Python Inheritance

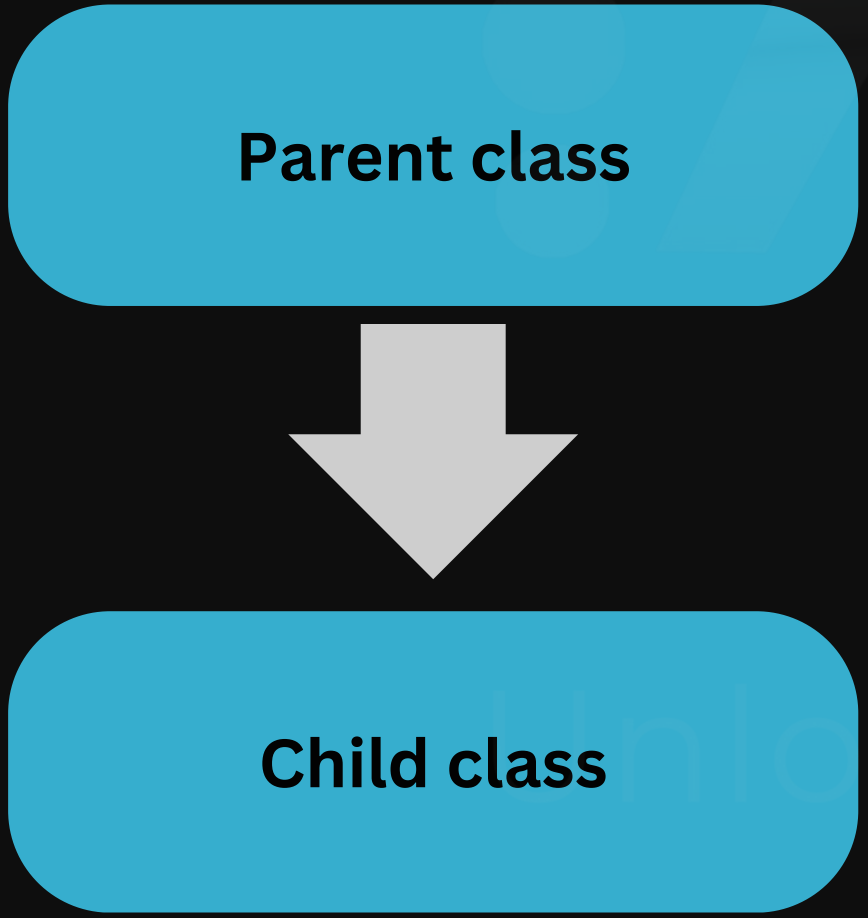
Inheritance is one of the fundamental concepts of Object-Oriented Programming (OOP) in Python. It allows a class to inherit attributes and methods from another class, promoting code reusability and hierarchy.

Codes With Pankaj
Unlock the world of coding

Pankaj Chouhan
www.codeswithpankaj.com

What is Inheritance

Inheritance is a mechanism where one class derives properties and behaviors (methods) from another class.



Base class / Superclass
The class whose properties are inherited.

Derived class / Subclass
The class that inherits from another class

Codes With Pankaj
Unlock the world of coding

Why Use Inheritance ?

- **Code Reusability :**

  Avoids duplication of code.

- **Improves Maintainability :**

  Changes in the parent class reflect in the child class.

- **Encapsulation :**

  Allows you to structure your code in a hierarchical way

Types of Inheritance

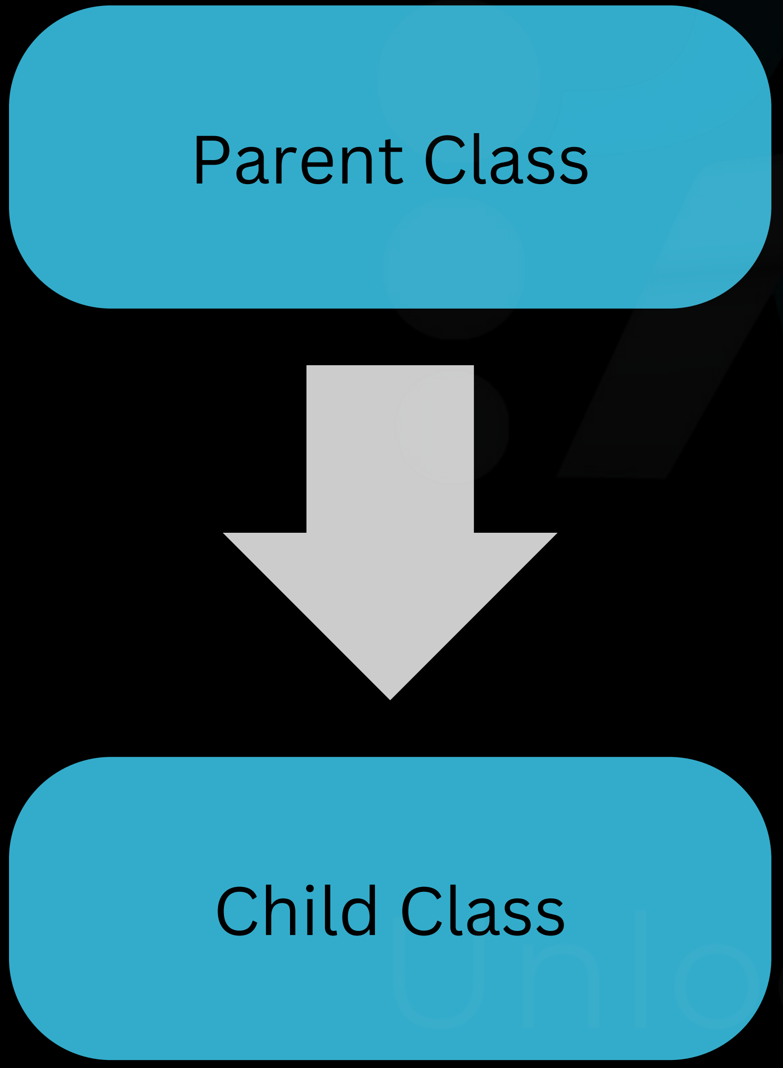
- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Codes With Pankaj

Unlock the world of coding

Single Inheritance

A child class inherits from a single parent class.



Example

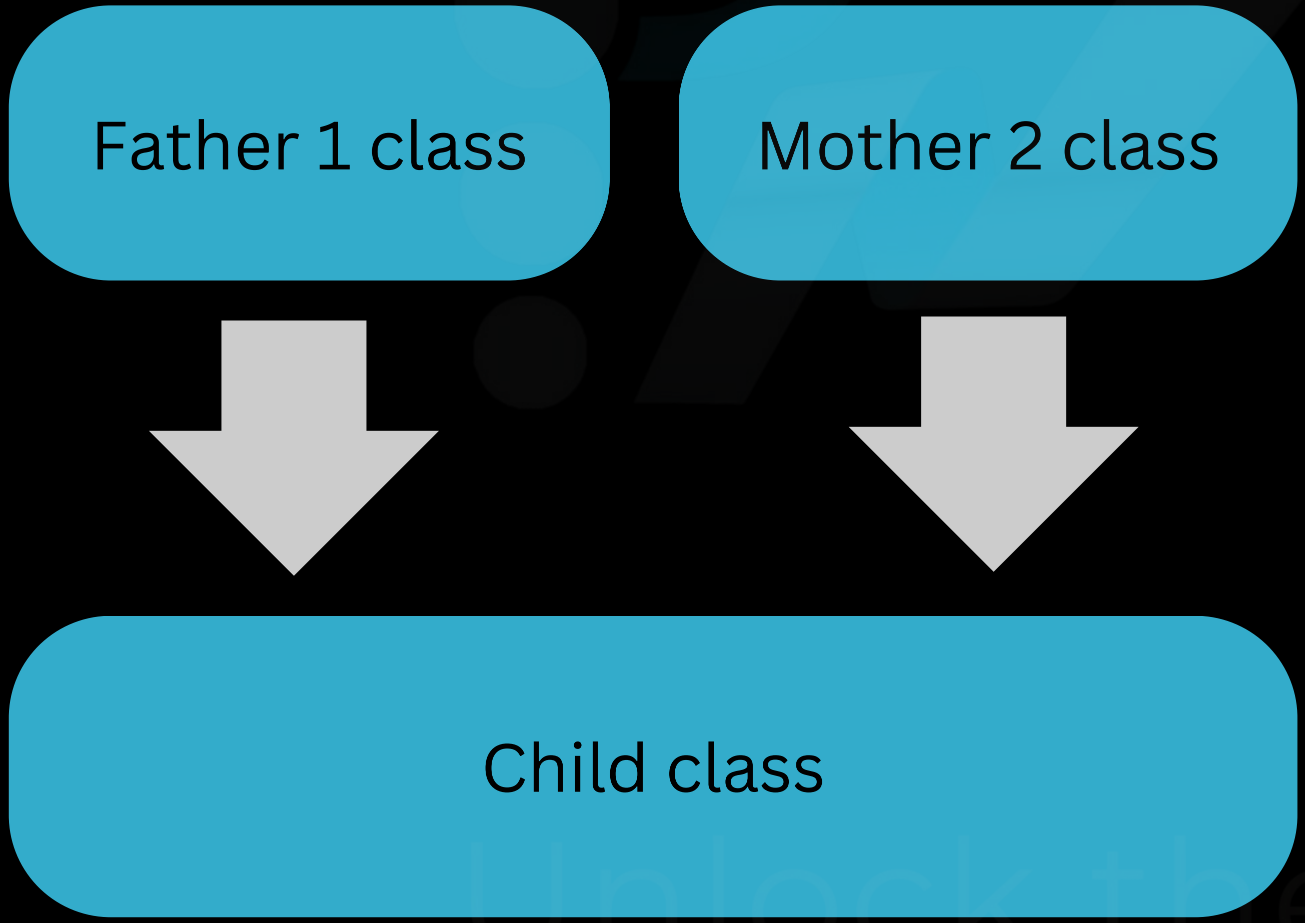
```
class Parent:
    def parent_method(self):
        return "This is parent class"

class Child(Parent):
    def child_method(self):
        return "This is child class"


# Usage
child = Child()
print(child.parent_method())
# Outputs: This is parent class
print(child.child_method())
 # Outputs: This is child class
```

Pankaj Chouhan
www.codeswithpankaj.com

Codes With Pankaj
Unlock the world of coding

Multiple Inheritance

In multiple inheritance, a child class inherits from more than one parent class.



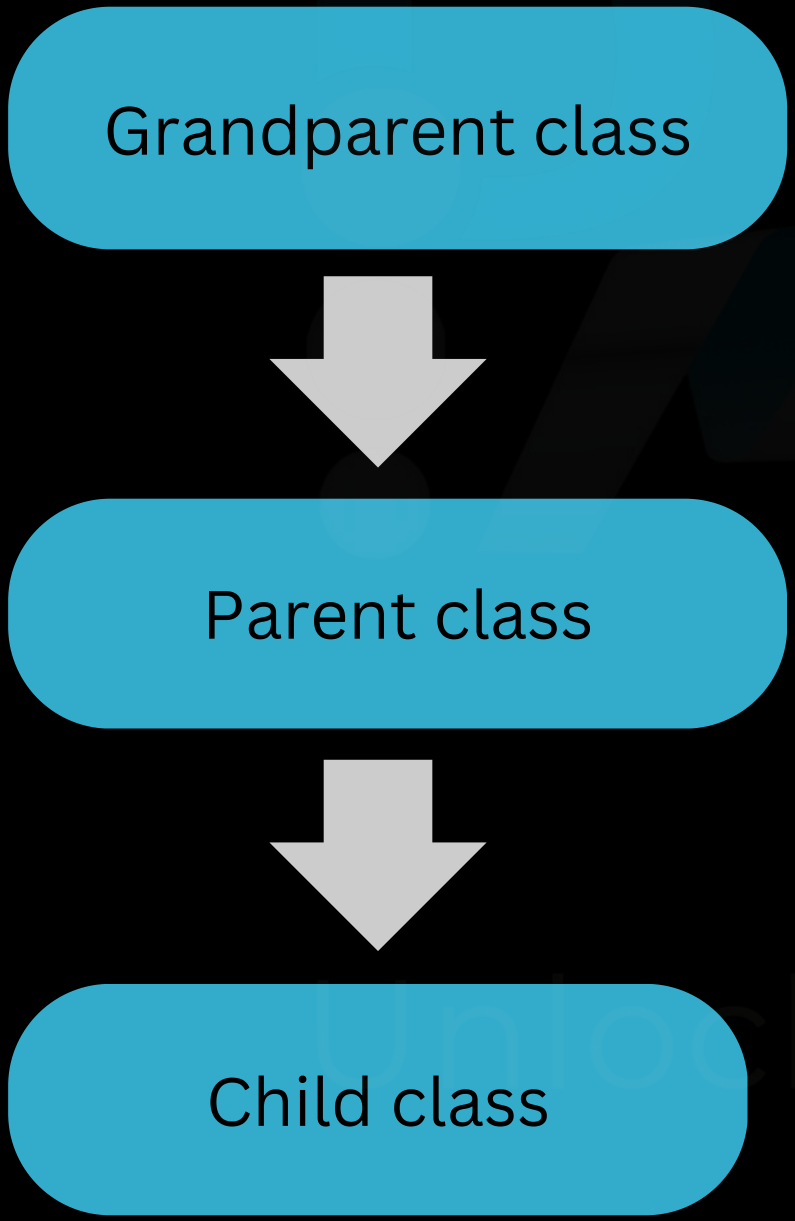
Example

```
class Father:
    def father_method(self):
        return "Father's trait"

class Mother:
    def mother_method(self):
        return "Mother's trait"

class Child(Father, Mother):
    def child_method(self):
        return "Child's trait"


# Usage
child = Child()
print(child.father_method())
# Outputs: Father's trait
print(child.mother_method())
# Outputs: Mother's trait
```

Pankaj Chouhan
www.codeswithpankaj.com

Multilevel Inheritance

In multilevel inheritance, a child class inherits from a parent class, and another child class inherits from that child class.



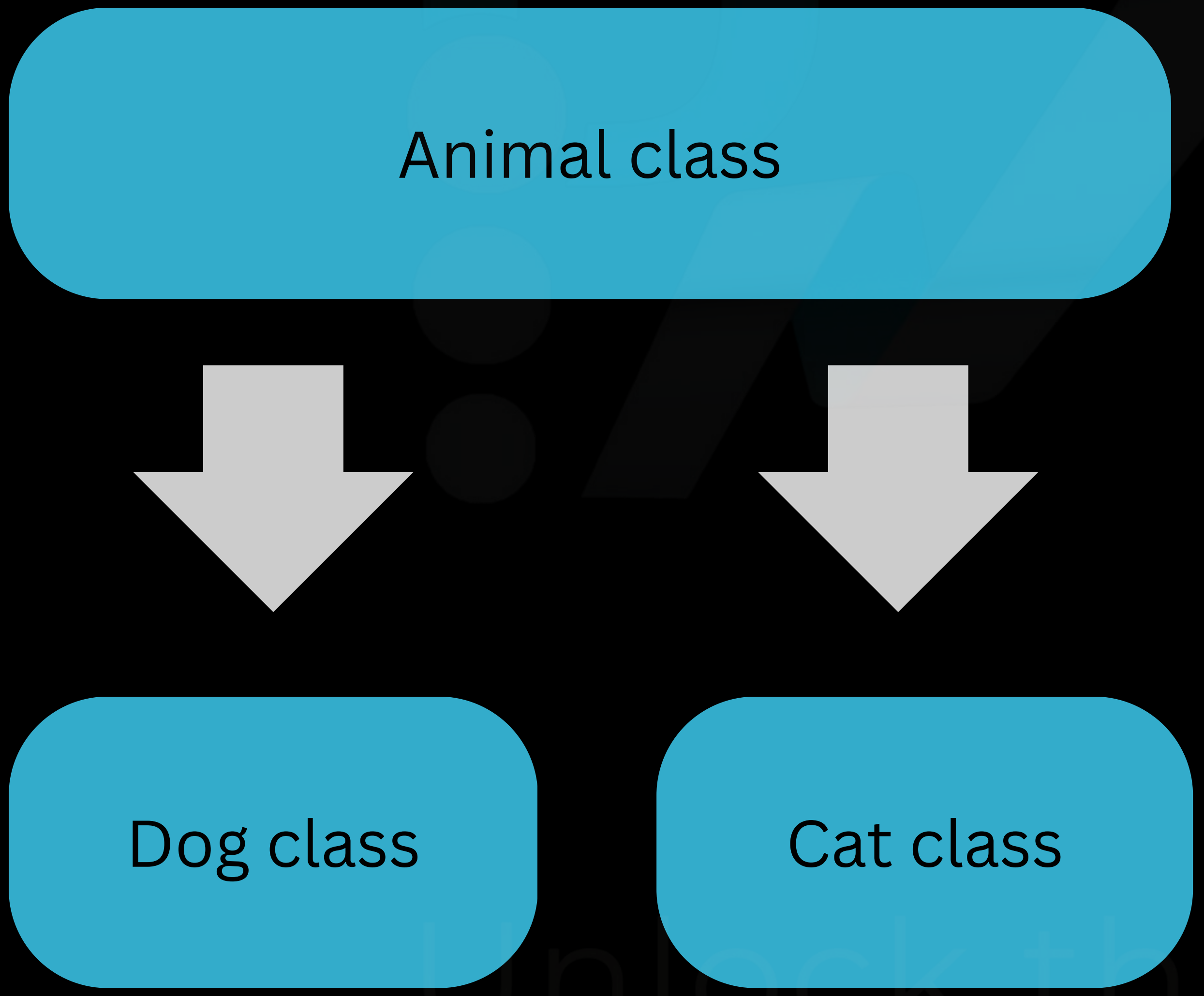
Example

```
class Grandparent:
    def grandparent_method(self):
        return "Grandparent's method"

class Parent(Grandparent):
    def parent_method(self):
        return "Parent's method"

class Child(Parent):
    def child_method(self):
        return "Child's method"

# Usage
child = Child()
print(child.grandparent_method())
# Outputs: Grandparent's method
print(child.parent_method())
# Outputs: Parent's method
```

Pankaj Chouhan
www.codeswithpankaj.com

Hierarchical Inheritance

In hierarchical inheritance, multiple child classes inherit from a single parent class.



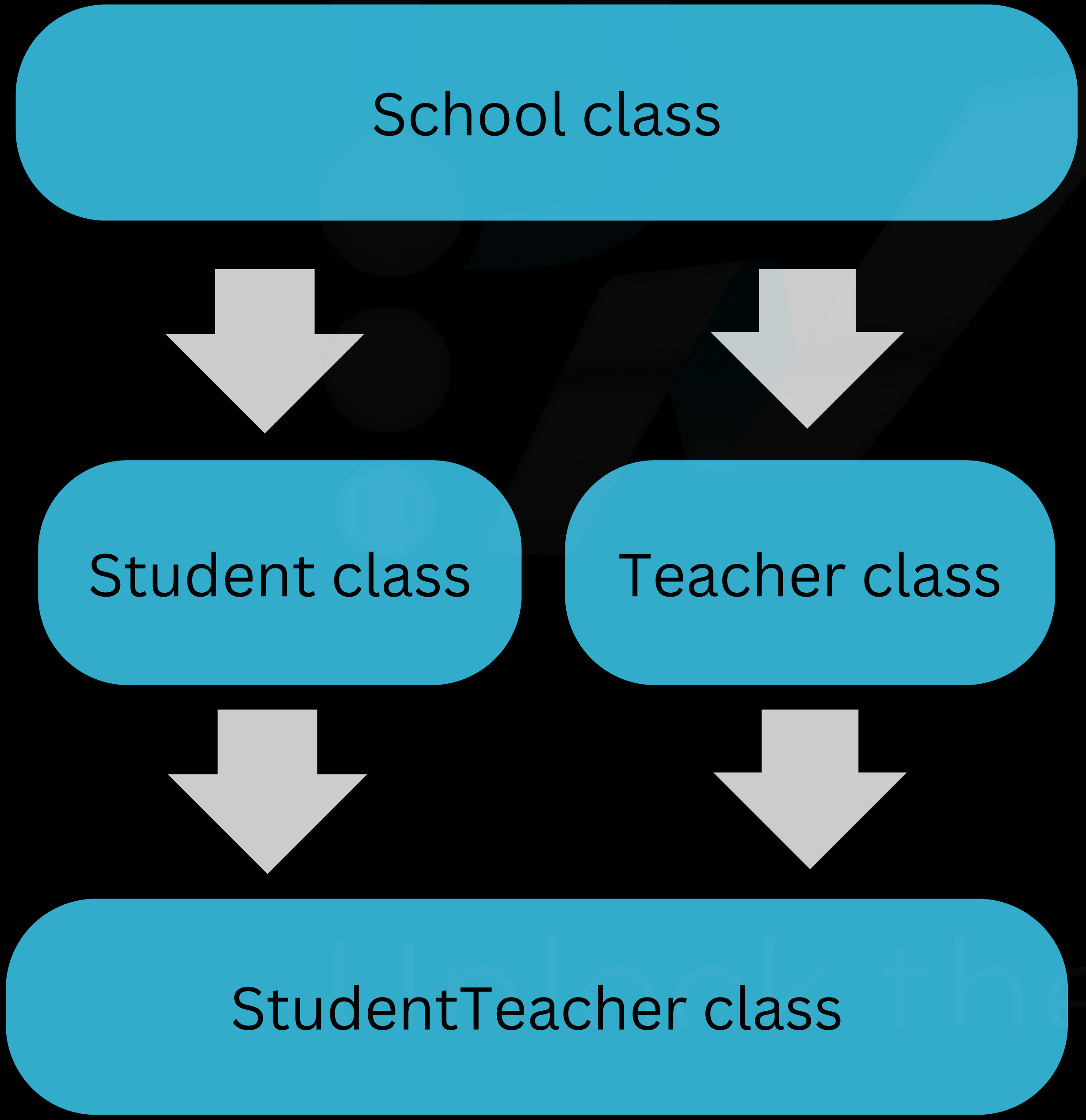
Example

```
class Animal:
    def speak(self):
        return "Animal makes sound"

class Dog(Animal):
    def speak(self):
        return "Dog barks"

class Cat(Animal):
    def speak(self):
        return "Cat meows"

# Usage
dog = Dog()
cat = Cat()
print(dog.speak())  # Outputs: Dog barks
print(cat.speak())  # Outputs: Cat meows
```

Pankaj Chouhan
www.codeswithpankaj.com

Hybrid Inheritance

Hybrid inheritance is a combination of two or more types of inheritance.



Pankaj Chouhan

www.codeswithpankaj.com

Example

```
class School:
    def school_name(self):
        return "ABC School"

class Student(School):
    def student_info(self):
        return "Student class"

class Teacher(School):
    def teacher_info(self):
        return "Teacher class"

class StudentTeacher(Student, Teacher):
    def student_teacher_info(self):
        return "Student Teacher class"

# Usage
st = StudentTeacher()
print(st.school_name())       # Outputs: ABC School
print(st.student_info())      # Outputs: Student
class
print(st.teacher_info())      # Outputs: Teacher class
```

Data Abstraction

Pankaj Chouhan
www.codeswithpankaj.com

Data abstraction is a concept in object-oriented programming that hides unnecessary details from the user and only shows the essential features of an object. It helps in reducing complexity and increasing code readability.

How Does Abstraction Work ?

- In Python, abstraction is achieved using abstract classes and abstract methods.
- An abstract class is a class that cannot be instantiated (you cannot create an object of it).
- It contains abstract methods (methods without implementation) that must be implemented in the child class.

Codes With Pankaj
Unlock the world of coding

Think of data abstraction like a TV remote control. You just need to know which buttons to press, but you don't need to know how it works inside!

Codes With Pankaj

Unlock the world of coding

Example using a Mobile Phone

Think of it this way :

- When you use your real mobile phone, you just press the power button
- You don't need to know how the battery works inside
- You just need to know how to check battery level

This is exactly what abstraction does :

1. Hides complicated stuff inside (using __)
2. Gives you simple methods to use (like switch_on())
3. Protects the data from accidental changes
4. Makes the code easier to use

READ MORE

Pankaj Chouhan

www.codeswithpankaj.com

Example

```
class MobilePhone:
    def __init__(self):
        self.__battery_level = 100
        self.__is_on = False

    def switch_on(self):
        self.__is_on = True
        print("Phone is switched ON")

    def switch_off(self):
        self.__is_on = False
        print("Phone is switched OFF")

    def check_battery(self):
        return f"Battery level: {self.__battery_level}%"

# Using the phone
my_phone = MobilePhone()
my_phone.switch_on()
# Output: Phone is switched ON
print(my_phone.check_battery())
# Output: Battery level: 100%
```

Codes with Pankaj

Unlock the world of coding

Polymorphism

Codes With Pankaj
Unlock the world of coding

Polymorphism means "many forms" in Greek. In Python, polymorphism allows objects of different classes to be treated as objects of a common class. It helps in writing flexible and reusable code.

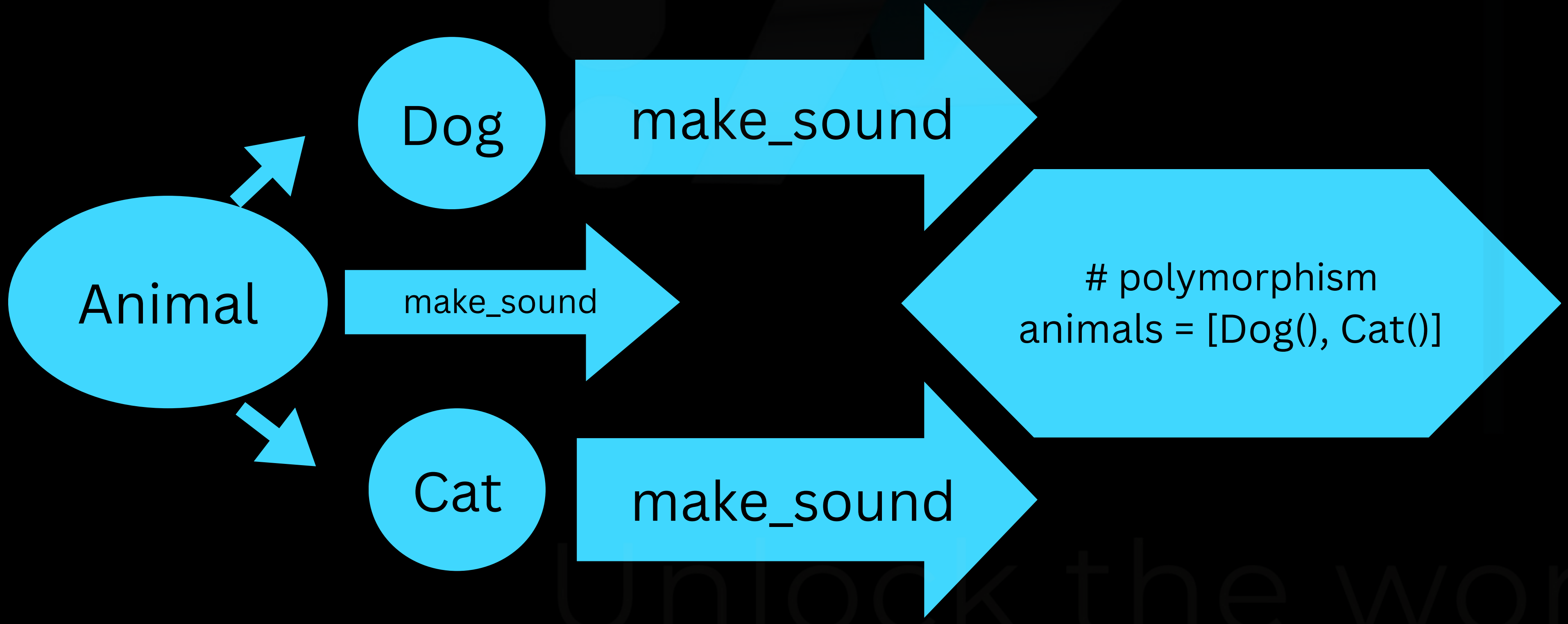
Types of Polymorphism in Python

1. Method Overriding (Runtime Polymorphism)
2. Method Overloading (Python does not support true method overloading but can be achieved using default arguments)
3. Operator Overloading

Codes With Pankaj
Unlock the world of coding

Method Overriding (Runtime Polymorphism)

When a child class provides a specific implementation of a method that is already defined in its parent class.



Pankaj Chouhan

www.codeswithpankaj.com

Example

```
class Animal:
    def make_sound(self):
        print("Animal makes a sound")

class Dog(Animal):
    def make_sound(self):
        # Overriding parent method
        print("Dog barks")

class Cat(Animal):
    def make_sound(self):
        # Overriding parent method
        print("Cat meows")

# Using polymorphism
animals = [Dog(), Cat()]
for animal in animals:
    animal.make_sound()

# Output:
# Dog barks
# Cat meows
```

Codes with Pankaj Unlock the world of coding

Method Overloading

(Not Directly Supported in Python)

Python does not support method overloading like Java/C++, but it can be done using default arguments.

Example

```
class MathOperations:
    def add(self, a, b, c=0):
        # Default argument c
        return a + b + c

obj = MathOperations()
print(obj.add(2, 3))
# Output: 5
print(obj.add(2, 3, 4))
# Output: 9
```

Unlock the world of coding

Operator Overloading

Python allows operators like +, -, * to work differently for different data types by defining special methods like __add__(), __sub__(), etc.

Example

```
class Number:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        # Overloading '+' operator
        return Number(self.value +
other.value)

num1 = Number(5)
num2 = Number(10)
result = num1 + num2
 # Calls __add__() method
print(result.value)
# Output: 15
```

Pankaj Chouhan
www.codeswithpankaj.com

✅ Polymorphism

allows the same method name to have different behaviors.

✅ Method overriding

lets child classes redefine a parent class method.

✅ Method overloading

can be simulated using default arguments.

✅ Operator overloading

lets us use operators with custom classes.

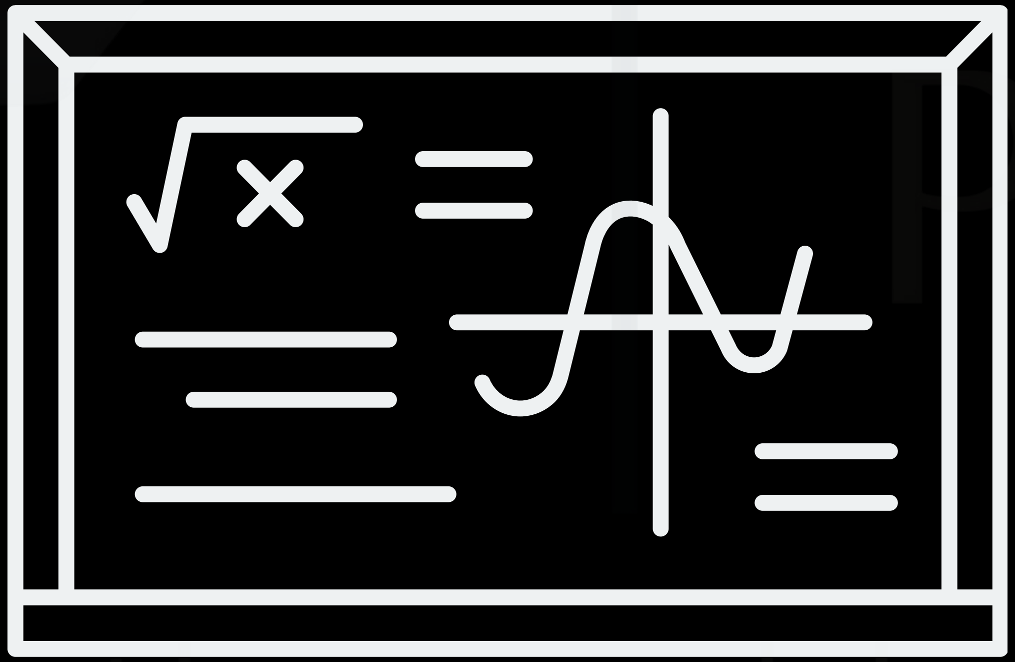
Codes With Pankaj
Unlock the world of coding

Python Recursive Functions

Codes With Pankaj
Unlock the world of coding

Pankaj Chouhan
www.codeswithpankaj.com

Recursion is a fundamental programming concept where a function calls itself to solve smaller instances of a problem. In Python, recursive functions are commonly used to solve problems that can be broken down into smaller, similar subproblems, such as factorial calculation, Fibonacci series, and tree traversals.



Codes With Pankaj
Unlock the world of coding

What is Recursion ?

Recursion is a process where a function calls itself. It continues until a stopping condition (called the base case) is met.

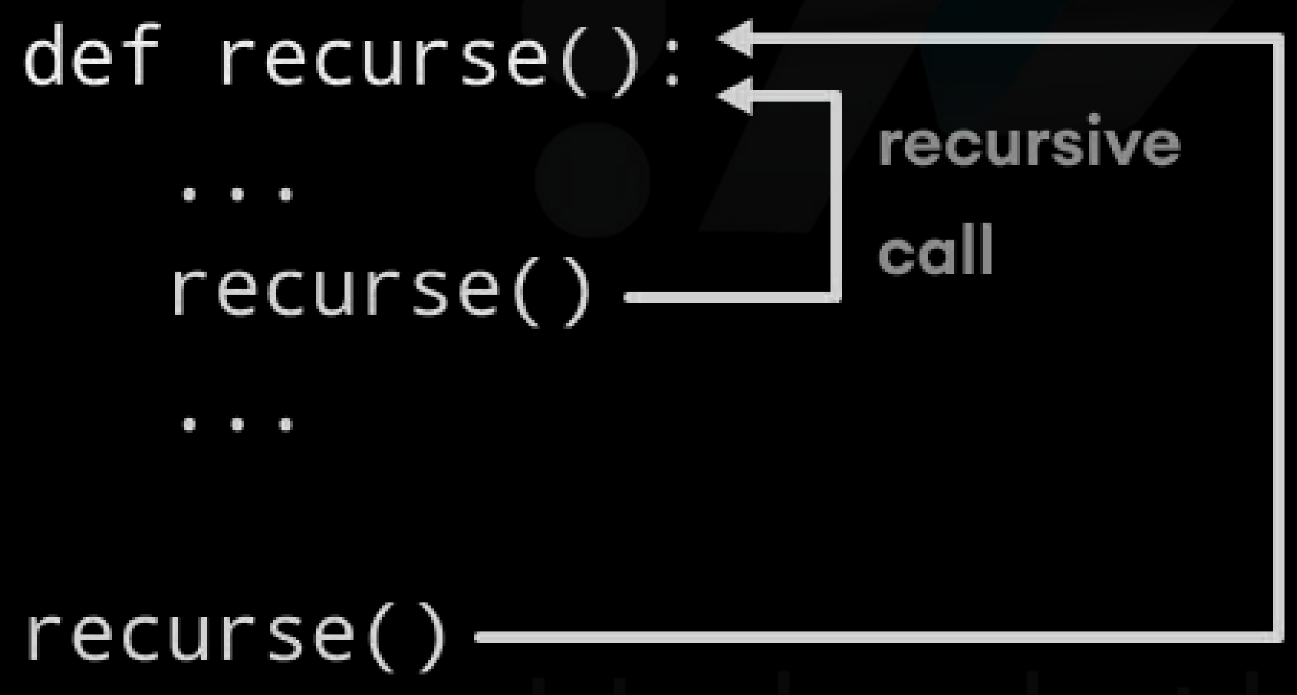
Base case
The condition that stops the recursion.

Recursive case
The condition that stops the recursion.

Codes With Pankaj
Unlock the world of coding

How Does a Recursive Function Work ?

When a function calls itself, each call is stored in the call stack. The function keeps calling itself until it reaches the base case. Then, the function starts returning values and unwinding back through the call stack.


```
def recurse():
    ...
    recurse()
    ...

recurse()
```

Pankaj Chouhan
www.codeswithpankaj.com

Example

```
def recurse():
    print("Calling recurse()")
    recurse()
    # Recursive call

recurse()
```

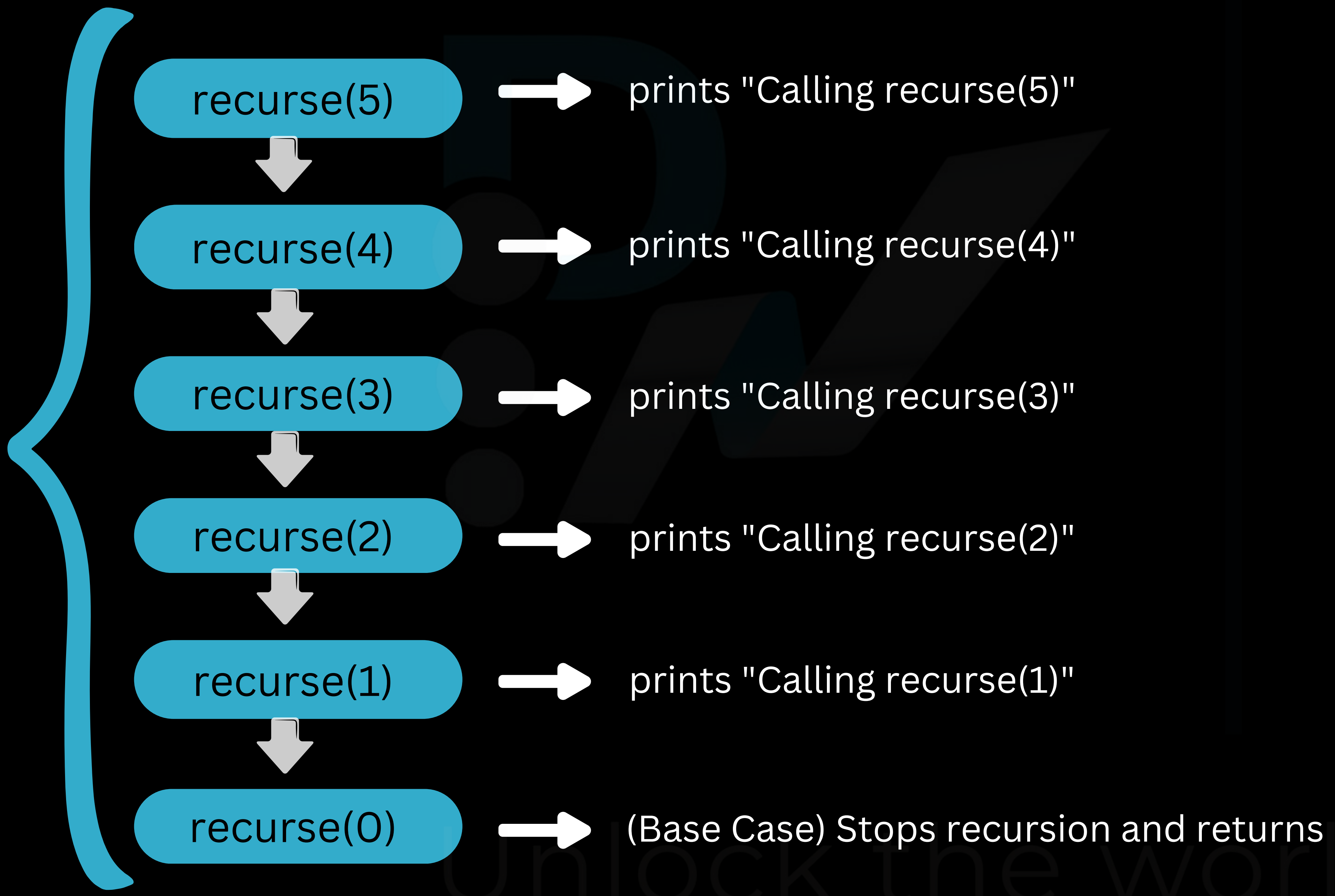
output

```
Calling recurse()
Calling recurse()
Calling recurse()
...
RecursionError: maximum recursion depth exceeded
```

- The function recurse() prints "Calling recurse()".
- Then, it calls itself, leading to an infinite recursion (no base case).
- This will eventually cause a RecursionError because Python has a recursion limit.

To prevent infinite recursion, always include a base case like this :

Step-by-Step Execution


recurse(5)
prints "Calling recurse(5)"
recurse(4)
prints "Calling recurse(4)"
recurse(3)
prints "Calling recurse(3)"
recurse(2)
prints "Calling recurse(2)"
recurse(1)
prints "Calling recurse(1)"
recurse(0)
(Base Case) Stops recursion and returns

Example

```
def recurse(n):
    if n == 0:  # Base case to stop recursion
        return
    print(f"Calling recurse({n}")  # Print
current call
    recurse(n - 1)  # Recursive call

recurse(5)  # Starts recursion
```

Output

```
Calling recurse(5)
Calling recurse(4)
Calling recurse(3)
Calling recurse(2)
Calling recurse(1)
```

Codes With Pankaj
Unlock the world of coding

Pankaj Chouhan
www.codeswithpankaj.com

Examples of Recursive Functions

Factorial of a Number

The formula for factorial of a number n is:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

Or, using recursion:

$$n! = \begin{cases} 1, & \text{if } n = 0 \text{ or } n = 1 \quad \text{(Base Case)} \\ n \times (n-1)!, & \text{if } n > 1 \quad \text{(Recursive Case)} \end{cases}$$

Examples:

- $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
- $4! = 4 \times 3 \times 2 \times 1 = 24$
- $3! = 3 \times 2 \times 1 = 6$
- $2! = 2 \times 1 = 2$
- $1! = 1$ (Base Case)
- $0! = 1$ (Base Case)

Example

```
def factorial(n):
    if n == 0 or n == 1:  # Base case
        return 1
    else:  # Recursive case
        return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

Output $\left\{ \right.$ **120**

Codes With Pankaj
Unlock the world of coding

Pankaj Chouhan
www.codeswithpankaj.com

Step-by-Step Execution

```
def factorial(n):
    if n == 0 or n == 1:  # Base case
        return 1
    else:  # Recursive case
        return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

Function Calls (Going Down the Stack)

Each function call breaks the problem into a smaller one:

Function Call	Computation
factorial(5)	5 * factorial(4)
factorial(4)	4 * factorial(3)
factorial(3)	3 * factorial(2)
factorial(2)	2 * factorial(1)
factorial(1)	Base Case → Returns 1

Function Returns (Unwinding the Stack)

Now, the recursive calls return their values:

Function Call	Returns
factorial(1)	1
factorial(2)	2 * 1 = 2
factorial(3)	3 * 2 = 6
factorial(4)	4 * 6 = 24
factorial(5)	5 * 24 = 120

Thus, factorial(5) returns 120.

Codes With Pankaj
Unlock the world of coding

Lambda Functions (Anonymous Functions)

Pankaj Chouhan
www.codeswithpankaj.com

A **lambda function** in Python is a small, anonymous function that can have any number of arguments but only one expression. It's used for quick, throwaway functions where defining a full function is unnecessary.

Syntax

lambda arguments: expression

Example

```
# Normal function
def square(x):
    return x * x

print(square(5))

# Output: 25

# Equivalent lambda function
square_lambda = lambda x: x * x
print(square_lambda(5))

# Output: 25
```

Unlock the world of coding

Pankaj Chouhan
www.codeswithpankaj.com

Lambda with Multiple Arguments

```
add = lambda x, y: x + y
print(add(3, 7))
# Output: 10
```

Lambda Inside Another Function

```
def multiplier(n):
    return lambda x: x * n

times2 = multiplier(2)
print(times2(5))
# Output: 10
```

Codes With Pankaj
Unlock the world of coding

filter() Function

Codes With Pankaj

Unlock the world of coding

Pankaj Chouhan
www.codeswithpankaj.com

The filter() function is used to filter elements of an iterable based on a condition. It takes a function (which returns True or False) and an iterable.

Syntax

```
filter(function, iterable)
```

Codes With Pankaj
Unlock the world of coding

Filtering Even Numbers

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using lambda with filter
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4, 6, 8, 10]
```

Filtering Words

```
words = ["apple", "banana", "cherry", "avocado"]

# Filter words that start with 'a'
filtered_words = list(filter(lambda word: word.startswith('a'), words))
print(filtered_words)  # Output: ['apple', 'avocado']
```

Codes With Pankaj
Unlock the world of coding

map() Function

Codes With Pankaj

Unlock the world of coding

The **map() function** applies a given function to all items in an iterable and returns a new iterable with modified values.

Syntax

```
map(function, iterable)
```

Codes With Pankaj
Unlock the world of coding

Squaring Numbers

```
numbers = [1, 2, 3, 4, 5]

# Using map with lambda
squared_numbers = list(map(lambda x: x ** 2, numbers))
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

Converting Strings to Uppercase

```
words = ["hello", "world"]

uppercase_words = list(map(lambda word: word.upper(),
words))
print(uppercase_words)  # Output: ['HELLO', 'WORLD']
```

Codes With Pankaj
Unlock the world of coding

reduce() Function (from functools module)

Codes With Pankaj
Unlock the world of coding

The **reduce() function** is used to apply a function cumulatively to the items in an iterable, reducing it to a single value.

Syntax

```
from functools import reduce
reduce(function, iterable)
```

Codes With Pankaj
Unlock the world of coding

Summing All Numbers

```
from functools import reduce

numbers = [1, 2, 3, 4, 5]

# Using reduce with lambda
sum_result = reduce(lambda x, y: x + y, numbers)
print(sum_result)  # Output: 15
```

Finding Maximum Value

```
numbers = [3, 5, 2, 8, 1]

max_number = reduce(lambda x, y: x if x > y else y, numbers)
print(max_number)  # Output: 8
```

Codes With Pankaj
Unlock the world of coding

Comparison: filter() vs map() vs reduce()

Function	Purpose	Returns
filter()	Filters elements based on a condition	A subset of the original iterable
map()	Applies a function to each element	A modified iterable of the same length
reduce()	Reduces the iterable to a single value	A single computed result

Codes With Pankaj
Unlock the world of coding

File Handling in Python

Codes With Pankaj
Unlock the world of coding

What is File Handling ?

File handling is an essential part of any programming language, allowing programs to read from and write to files stored on the system. Python provides built-in functions to create, read, write, and manipulate files efficiently.

Codes With Pankaj
Unlock the world of coding

Different Modes to Open a File
Python provides several modes to open a file :

Mode	Description
r	Read mode (default). Opens the file for reading. If the file doesn’t exist, it throws an error.
w	Write mode. Creates a new file if it doesn’t exist, or overwrites the existing file.
a	Append mode. Adds data to the end of an existing file.
r+	Read and write mode. The file must exist, otherwise, an error occurs.
w+	Write and read mode. Overwrites the file if it exists.
a+	Append and read mode. Adds new content while preserving existing content.
rb, wb, ab	Same as r, w, a, but for binary files.

Codes With Pankaj
Unlock the world of coding

Reading Files in Python

Methods

read()
Reads the entire file content.

```
file = open("example.txt", "r")
content = file.read()
print(content)
file.close()
```

readline()
Reads one line at a time.

```
file = open("example.txt", "r")
line = file.readline()
print(line)
file.close()
```

readlines()
Reads all lines and returns them as a list.

```
file = open("example.txt", "r")
lines = file.readlines()
print(lines)
file.close()
```

Pankaj Chouhan
www.codeswithpankaj.com
Codes With Pankaj
Unlock the world of coding

Writing to Files in Python

Methods

write()
Writes a string to a file

```
file = open("example.txt", "w")
file.write("Hello, World!\n")
file.close()
```

writelines()
Writes a list of strings to a file

```
file = open("example.txt", "w")
file.writelines(["Line 1\n", "Line 2\n"])
file.close()
```

Codes With Pankaj
Unlock the world of coding

Alternatively, use a with statement to automatically close the file:

```
{
  with open("example.txt", "r") as file:
      print(file.read())  # No need to explicitly close the file
}
```

Codes With Pankaj
Unlock the world of coding

Working with File Pointers: seek() and tell()

tell()

Returns the current position of the file pointer.

```
file = open("example.txt", "r")
print(file.tell())  # Shows position
file.read(5)
print(file.tell())  # Updated position
file.close()
```

Using seek()

Moves the pointer to a specific position

```
file = open("example.txt", "r")
file.seek(5)  # Moves pointer to the 5th byte
print(file.read())
file.close()
```

Codes With Pankaj
Unlock the world of coding

Pickling & Unpickling (Storing Python Objects)

Pickling is a way to serialize (save) Python objects, and unpickling retrieves them.

Pickling (Saving Object)

```
import pickle

data = {"name": "Alice", "age": 25}
with open("data.pkl", "wb") as file:
    pickle.dump(data, file)
```

Unpickling (Loading Object)

```
with open("data.pkl", "rb") as file:
    data = pickle.load(file)
print(data)
```

Codes With Pankaj
Unlock the world of coding

Python Exception Handling

Codes With Pankaj
Unlock the world of coding

What are Exceptions ?

Exception handling is an essential concept in Python that allows developers to manage errors gracefully without crashing their programs.

An exception is an error that occurs during the execution of a program, disrupting the normal flow of the program.
For example :

```
print(10 / 0)  # ZeroDivisionError: division by zero
```

Instead of letting the program crash, we can handle this error using Python's built-in exception handling mechanism.

Codes With Pankaj
Unlock the world of coding

Common Python Exceptions

- **ZeroDivisionError :** Division by zero.
- **TypeError :** Invalid operation between different data types.
- **ValueError :** Incorrect value given to a function.
- **IndexError :** Accessing an invalid index in a list.
- **KeyError :** Accessing a non-existent dictionary key.
- **FileNotFoundError :** Trying to open a file that doesn't exist.

Codes With Pankaj
Unlock the world of coding

Basic Exception Handling with try and except

The try block is used to test a piece of code that may raise an exception. If an exception occurs, it is caught by the except block.

```
print(10 / 0)
 # ZeroDivisionError : division by zero
```

```
try:
   x = 10 / 0  # This will raise ZeroDivisionError
except ZeroDivisionError:
   print("Error: You cannot divide by zero!")
```

Handling a Single Exception

Codes With Pankaj
Unlock the world of coding

```
print(10 / 0)
 # ZeroDivisionError
 : division by zero
```

```
try:
    num = int(input("Enter a number: "))
# User might enter a non-numeric
value
    result = 10 / num  # Might cause
ZeroDivisionError
except ZeroDivisionError:
    print("Error: Division by zero is not
allowed.")
except ValueError:
    print("Error: Invalid input! Please
enter a numeric value.")
```

Unlock the world of coding

Handling Multiple Exceptions

Using else and finally

- else Block: Executes if no exception occurs.
- finally Block: Always executes, whether an exception occurs or not.

```
print(10 / 0)
 # ZeroDivisionError
 : division by zero
```

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError:
    print("Error: Cannot divide by zero!")
except ValueError:
    print("Error: Enter a valid number!")
else:
    print(f"Result: {result}")  # Runs if no
exception occurs
finally:
    print("Execution completed.")  #
Always runs
```

Codes with Pankaj
Unlock the world of coding

Handling Multiple Exceptions with a Single except Block

- Instead of writing multiple except blocks, we can use a tuple to catch multiple exceptions in a single block.

```
print(10 / 0)
 # ZeroDivisionError
 : division by zero
```

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except (ZeroDivisionError, ValueError)
as e:
    print(f"Error occurred: {e}")
```

Codes With Pankaj
Unlock the world of coding

Regular Expressions (RegEx)

Pankaj Chouhan
www.codeswithpankaj.com

What is Regular Expression ?

A Regular Expression (RegEx) is a pattern used to search for specific words, numbers, or symbols in text. It helps in finding, replacing, and extracting information from large text data easily.

Contact me at $\underbrace{\text{codeswithpankaj@gmail.com}}_{1}$ and $\underbrace{\text{p4n.learning@gmail.com}}_{2}$

$$\begin{tikzcd} & \text{codeswithpankaj@gmail.com} \\ \text{Emails found} \arrow[ur] \arrow[r] & \text{p4n.learning@gmail.com} \end{tikzcd}$$

Codes With Pankaj
Unlock the world of coding

Basic RegEx Patterns

Pattern	Meaning	Example
\d	Matches any digit (0-9)	"My number is 1234" → \d+ finds 1234
\w	Matches any letter or number	"Hello_123" → \w+ finds Hello_123
\s	Matches spaces	"Hello World" → \s finds space
.	Matches any character	"a.b" → . finds a and b
*	Matches 0 or more times	"go*" → matches "g", "go", "goo"
+	Matches 1 or more times	"go+" → matches "go", "goo", but not "g"
?	Matches 0 or 1 time	"colou?r" → matches "color" and "colour"
^	Matches start of string	"^Hello" → matches "Hello World" but not "World Hello"
$	Matches end of string	"world$" → matches "Hello world" but not "world Hello"
[]	Matches any character inside brackets	[aeiou] → matches "a" in "apple"
()	Groups patterns	(ab)+ → matches "ababab" in "abababxyz"

Codes With Pankaj
unlock the world of coding

How to Use RegEx in Python ?

Python has a built-in module called re that helps us use regular expressions. To use it, we first need to import it:

built-in module

import re

This module provides regular expression matching operations similar to those found in Perl

Codes With Pankaj
Unlock the world of coding

Finding Numbers in a Sentence

```
import re

text = "The price is 150 dollars."
match = re.search(r'\d+', text)
if match:
    print("Number found:", match.group())  # Output: 150
```

Output: 150

The price is **150** dollars.

Codes With Pankaj
Unlock the world of coding

Finding Emails in a Text

```
import re

text = "Contact me at codeswithpankaj@gmail.com and p4n.learning@gmail.com"
emails = re.findall(r'[\w.-]+@[\w.-]+', text)
print("Emails found:", emails)
```

Output

```
Emails found: ['codeswithpankaj@gmail.com', 'p4n.learning@gmail.com']
```

Codes With Pankaj
Unlock the world of coding

Replacing Text Using RegEx

```
import re

text = "Hello 123, this is a test 456."
new_text = re.sub(r'\d+', '###', text)
print(new_text)
```

Output

```
"Hello ###, this is a test ###."
```

Extracting Words from a Sentence

```
import re

text = "Python is an amazing programming language!"
words = re.findall(r'\w+', text)
print("Words found:", words)
```

Output

```
Python is an amazing programming language !
   1    2  3    4          5          6

['Python', 'is', 'an', 'amazing', 'programming', 'language']
```

Pankaj Chouhan
www.codeswithpankaj.com

Conclusion

1. Regular Expressions (RegEx) help search for patterns in text.
2. Useful for finding words, numbers, emails, and more.
3. We can replace and extract specific parts of text using RegEx.
4. Python's re module makes working with RegEx easy.

Codes With Pankaj
Unlock the world of coding

Web Scraping in Python

Codes With Pankaj

Unlock the world of coding

Pankaj Chouhan

www.codeswithpankaj.com

What is Web Scraping ?

Web scraping is a technique to extract information from websites. It is used to collect data from web pages automatically.

Libraries for Web Scraping

- **Requests :** Fetches web pages.
- **BeautifulSoup :** Extracts data from HTML.

Codes With Pankaj
Unlock the world of coding

Prerequisites

Install Required Libraries

Run the following command to install them

```
!pip install requests beautifulsoup4
%pip install requests beautifulsoup4
```

Import Required Libraries

```
import requests
from bs4 import BeautifulSoup
```

Codes With Pankaj
Unlock the world of coding

Step by step. We will

✅ Fetch a webpage using requests

✅ Parse and extract data using BeautifulSoup

✅ Extract specific information like blog titles and links

✅ Save data to a CSV file

Codes with Pankaj

Unlock the world of coding

Step 1: Install Required Libraries

```
pip install requests beautifulsoup4 lxml pandas
```

- **requests** → To fetch the website HTML
- **beautifulsoup4** → To parse and extract data from the HTML
- **lxml** → To improve parsing performance
- **pandas** → To save the data in a structured format

Codes With Pankaj
Unlock the world of coding

Step 2: Fetch the Website HTML

Let's send a request to www.codeswithpankaj.com and retrieve the page content.

```
import requests

url = "https://www.codeswithpankaj.com/"
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}

response = requests.get(url, headers=headers)

# Check if the request was successful
if response.status_code == 200:
    print("Page fetched successfully!")
    print(response.text[:500])  # Print first 500 characters of the HTML
else:
    print(f"Failed to fetch page. Status code: {response.status_code}")
```

Explanation :

✓ We use requests.get(url) to fetch the webpage.
✓ We add a User-Agent header to prevent blocking.
✓ If the request is successful (status_code == 200), we print a portion of the HTML.

Codes With Pankaj
Unlock the world of coding

Step 3: Parse the HTML with BeautifulSoup

Once we get the page content, we parse it using BeautifulSoup.

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(response.text, "html.parser")

# Print the page title
print("Page Title:", soup.title.string)
```

Explanation :

✓ We pass the HTML content to BeautifulSoup for parsing.
✓ soup.title.string extracts the webpage title.

Codes With Pankaj
Unlock the world of coding

Step 4: Extract Blog Titles & Links

Now, let's extract the latest blog post titles and their links from the homepage.

```
# Find all blog post titles and links
articles = soup.find_all("h2", class_="post-title")

for article in articles:
    title = article.text.strip()
    link = article.a["href"]
    print(f"Title: {title}")
    print(f"Link: {link}\n")
```

Explanation :

✓ soup.find_all("h2", class_="post-title") finds all <h2> elements with the class post-title.

✓ We extract the blog title and link using .text.strip() and .a["href"].

Codes With Pankaj
Unlock the world of coding

Step 5: Save Data to CSV

Let's store the extracted data in a CSV file.

```
import csv

# Open CSV file to save data
with open("blog_posts.csv", "w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow(["Title", "Link"])  # Column headers

    # Loop through extracted articles
    for article in articles:
        title = article.text.strip()
        link = article.a["href"]
        writer.writerow([title, link])

print("Data saved to blog_posts.csv")
```

Explanation :

✓ We open a CSV file in write mode.
✓ We write column headers: "Title", "Link".
✓ We loop through the extracted data and store it in the file.

Codes With Pankaj
Unlock the world of coding

Step 6: Handling Pagination (Multiple Pages)

If the website has multiple pages, we can scrape all pages by looping through them.

```
page = 1
all_posts = []

while True:
    url = f"https://www.codeswithpankaj.com/page/{page}/"
    response = requests.get(url, headers=headers)

    if response.status_code != 200:
        break  # Stop if no more pages

    soup = BeautifulSoup(response.text, "html.parser")
    articles = soup.find_all("h2", class_="post-title")

    if not articles:
        break  # Stop if no more blog posts

    for article in articles:
        title = article.text.strip()
        link = article.a["href"]
        all_posts.append([title, link])

    page += 1

# Save to CSV
with open("all_blog_posts.csv", "w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow(["Title", "Link"])
    writer.writerows(all_posts)

print(f"Scraped {len(all_posts)} blog posts and saved to all_blog_posts.csv")
```

Explanation :

✓ We loop through pages (/page/1/, /page/2/, etc.).
✓ If no articles are found, we stop scraping.
✓ We save all posts to all_blog_posts.csv.

Codes With Pankaj
Unlock the world of coding

Step 7: Avoid Getting Blocked

Websites may block scrapers if they detect too many requests. Here’s how to avoid that:

```
import time
import random

time.sleep(random.randint(2, 5))
# Wait 2-5 seconds before the next request
```

Use Random Delays

```
import random

user_agents = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "Mozilla/5.0 (X11; Ubuntu; Linux x86_64)"
]

headers = {"User-Agent": random.choice(user_agents)}
```

Rotate User-Agents

Pankaj Chouhan

Final Thoughts

🎉 Congratulations! You've successfully scraped www.codeswithpankaj.com.

✅ Fetched the webpage

✅ Extracted blog titles & links

✅ Saved data to a CSV file

✅ Scraped multiple pages

✅ Avoided getting blocked

More projects