re
f1=open("input.txt","r")
f2=open("output.txt","w")
for line in f1:
    print(line)
```

Output

Hello this is prathap with mobile number 9885768283

hello this is karthi with mobile number 8787878787

hello this is sahasra with mobile number 9101919178

In [ ]:

```
import re
f1=open("input.txt","r")
f2=open("output.txt","w")
for line in f1:
    list=re.findall("[7-9]\d{9}",line)
    for n in list:
        f2.write(n+"\n")
print("Extracted all Mobile Numbers into output.txt")
f1.close()
f2.close()
```

Output

Extracted all Mobile Numbers into output.txt

If you want to see the contents of output.txt file, use the following command,

C:\Pythonpractice>type output.txt

9885768283

8787878787

9101919178

9999999999

9633749449

App 5. Write a Python Program to check whether the given mail id is valid gmail id or not?

In [3]:

```
import re
s=input("Enter Mail id:")
m=re.fullmatch("\w[a-zA-Z0-9_.]*@gmail[.]com",s)
if m!=None:
    print("Valid Mail Id");
else:
    print("Invalid Mail id")
```

```
Enter Mail id:prathapnaidu81@gmail.com
Valid Mail Id
```

In [4]:

```
import re
s=input("Enter Mail id:")
m=re.fullmatch("\w[a-zA-Z0-9_.]*@gmail[.]com",s)
if m!=None:
    print("Valid Mail Id");
else:
    print("Invalid Mail id")
```

```
Enter Mail id:prathapnaidu81
Invalid Mail id
```

App 6. Write a python program to check whether given car registration number is valid Telangana State Registration number or not?

In [5]:

```
import re
s=input("Enter Vehicle Registration Number:")
m=re.fullmatch("TS[012][0-9][A-Z]{2}\d{4}",s)
if m!=None:
    print("Valid Vehicle Registration Number");
else:
    print("Invalid Vehicle Registration Number")
```

```
Enter Vehicle Registration Number:TS07EA7777
Valid Vehicle Registration Number
```

In [6]:

```
import re
s=input("Enter Vehicle Registration Number:")
m=re.fullmatch("TS[012][0-9][A-Z]{2}\d{4}",s)
if m!=None:
    print("Valid Vehicle Registration Number");
else:
    print("Invalid Vehicle Registration Number")
```

```
Enter Vehicle Registration Number:AP07EA7898
Invalid Vehicle Registration Number
```

In [8]:

```
import re
s=input("Enter Vehicle Registration Number:")
m=re.fullmatch("TS[012][0-9][a-zA-Z]{2}\d{4}",s)
if m!=None:
    print("Valid Vehicle Registration Number");
else:
    print("Invalid Vehicle Registration Number")
```

```
Enter Vehicle Registration Number:TS123ek5678
Invalid Vehicle Registration Number
```

App 7. Python Program to check whether the given mobile number is valid OR not (10 digit OR 11 digit OR 12 digit)

In [10]:

```
import re
s=input("Enter Mobile Number:")
m=re.fullmatch("(0|91)?[7-9][0-9]{9}",s)
if m!=None:
    print("Valid Mobile Number");
else:
    print("Invalid Mobile Number")
```

```
Enter Mobile Number:09885768283
Valid Mobile Number
```

In [11]:

```
import re
s=input("Enter Mobile Number:")
m=re.fullmatch("(0|91)?[7-9][0-9]{9}",s)
if m!=None:
    print("Valid Mobile Number");
else:
    print("Invalid Mobile Number")
```

```
Enter Mobile Number:919885768283
Valid Mobile Number
```

Exercices

1. Write a Python program to collect all .com urls from the given text file.
2. Write a Python program to display all .txt file names from the given directory.

In [ ]: