



PYTHON

101

Created by
Cameron McKimm

For Kerry Annett.

TABLE OF CONTENTS

Introduction	4
<i>What is a Program?.....</i>	<i>4</i>
<i>What is Python?</i>	<i>5</i>
Setting up Python.....	5
Your First Python Program	11
<i>Creating your Program in the Console.....</i>	<i>11</i>
<i>Creating your Program in a Script.....</i>	<i>12</i>
Understanding the Hello World Program.....	16
<i>Data types.....</i>	<i>16</i>
<i>Functions and Methods.....</i>	<i>23</i>
<i>Discussing the Hello World Program.....</i>	<i>26</i>
Conditional Statements	27
For and While Loops	29
Thank you for Reading!.....	33

INTRODUCTION

Programming Languages are the basis of modern-day technology. More than ever, it is such a vital thing to learn with the rapidly expanding environment of electronics that we have at our fingertips.

Knowing how to code will allow the programmer to communicate to a plethora of people across countries, cultures and computers.

Furthermore, you will be able to think logically about a problem and apply efficient solutions to solve that issue. Hence, opening up a world of possibility: the code is your oyster!

This manual will be useful for getting the core concepts of programming under your belt and it will walk you through some practical problems.

Try not to be too daunted at trying something new, we will get through this together. I can't wait to see what you'll create!

Let's get stuck in.

WHAT IS A PROGRAM?

A program is a collection of mathematical and logical functions grouped to perform a specific task.

Programs are usually created by computer programmers in a high-level language such as Python (although there are a lot of other programming languages, such as Java, C++ etc).

The compiler for these programs converts them into machine code which is understood by the computer and can be executed directly.

Alternatively, a computer program can be executed with the aid of an interpreter.

WHAT IS PYTHON?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

Python is easy to use, powerful, and versatile, making it a great choice for beginners and experts alike. Python’s readability makes it a great first programming language — it allows you to think like a programmer and not waste time with confusing syntax.

SETTING UP PYTHON

First of all, you need to navigate to <https://www.python.org/downloads/>. This will prompt you with a webpage that shouldn’t look to dissimilar to **Fig 1**. Find the big button which remarks “Download Python X.Y.Z” (the latest version as I write this is: 3.9.1) and click it. If you are looking for a version which is compatible with your operating system (OS) and your browser has not correctly identified which OS you are using, click the corresponding OS below the Download button.



Fig 1

Note: For the duration of this manual, I shall be using MacOS, so your setup might look a little different, but it is in essence the same idea.

Now that you have downloaded the latest version of Python, you need to find the installation file. My download was saved to my Downloads folder at ~/Downloads. The file is shown in *Fig 2*.

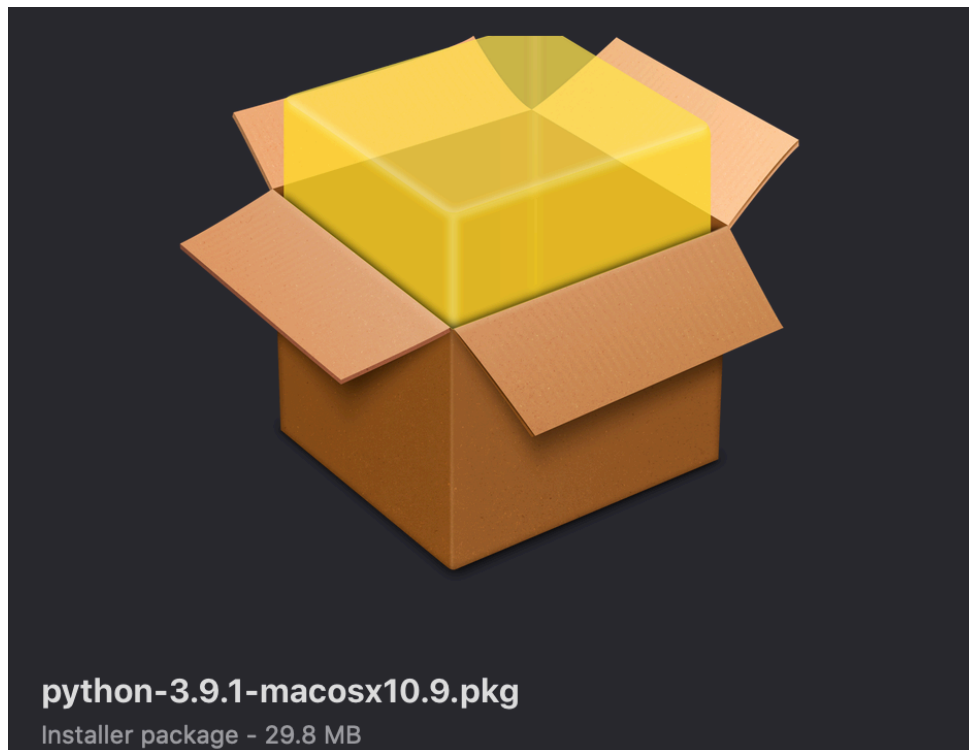


Fig 2

Once you have found it, click on it, this should take you to the installer which looks like *Fig 3* for me.

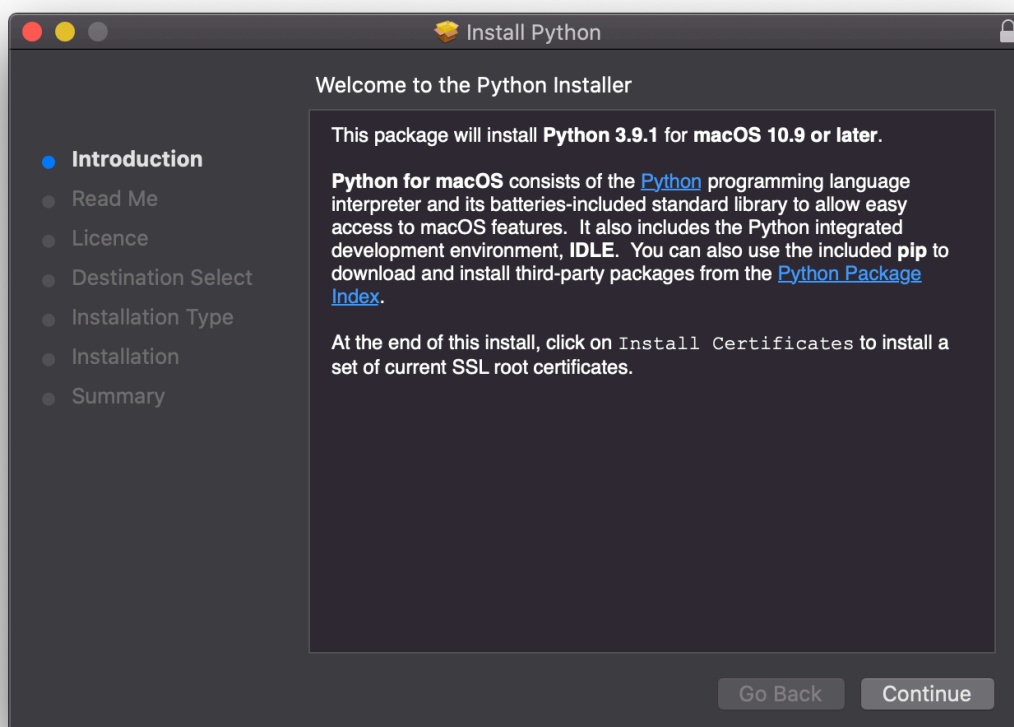


Fig 3

Press the *Continue* button until you get *License*. You will be prompted with a message asking you to Agree or Disagree with the terms of the Software License Agreement. Just click Agree. The window should like *Fig 4*.

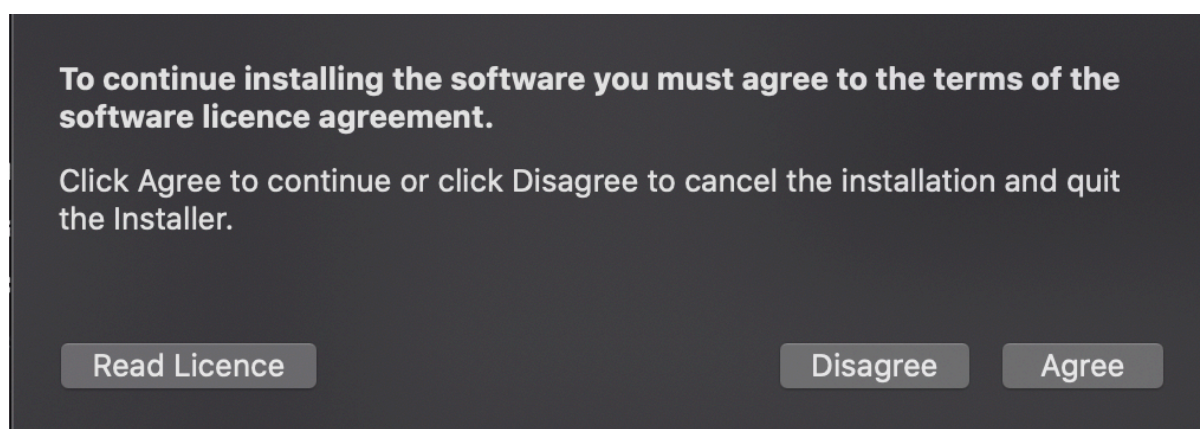


Fig 4

Next, you need to select the drive where you would want the download to be stored. This is easy for me as I only have one drive installed. So just click *Macintosh HD* and then *Continue* in *Fig 5*.

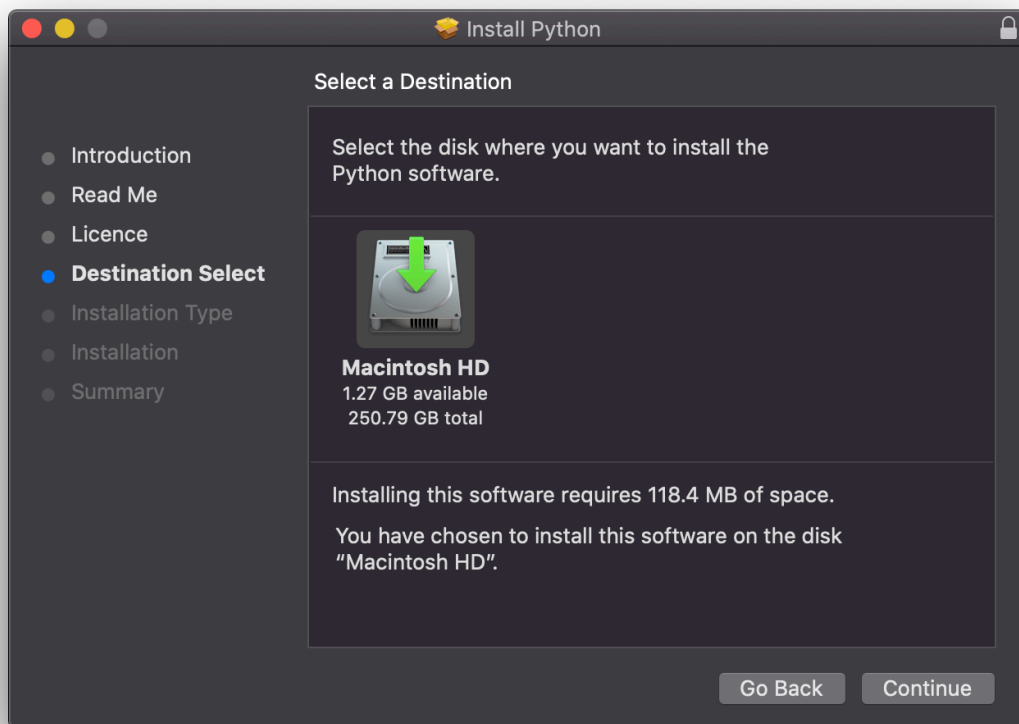


Fig 5

For the next step, click *Install* in **Fig 6**. This will begin the full installation of the software. You might be prompted with a window for your computer Credentials, if so, just use the same username and password that you would use if you were logging into your computer.

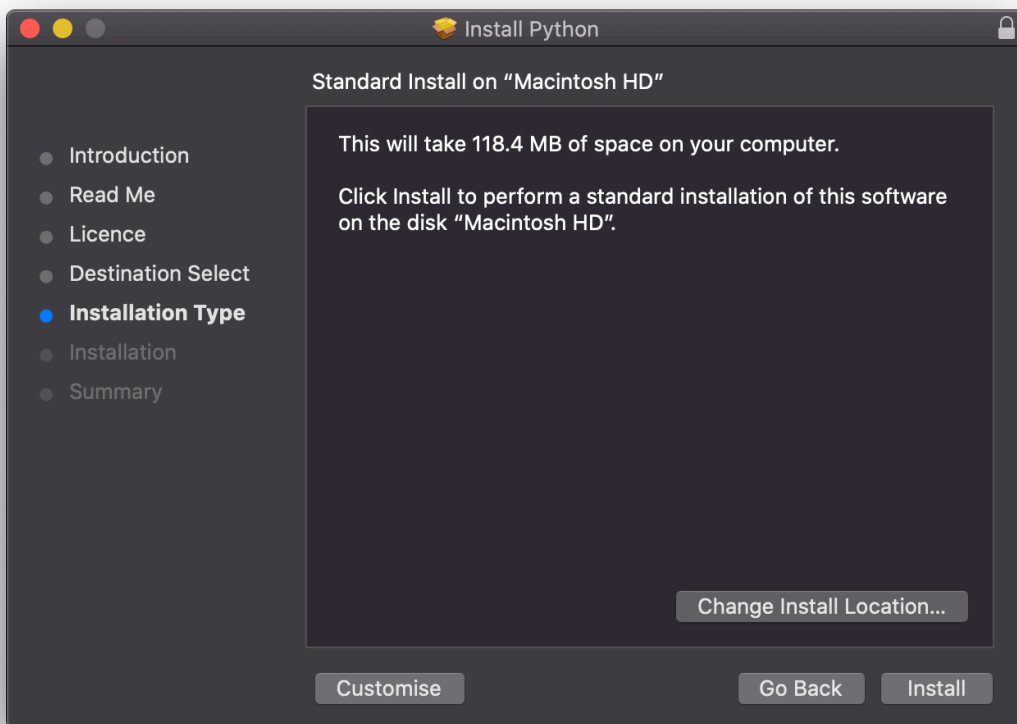


Fig 6

Now that it has successfully installed, you need to find the folder that it has created. My folder was at **/Applications/Python 3.9** and it contained the following in **Fig 7**. Notice that there is a file called **IDLE**: this stands for *Integrated Development and Learning Environment*. This is your main point of access to python (for now...), now you can click on it.

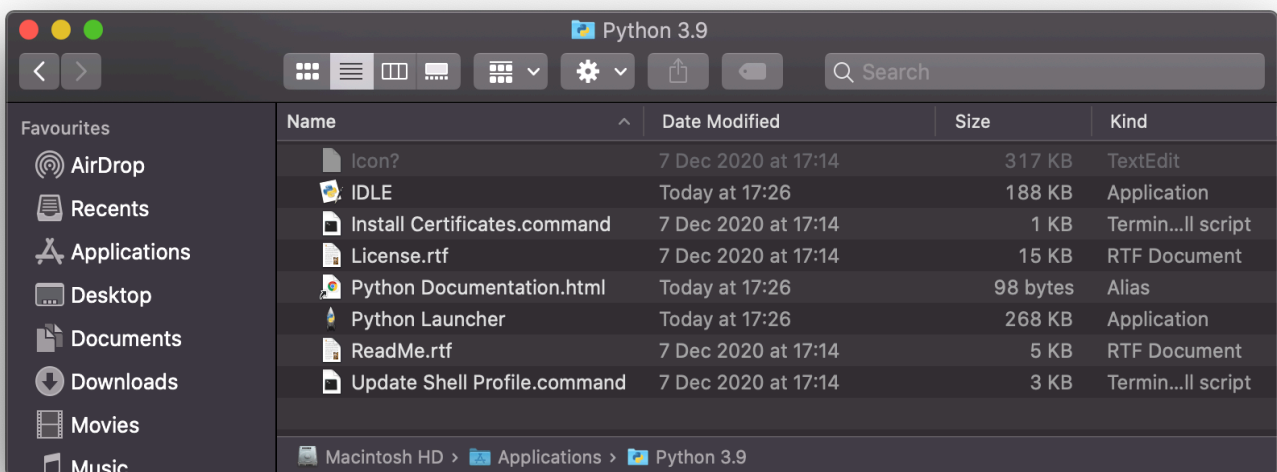


Fig 7

We're almost there!

You might be prompted with the window in **Fig 8**. If so, just click *OK*.

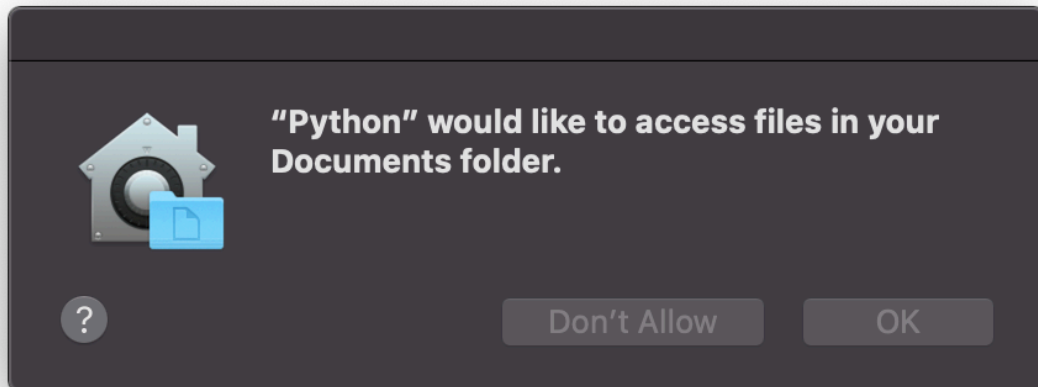


Fig 8

Finally, you should now have IDLE open and it should look like **Fig 9**.

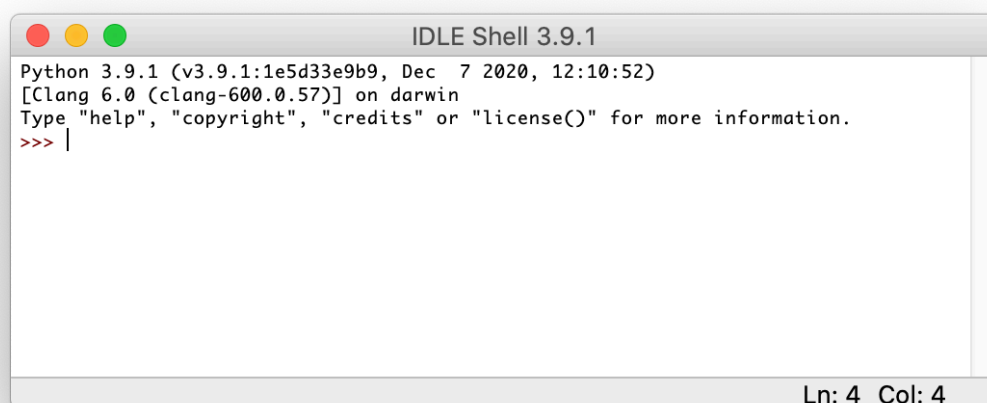


Fig 9

Congratulations! You have successfully installed Python! 😊

YOUR FIRST PYTHON PROGRAM

There is a convention when it comes to your very first program, it's called Hello World! It is a very simple program which simply outputs "Hello, World!" to the console.

History: While small test programs have existed since the development of programmable computers, the tradition of using the phrase "Hello, World!" as a test message was influenced by an example program in the seminal 1978 book *The C Programming Language*. The example program in that book prints "hello, world", and was inherited from a 1974 Bell Laboratories internal memorandum by Brian Kernighan, *Programming in C: A Tutorial*:

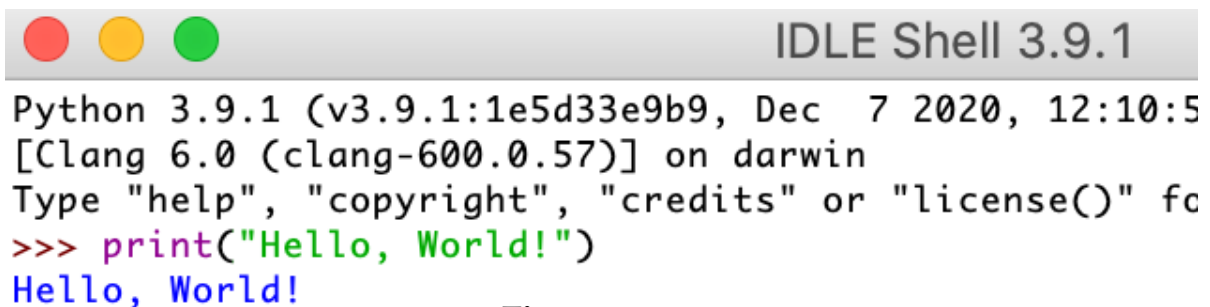
```
main( ) {  
    printf("hello, world\n");  
}
```

CREATING YOUR PROGRAM IN THE CONSOLE

Now it's your turn to write your first Python Program. Type the following into IDLE:

```
1. print("Hello, World!")
```

A blue output should have been returned which should look like **Fig 10**.



```
IDLE Shell 3.9.1  
Python 3.9.1 (v3.9.1:1e5d33e9b9, Dec 7 2020, 12:10:5  
[Clang 6.0 (clang-600.0.57)] on darwin  
Type "help", "copyright", "credits" or "license()" fo  
>>> print("Hello, World!")  
Hello, World!
```

Fig 10

Congratulations, you have made your very first Python program!



CREATING YOUR PROGRAM IN A SCRIPT

Since we have that under our belt, we can talk semantics. Usually, you would not use the console here for programs any larger than this. You would create a *script*. A script does the exact same thing as typing the commands into the console; however, you can reference scripts from other scripts, and you can debug your code much easier.

To write the same Hello World! Program as a script, we do things slightly differently. Notice in **Fig 11** there is a tab named *File* at the top left-hand side, we shall click on this.

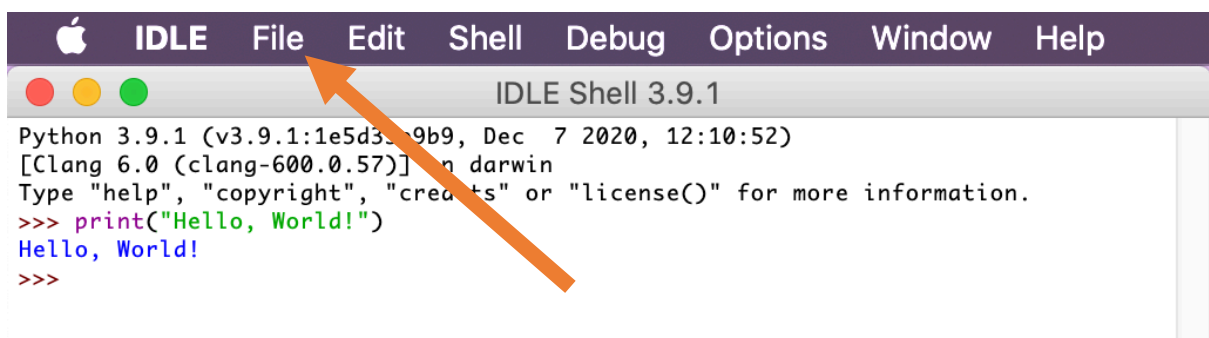


Fig 11

Hence, we will be prompted with **Fig 12**, we shall click the top option which is *New File*.

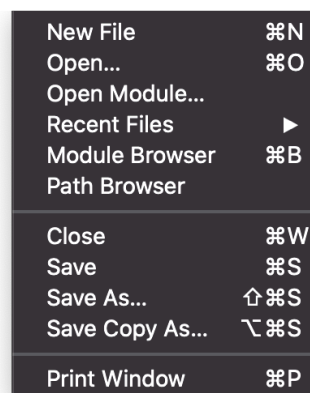


Fig 12

So, we should now have a blank python file which we can type our commands in. This is where we will write our script for Hello World! See **Fig 13**.

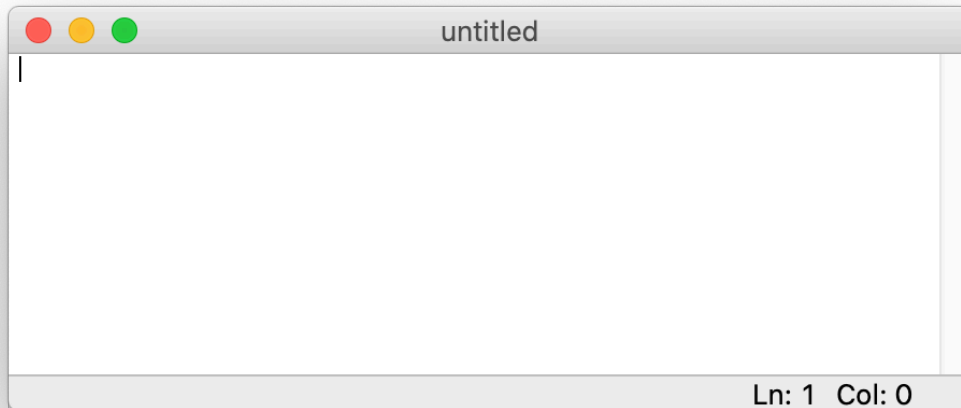


Fig 13

Here we can type the same command as we did in the console:

```
1. print("Hello, World!")
```

Now the script should look like **Fig 14**, notice at the top the title of the window is **untitled**, this means that we have not saved the file yet. Usually, when you make any change to a file, the title of the script will be encapsulated in asterisks, so say the title of the file was *DogsAndCats*, if we were to make any change, like adding a line of code, removing code, even adding a space, then the title would change to **DogsAndCats**. After saving the file again, the title would resort back to *DogsAndCats*.



Fig 14

Therefore, we shall save the file and call it *HelloWorld.py* which will make it easy to identify what the file does. Go to the File tab, as we did previously and click on *Save*. This will prompt you to see **Fig 15**, fill out the information accordingly:

Save As – *HelloWorld*,

Tags – **YOU CAN LEAVE BLANK**,

Where – **SELECT WHAT FOLDER YOU WANT TO SAVE THE FILE TO**
and Format – Python files (.py, .pyw).

Now, click on *Save*.

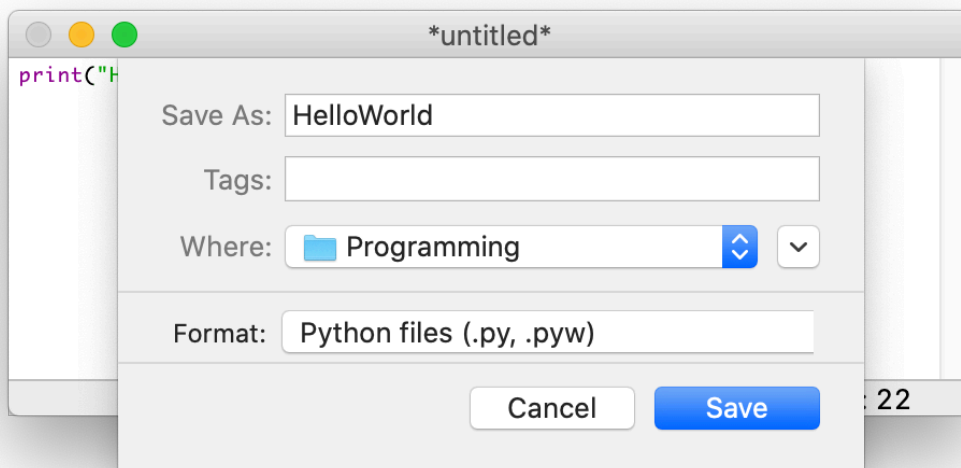


Fig 15

Consequently, we can now run the script. If we find the *Run* tab in **Fig 16**, we can click *Run Module* in **Fig 17**.

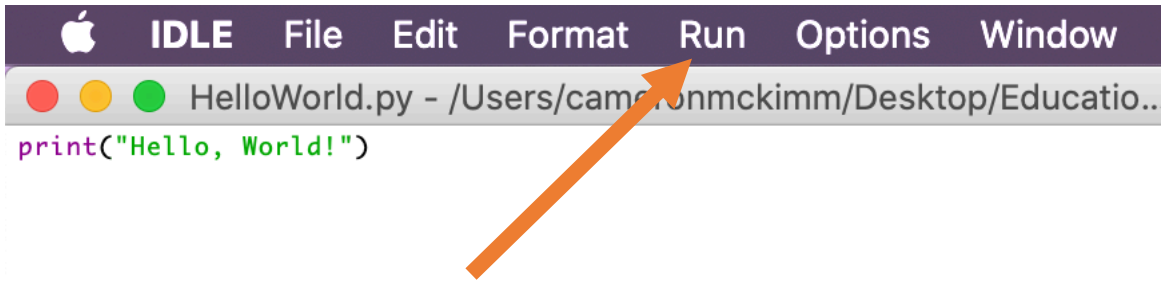


Fig 16

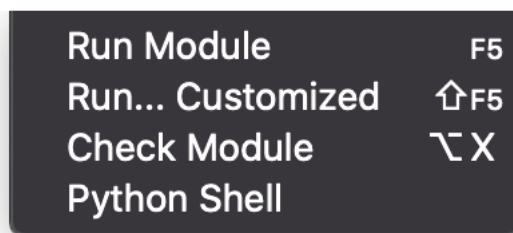


Fig 17

Finally, we have run our first scripted program which can be seen in the blue output of **Fig 18**.

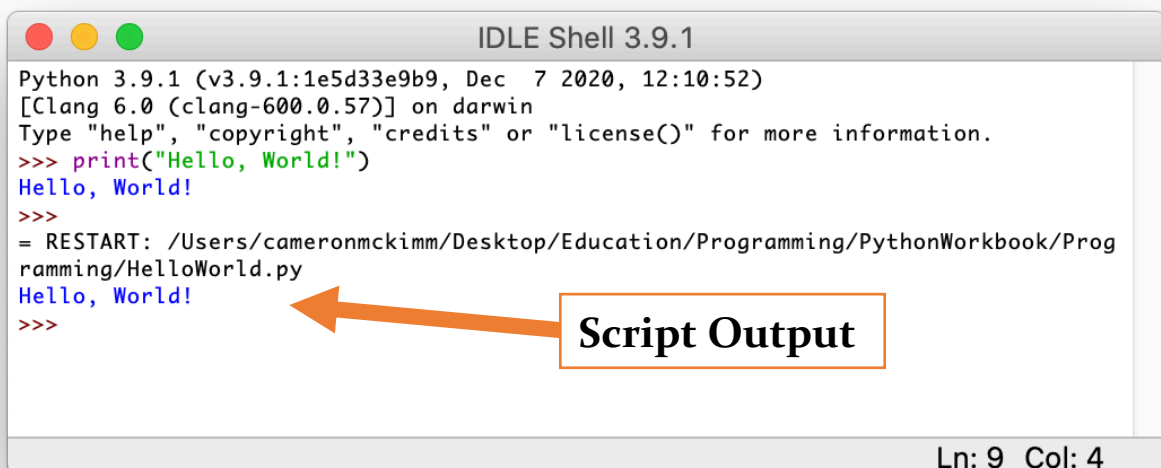


Fig 18

UNDERSTANDING THE HELLO WORLD PROGRAM

It is all well and good to get this program working, but why does it work? To understand this, we need to know about data types and functions, and their relationship.

DATA TYPES

A data type constrains the values that an expression, such as a variable or a function, might take. This data type defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored. There are a good number of data types, but for the meantime, we will only concern ourselves with the following datatypes:

Data type	Description	Sample data
INTEGER	Stores positive or negative whole numbers	17
REAL / DOUBLE	Stores numbers that contain decimal places/values and can also store integers	17.65
CHARACTER	Stores a single character which can be a letter, number or symbol	\$
STRING	Stores alphanumeric combinations and text. A string is a group of characters stored together as one. Numbers to be used for calculations should not be stored as string data even though they can be. They should be stored as INTEGER or REAL	Elephant OR \$4m%
BOOLEAN	Stores True or False only. This is sometimes taught as 1 or 0 only where 1 is true and 0 false	True

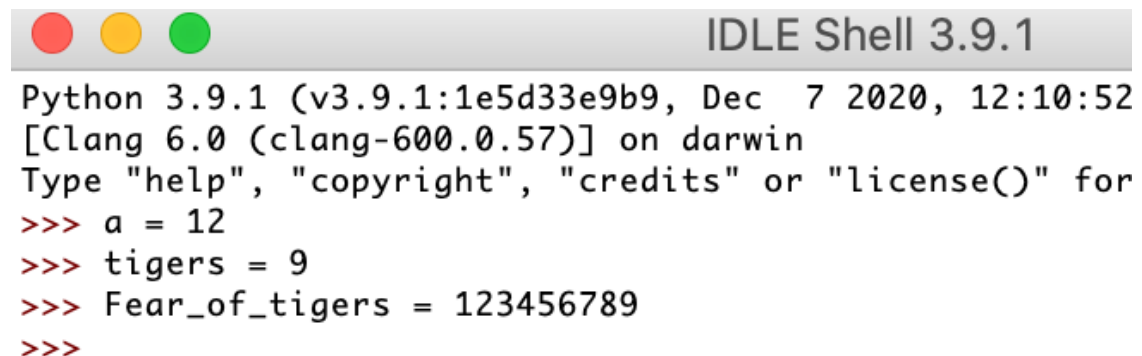
Hence, we can store these data types in python with *variable names*. A variable is a storage location with an associated identifier. Let's see some examples of variables and the data types mentioned above.

To begin, we will look at the INTEGER data type. In Python 3, the plain type integer is unbounded. However, in Python 2, the maximum value of the plain type integer is available as `sys.maxint` from the console. In **Fig 19**, we can see that I have declared three variables: `a`; `tigers` and `Fear_of_tigers`. IDLE is smart as it recognises that I have created a variable and has done some work behind the scenes to store the corresponding variable values in a memory location in the computer which is accessible by the variable names which I have assigned to them.

Note: This method of storing variables in a memory location is not technically true for Python, however, it is easier (to some people) to think this way.

Python employs a different approach. Instead of storing values in the memory space reserved by the variable, Python has the variable refer to the value.

In fact, all variable names in Python are said to be references to the values, some of which are front loaded by Python and therefore exist before the name references occur.



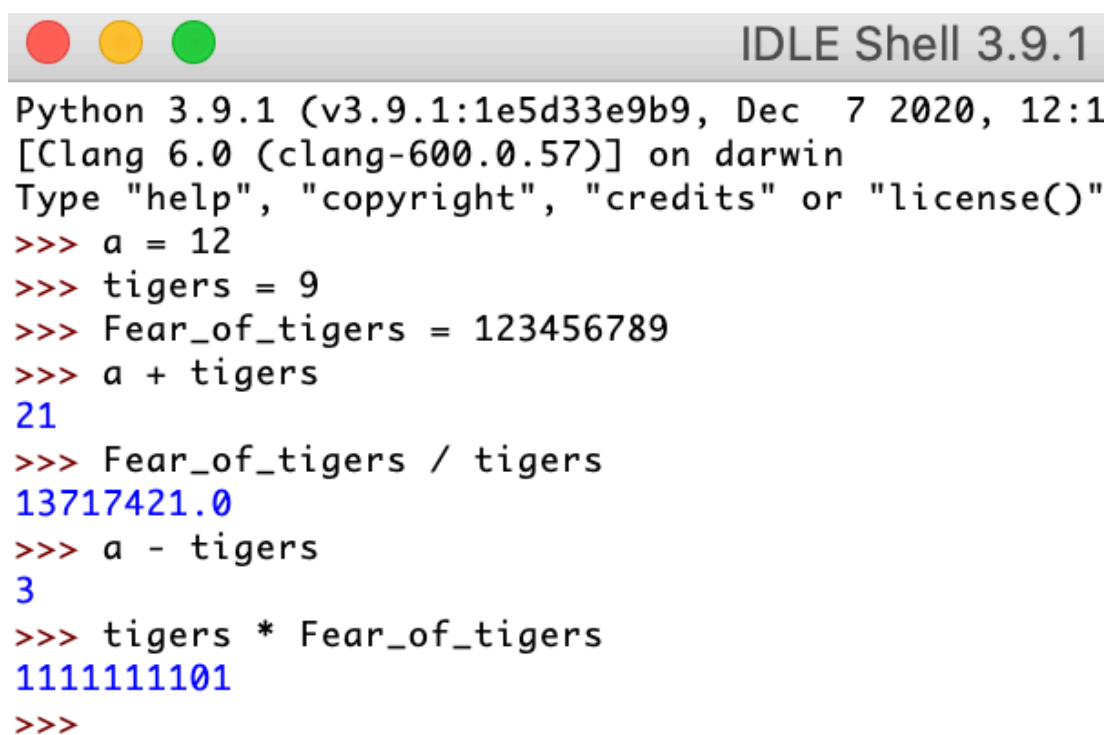
```
Python 3.9.1 (v3.9.1:1e5d33e9b9, Dec 7 2020, 12:10:52
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for
>>> a = 12
>>> tigers = 9
>>> Fear_of_tigers = 123456789
>>>
```

Fig 19

Notice how declarations of variables can be of any STRING type. It is usually an illegal operation to have space in the variable name. Instead, certain conventions are followed: Underscore instead of a space;

camelCase or TitleCase. There is a profusion of conventions which could be used to circumvent this scenario. It is entirely up to you what you use, however, if you are working as part of a team, ensure you are using the same convention as your team members, to keep everything clear and concise. Trust me, there is nothing more frustrating than unreadable code...

Now that we have the variables set, we can use them and manipulate them. Say we wanted to perform some arithmetic tasks; we can do so as shown in *Fig 20*.

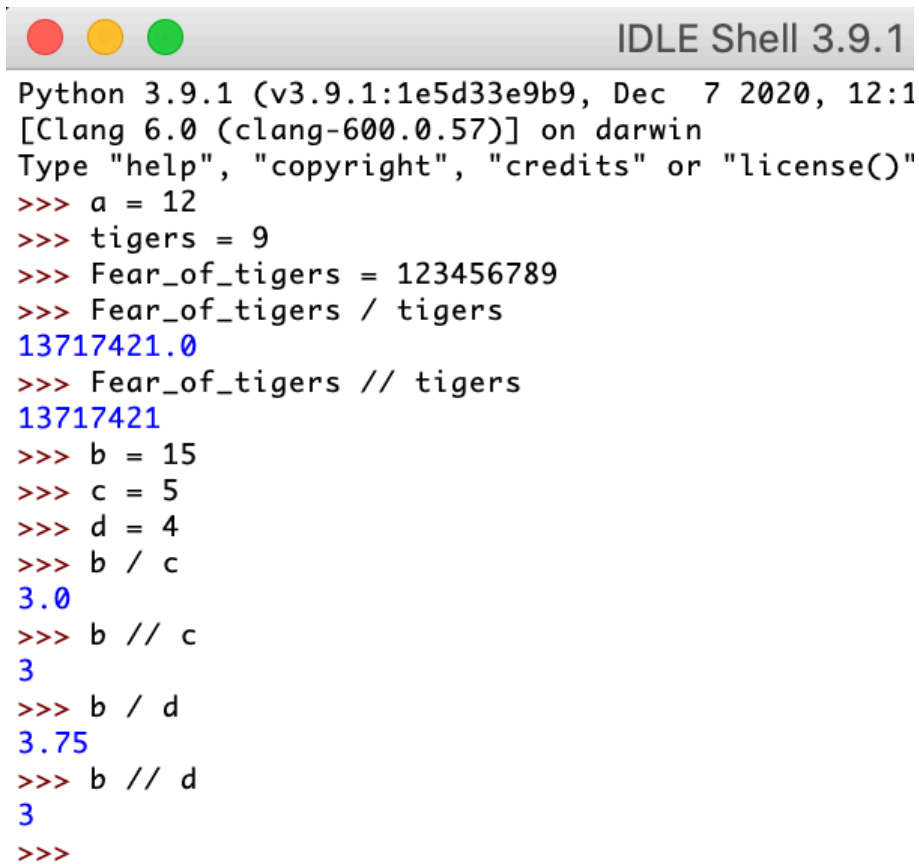


```
Python 3.9.1 (v3.9.1:1e5d33e9b9, Dec 7 2020, 12:1
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()"
>>> a = 12
>>> tigers = 9
>>> Fear_of_tigers = 123456789
>>> a + tigers
21
>>> Fear_of_tigers / tigers
13717421.0
>>> a - tigers
3
>>> tigers * Fear_of_tigers
1111111101
>>>
```

Fig 20

Furthermore, we need to understand the REAL or DOUBLE data type. This is any number with a decimal or fractional part. Notice in *Fig 20*, how $Fear_of_tigers / tigers = 13717421.0$, this has given a DOUBLE type answer instead of an INTEGER type. This is because we have used floating-point division instead of floor division.

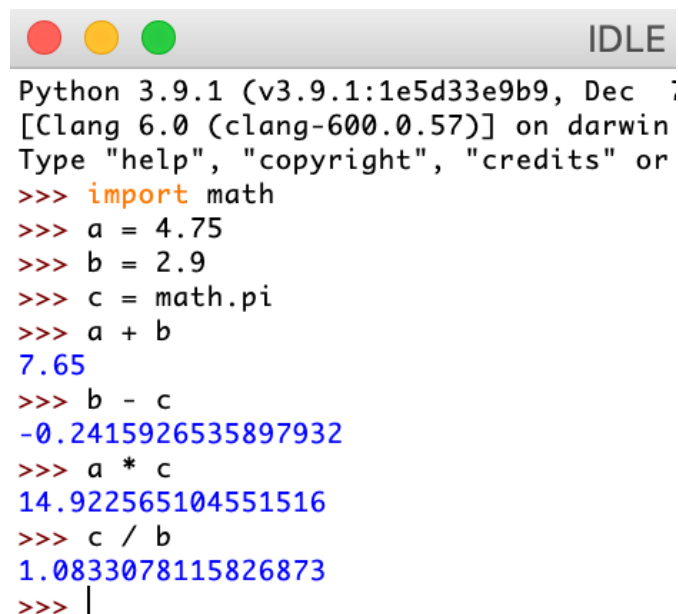
In Python 3.x, $5 / 2$ will return 2.5 and $5 // 2$ will return 2 . The former is floating-point division, and the latter is floor division, sometimes also called integer division. See in *Fig 21*, how our answers differ.



```
Python 3.9.1 (v3.9.1:1e5d33e9b9, Dec 7 2020, 12:1
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()"
>>> a = 12
>>> tigers = 9
>>> Fear_of_tigers = 123456789
>>> Fear_of_tigers / tigers
13717421.0
>>> Fear_of_tigers // tigers
13717421
>>> b = 15
>>> c = 5
>>> d = 4
>>> b / c
3.0
>>> b // c
3
>>> b / d
3.75
>>> b // d
3
>>>
```

Fig 21

We can also perform arithmetic tasks on values of DOUBLE type. As shown in *Fig 22*.



```
Python 3.9.1 (v3.9.1:1e5d33e9b9, Dec 7 2020, 12:1
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()"
>>> import math
>>> a = 4.75
>>> b = 2.9
>>> c = math.pi
>>> a + b
7.65
>>> b - c
-0.2415926535897932
>>> a * c
14.922565104551516
>>> c / b
1.0833078115826873
>>> |
```

Fig 22

We shall talk about CHARACTERS and STRINGS concurrently. You need to know, at the most basic level, computers work with numbers. To work with Characters, a translation scheme needs to be used, to distinguish between the numbers that a computer uses and the characters that humans understand. One type of translation scheme is called the **American Standard Code for Information Interchange (ASCII)**. It is well known for being a simple technique which is in common use. Have a look at *Fig 23*, we can see what ASCII codes are used for certain characters by using the `ord()` function.



```
Python 3.9.1 (v3.9.1:1e5d33e9b9;
[Clang 6.0 (clang-600.0.57)] on
Type "help", "copyright", "cred-
>>> ord("a")
97
>>> ord("5")
53
>>> ord(":")
58
>>> ord(" ")
32
>>>
```

Fig 23

There is a downside when using ASCII because it is limited and only covers the Latin characters which you are probably accustomed with. With this in mind, we also need to cover other languages in the world with many more characters than what ASCII can handle. Hence, we make way for Unicode. It is an attempt to provide a numeric code to cover every possible character, in every possible language, in every possible platform.

Resource: If you want to know more about ASCII check out:
<http://www.asciitable.com/>

You might come across *UTF-8* and *UTF-16*, which are common forms of Unicode character encodings.

Generally, if you aren't using characters outside of your usual character set, ASCII is perfectly fine for the job. But it is always good to be prepared, just in case.

Now that we know about character encodings, we can discuss STRINGS. Strings are simply a collection of one or more characters: these do include the alphabet, spaces and much more! See below in *Fig 24* how a string is structured. Notice that there are comments in the code. To write a comment in Python, we simply put a *#* symbol before our comment. Comments (in programming languages) are bits of text which are ignored when programs are running, they are only to be read by the reader of the code and there should not be an attempt to execute them.

```
>>> # This is a comment.
>>>
>>> # Strings can be stated using " or ' symbols before and after the text.
>>> "hello!"
'hello!'
>>> 'world'
'world'
>>> # Strings can be assigned to variables.
>>> a = "Bob"
>>> a_string_with_strange_text = '1 4m 4 57r4ng3 57r1ng'
>>>
```

Fig 24

Python is special, as it can manipulate strings much more than some other programming languages. Let's have a look at some of the basic operations in *Fig 25*.

```

>>> word = "Hello, World!"
>>> word[0] #get the first character of the string
'H'
>>> word[0:3] #get the first three characters of the string
'Hel'
>>> word[:3] #get the first three characters of the string (same as above)
'Hel'
>>> word[-3:] #get the last three characters of the string
'ld!'
>>> word[3:] #get all but the three first characters of the string
'lo, World!'
>>> word[:-3] #get all but the three last characters of the string
'Hello, Wor'
>>> word + "Python is cool..."
'Hello, World!Python is cool...'
>>> word * 3
'Hello, World!Hello, World!Hello, World!'
>>>

```

Fig 25

The final data type that we will mention is BOOLEAN. I love Boolean operations as they are very useful to have. In programming, you often need to know if something is true or false, with this knowledge, you can properly implement the correct and valid operation for the task. In *Fig 26*, we can see that python evaluates our expressions and provides a true or false value accordingly.

```

>>> a = 3
>>> b = 5
>>> b > a # b is greater than a
True
>>> a < b # a is less than b
True
>>> b == a # b is equal to a
False
>>>

```

Fig 26

These Boolean operations are very useful when we look at “*if statements*” later on in this manual. We shall now look at the most common functions used in python.

FUNCTIONS AND METHODS

Functions are going to be one of the most useful tools to you during your programming career. Functions are called by the programmer; arguments are passed to it and the relevant results are returned to the user. Let's first look at the most common of functions, the `print()` command. We saw this used in the "Hello, World!" program. Print takes arguments inside the brackets and will print out the result to the console. See **Fig 27**.

```
>>> print("Hello, World!")
Hello, World!
>>> print(123)
123
>>> print([1,3,5])
[1, 3, 5]
>>>
```

Fig 27

In the above case, "Hello, World!", 123 and [1,3,5] were the arguments passed to the `print()` function.

This function can be very helpful when trying to debug your code, you can print out variables, return results and more, which can help you understand how your program is properly executing.

Next, we will look at the `min()` and `max()` functions. These do exactly what they say: find the minimum and maximum of their arguments respectively. We used both functions for a list of numbers and a string so you can see their effect, in **Fig 28**.

```

>>> data = [5,1,2,3,7,6,4]
>>> min(data)
1
>>> max(data)
7
>>> new_data = "Python Programming"
>>> min(new_data)
' '
>>> max(new_data)
'y'
>>>

```

Fig 28

The `min()` function when passed the `new_data` as an argument, returned that the minimum value was ' '. This simply means that a “space” is “*alphabetically*” before any of the letters in the alphabet in the ASCII table.

Programmers would also use this function quite a bit: `len()`. This is the length function which returns the number of elements in a list or the number of characters in a string. See how we can use it in **Fig 29**.

```

>>> len('Python')
6
>>> len([1,2,3,4,5])
5
>>>

```

Fig 29

Finally, for this section, we shall learn about basic python methods. The most familiar will be the `.lower()` and the `.upper()` methods. Notice how a method is in the form `object.method()`, this is a distinguishing factor between functions and methods. As shown below in **Fig 30**.

```

>>> a = "DoG"
>>> a.lower()
'dog'
>>> a.upper()
'DOG'
>>>

```

Fig 30

Additionally, another method which comes in useful is the `.strip()` method. If there is any whitespace at the beginning or the end of a string, then this method shall remove it. As we can see in *Fig 31*.

Note: The `.strip()` method does not remove space anywhere other than the beginning or end of a string, hence if there are spaces between words, they shall not be removed. Moreover, if you provide this method with an argument of characters then you can remove those instead of whitespace. Whitespace is simply the default removal for this method unless otherwise specified.

```
>>> b = "    Cats    and    Dogs    "
>>> b.strip()
'Cats    and    Dogs'
>>>
```

Fig 31

The `.replace()` method replaces a string with another string (it is case-sensitive). It can be useful for string manipulation if the text is incorrectly formatted. There are many uses for this method. See *Fig 32*.

```
>>> text = "some random text"
>>> text.replace("a", "c")
'some rcndom text'
>>> text.replace("m", "l")
'sole randol text'
>>> text.replace("om", "an")
'sane randan text'
>>>
```

Fig 32

The `.split()` method splits up a string into a list, the argument you pass to the method is the delimiter. If no argument is supplied, the default delimiter is a space character. To understand the full picture, see below in *Fig 33*.

```

>>> data = "Some text, you can see. Hello, here is some more text. And again."
>>> data.split()
['Some', 'text,', 'you', 'can', 'see.', 'Hello,', 'here', 'is', 'some', 'more',
'text.', 'And', 'again. ']
>>> data.split(",")
['Some text', ' you can see. Hello', ' here is some more text. And again. ']
>>> data.split(".")
['Some text, you can see', ' Hello, here is some more text', ' And again', '']
>>>

```

Fig 33

The opposite of the `.split()` method is the `.join()` method. It joins elements from a list into one string. You can specify a delimiter as an argument. If you don't specify a delimiter, it will concatenate all elements of the list together with no "space" or "character" in between each element. Finally, we can see this method being implemented before discussing the "Hello, World!" program. The `.join()` method is shown in *Fig 34*.

```

>>> list_of_words = ["Here", "is", "a", "list", "of", "words"]
>>> " ".join(list_of_words) # Here a space is used as the delimiter.
'Here is a list of words'
>>> "".join(list_of_words) # Here no delimiter is specified.
'Hereisalistofwords'
>>>

```

Fig 34

DISCUSSING THE HELLO WORLD PROGRAM

Now that we fully understand what data types, functions and methods are, we can now discuss your first program. The "Hello, World!" program is simply just a function passing a string as an argument and returning it to the console for the user to see. It is used as the steppingstone into the programming world – which from the outside, can look pretty daunting.

"Hello, World!" is a beautiful way to introduce new people to programming. It's short, simple and clean. That is why it is important to show this to the prospering programmer and not something like what you would see in the movies!

If you go on to learn any other programming languages (which hopefully you do!) this is the perfect place to start. It means that you know with confidence that your code can compile, load and run when you see the output to the console.

CONDITIONAL STATEMENTS

Conditional statements are one of the main structures of producing effective code. They perform different actions depending on whether a programmer-specified Boolean condition evaluates to true or false. Here's an example to help you visualise what we are talking about. See *Fig 35*.

```
>>> if (7 < 12):  
    print("That argument evaluated to true.")
```

```
That argument evaluated to true.  
>>>
```

Fig 35

Now that we are discussing conditional statements, we need to be aware of our spacing/indentation. After the colon (:), notice how the print statement is indented forward and is not in line with the *if statement*, this is intentional. Python is picky about this, so ensure your spacing is correct.

This *if statement* has evaluated that $7 < 12$ is true, so the program follows this path and prints out the string. If a condition is not true, as in *Fig 36*, then the body of the condition will not be executed.

```
>>> if (5 > 8):  
    print("This should not print as the condition is false.")
```

```
>>>
```

Fig 36

Notice that the console did not return any output, hence the body of the conditional statement was not executed.

Furthermore, we can create catching conditional statements by appending the *else statement* to the block of code. This is always used in tandem with the *if statement*, it is never used on its own. It will evaluate the *if statement*, if it returns false and no error is raised, then the *else statement* shall always execute subject to those conditions. As shown in **Fig 37**.

```
>>> if (3 > 9):
    print("This will not print as the condition is false.")
else:
    print("This else statement will run since the 'if' was false.")
```

```
This else statement will run since the 'if' was false.
>>>
```

Fig 37

We can also see in **Fig 38**, that if the *if statement* evaluates to true, then the *else statement* will not be executed.

```
>>> if (4 < 8):
    print("This 'if' statement is true.")
else:
    print("This will not print as the 'if' statement is true.")
```

```
This 'if' statement is true.
>>>
```

Fig 38

Finally, we can also utilise *else-if statements*. In Python, these are called *elif statements* which is just a shortened name but operates in the same manner. These execute providing that the prior *if* or *elif statements* evaluated to false, and the current condition for the *elif statement* is true. As shown in **Fig 39**.

```
>>> if (3 < 1):
    print("This will not print.")
elif (3 > 1):
    print("This elif statement was true, so this message was printed")
else:
    print("This will not print.")
```

This elif statement was true, so this message was printed

Fig 39

FOR AND WHILE LOOPS

A loop is the repetition of a specific set of instructions until a given criterion is met. Programmers use loops to cycle through values, add sums of numbers, repeat functions and many other things.

For loops can be used in a variety of ways, they iterate usually until a counter has met a limit, a condition has become true or a whole host of other things. They are known as a type of **definite iteration** because we know when the loop will end.

The format of a *for loop* is structured like this:

```
for <variable> in <iterable>:
    <statement(s)>
```

<**iterable**> in this case is a list, <**statement(s)**> in the loop body are denoted by indentation and are executed once for each <**iterable**>.

See *Fig 40* below for an example where we iterate through a list and print out each element in the list. This is what you call a *foreach loop*, they are named as such because they iterate through each element in the list from start to finish.

```
>>> food = ["banana","apple","kiwi","strawberry","grapes"]
>>> for fruit in food:
    print(fruit + " is in our list.")
```

```
banana is in our list.
apple is in our list.
kiwi is in our list.
strawberry is in our list.
grapes is in our list.
>>>
```

Fig 40

These are very useful for a plethora of situations. Imagine you had a bunch of student scores and you wanted to find the average score for the class. We can program a quick for loop to help us, as shown in *Fig 41*.

Note: These programs are becoming longer and there is more logic happening, so we stop programming in the console and begin scripting our programs.

```
student_scores = [90,23,54,71,64,38,84,67,86]
sum_of_scores = 0
counter = 0

for score in student_scores:
    sum_of_scores = sum_of_scores + score # Add score to the total
    counter = counter + 1 # Count the number of scores added

average = sum_of_scores // counter # Calculate the average, as loop is finished
print(average)

= RESTART: /Users/c
programming/average.py
64
>>>
```

Fig 41

Another type of *for loop* is the *numeric range loop*. These are used to specify a beginning and endpoint for the iteration to occur. The *range()* function is used with the arguments in the order: *start_number*, *end_number*, *step_number*[*this is optional!*]. Imagine you wanted to add

up all the even numbers from 0 to 100 and find the total. We can program a quick *for range loop* to help us, as shown in **Fig 42**.

```
total = 0

for number in range(0,100,2): #Start at 0, End at 100, Step 2 each iteration.
    #Below is the same as: total = total + number
    total += number # This is SHORTHAND, but does the same job.

print("Total of even numbers: " + str(total)) # str() converts total to String

= RESTART: /Users/cameronmckimm/I
gramming/evenSumNumbers.py
Total of even numbers: 2450
```

Fig 42

Finally, the last loop we will look at is the *while loop*. These are known as a type of **indefinite iteration**, as we don't necessarily know when the loop will end.

The format of a *while loop* is structured like this:

```
while <expression>:
    <statement(s)>
```

When a *while loop* is encountered, the expression is evaluated in a Boolean context, if the expression is true, the loop body is executed, the expression will then be checked again and so long that it is true, it will continue to cycle through this process. If the expression is false, the loop body will not execute, and the rest of the program can continue.

Imagine you wanted to see the numbers from 30 to 0 countdown. We can create a simple *while loop* to handle this problem. See below in **Fig 43**.

```
number = 30

while number != 0: # != means 'does not equal'
    print("Current number is: " + str(number))
    number -= 1 # Decrement by 1, same as number = number - 1
```

```
= RESTART: /Users/cameroc  
gramming/countdown.py  
Current number is: 30  
Current number is: 29  
Current number is: 28  
Current number is: 27  
Current number is: 26  
Current number is: 25  
Current number is: 24  
Current number is: 23  
Current number is: 22  
Current number is: 21  
Current number is: 20  
Current number is: 19  
Current number is: 18  
Current number is: 17  
Current number is: 16  
Current number is: 15  
Current number is: 14  
Current number is: 13  
Current number is: 12  
Current number is: 11  
Current number is: 10  
Current number is: 9  
Current number is: 8  
Current number is: 7  
Current number is: 6  
Current number is: 5  
Current number is: 4  
Current number is: 3  
Current number is: 2  
Current number is: 1  
>>>
```

Fig 43

THANK YOU FOR READING!

Well, that was a rollercoaster of fun!

In all seriousness, I hope this short manual has helped you in your quest to learning how to program. This is the first time I've ever written a document like this teaching a topic, so I hope that you have found it beneficial.

Python was my very first programming language and now I'm up to 10 total languages! I have thoroughly enjoyed learning about Computer Science and Programming from a very early age, so I am proud that I can produce this type of work.

Keep in mind, this manual is to give you the **fundamental knowledge** for learning any programming language; Python just so happens to be the one used here. You can take the concepts, such as *Conditional Statements* and *Data types*, and learn how they are implemented in other languages.

I do definitely recommend that you don't restrict yourself to just this manual, do read up on the topics that I have discussed in here. If you do become stuck, google is your friend. Use it. You will come across errors during your programming, just don't be afraid to make those mistakes as that is the best way to learn.

If you have any suggestions, criticisms or tips, I'd love to hear from you!

“Sometimes, it is the people who no one imagines anything of, who do the things no one can imagine.”

- Alan Turing