



# Python for Beginners



**PYTHON TRICKS**

BY LEARN SPACE TUTORIALS

Hira Mariam

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.



Authored by: Hira Mariam

## Contents

1 Introduction.....	7
Background.....	7
Why Python? .....	7
What Python is used for? .....	7
Who this book is for? .....	8
2 Setting Up Your Python Environment .....	9
Running Python on Windows Command Prompt:.....	9
Writing Your First Hello World Program on Windows .....	11
Running Python on OS X Terminal: .....	11
Writing Your First Hello World Program on OS X: .....	12
Running Python Externally on a Web Browser:.....	13
3 Variables and Data Types.....	14
Variables.....	14
Rules to Use Variables.....	15
Strings.....	15
Printing a Substring .....	16
Removing Whitespace From Strings .....	16
Changing Letter Cases in Python.....	17
Finding the Length of the String.....	17
Combining or Concatenating More Than One String .....	17
Adding Newlines and Tabs in Python .....	18
Numbers.....	18
Comments.....	19
The Zen of Python.....	20
4 Comments, Indentations and Statements .....	21
Python Comments and Statements.....	21
Multiline Python Statement.....	21
Multiple Python Statement in One Line .....	22
Strings Python Statements .....	23
Blank Lines Python Statements .....	23
Python Indentation.....	23
Python Comment.....	24
Multiline Python Comment.....	24
Indentation.....	25
Conclusion .....	25

5 Basic Operators .....	26
Arithmetic Operator.....	26
Comparison Operator .....	28
Assignment Operator .....	30
Logical Operator .....	31
Bitwise Operator .....	31
Special Operators .....	32
6 If Else and Elif Statements .....	33
If Statement.....	33
Else Statement.....	33
Elif Statement .....	34
7 Loops.....	35
While Loop .....	35
Syntax of While Loop.....	36
While Loops with Else:.....	36
For Loop.....	37
Syntax of For Loop.....	37
Using the range() function in for Loop.....	38
Using the if-else statements with For Loop: .....	39
Nested Loops .....	40
Syntax of Nested Loops: .....	40
8 Control Statements .....	41
Break Statement.....	41
Syntax.....	41
Continue Statement.....	42
Syntax.....	42
Pass Statement.....	43
Syntax.....	43
9 Number Types.....	45
Integers in Python.....	46
type() function.....	46
isinstance() .....	46
Exponential numbers.....	47
Float in Python .....	47
Complex Numbers in Python.....	47
Coefficient to the imaginary part.....	47
Operations on complex numbers .....	48

10 Strings .....	49
What are Strings .....	49
How Strings are Indexed? .....	50
Slicing a String .....	50
Positive and Negative Indexing of Strings .....	50
Updating a String Through Slicing .....	51
Deleting a String .....	51
String Operations in Python .....	51
The Concatenate Operation .....	51
Using Iteration in Strings .....	52
Testing Sub Strings within a String .....	52
Built-in Functions in Strings .....	53
Formatting Strings in Python .....	53
How to Ignore Escape Sequence? .....	54
11 Lists .....	55
Indexing and Splitting in Lists .....	55
Updating an Item in List .....	56
Adding elements to the list .....	57
Iterating a List .....	57
12 Tuples .....	58
Using a For Loop in Tuples .....	58
Adding Items to Tuple .....	58
Negative Indexing in Tuple .....	59
Changing Tuple Values .....	59
13 Dictionaries .....	60
Accessing Elements inside a Dictionary .....	60
Changing Elements inside a Dictionary .....	61
Using Loop to print keys and values .....	61
Removing Items from a Dictionary .....	62
14 Functions .....	63
How to define a Function? .....	63
How to Call a Function? .....	64
Adding Parameters in Function Calling .....	64
Setting a Default Parameter .....	64
*args or Multiple Arguments .....	65
**kwargs or Multiple Keyword Arguments .....	65

15 Modules .....	66
What is a Module?.....	66
Importing a Module.....	66
'From' in Import Statement .....	67
Renaming a Module using Alias.....	67
Built in Modules.....	68
16 File Handling .....	70
File in Python.....	70
Opening the File .....	70
Reading or Writing the File .....	71
Reading Parts of File.....	71
Reading Lines of File .....	71
Closing the File .....	72
Writing to a File .....	72
Creating a New File .....	72
Appending to A File.....	73
Deleting a File.....	73
17 Exception Handling.....	74
What is Error?.....	74
What is Exception Handling? .....	74
ZeroDivision Exception .....	75
FileNotFoundError Exception .....	75
Try-Except Code Blocks.....	75
Else Block .....	76
Finally Exception .....	77
More Built-in-Exceptions .....	77
Conclusion.....	79

*To Baba,*

# 1

## Introduction

### ***Background***

Python was created by Guido van Rossum and was released in 1991. It focuses on the philosophy of code readability and emphasizes on the importance of whitespace. It is used for writing both small- and large-scale programs.

### ***Why Python?***

Python is an easy to learn, readable and powerful programming language that focuses on the the high-level data structures in a simple but effective perspective towards object oriented programming. Due to its syntax and easily interpreted nature, it is the most favorite programming language of programmers for the past few decades. The freedom given by python makes it easy for the developers to not just create applications but also enable to generate and analyze data through its famous extensive libraries such as Numpy, Pandas and Matplotlib.

This guide introduces the reader informally about the basic concepts and features in Python language to have a hands-on experience with python by keeping the self-paced learner in thought to get started. This guidebook doesn't offer a detail explanation of Python language but is sufficient enough to provide with the basics of Python.

### ***What Python is used for?***

Python is used for many purposes such as:

- Web development
- Computing scientific and numeric data
- Creating GUI (Graphical User Interface) applications
- Software development
- Creating, managing business applications

It is viable to say that Python is an all-rounder programming language because of its versatile and accessible nature. Companies like Google, NASA, Yahoo! among many others are using Python, it is no



wonder that Python is the most favorite language of Data Scientists as it is widely used in Artificial Intelligence projects as well.

### ***Who this book is for?***

So, if you are a newbie, this guidebook is for you. After going through this guidebook, you will be able to write Python programs and import modules. Also, generally you must learn about programming if you want to automate things for yourself. Let's say that you have an unordered list and you are having a tough time arranging it. You can write a code and automate your list to be in order. Although, you can do plenty of things like listing, classifying data, looping, adding functions etc.

# 2

## Setting Up Your Python Environment

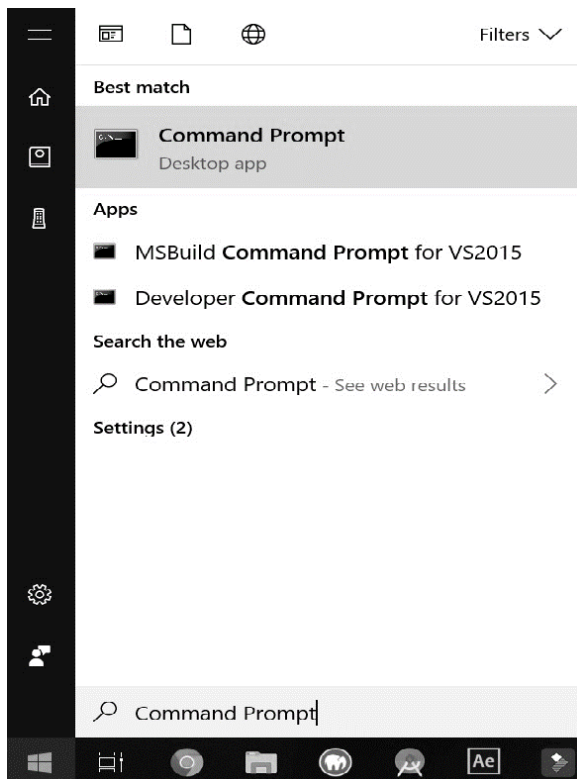
If Python is pre-installed on your system then you just have to open **Command Prompt** in Windows and **Terminal** application in MAC OS X. All you need to do is write a few lines of code and you are good to go

### *Running Python on Windows Command Prompt:*

In order to run Python code on your Windows Command Prompt then:

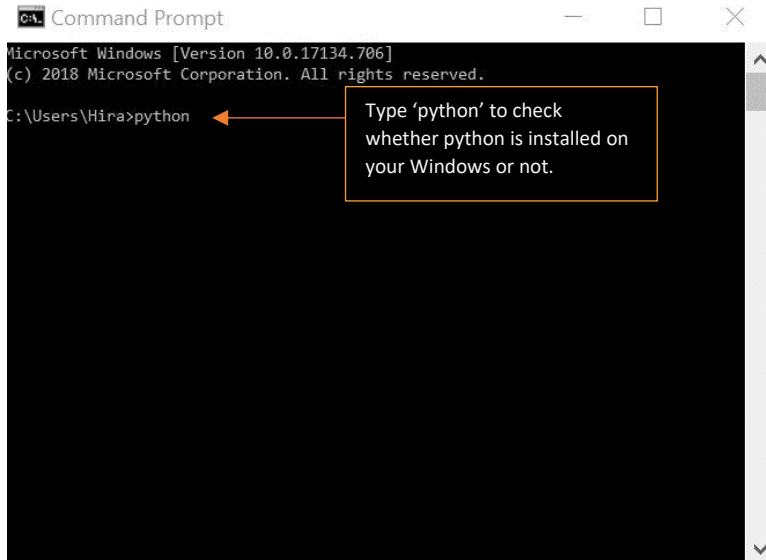
#### STEP 1:

Open Command Prompt on windows, once command prompt is opened



#### STEP 2:

Type **'python'** inside the cmd(Command Prompt) shell:



```

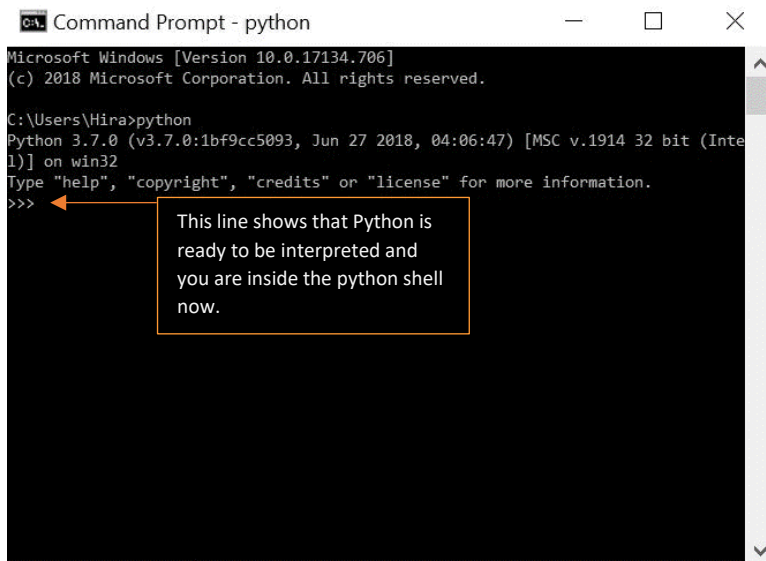
C:\Users\Hira>python

```

Type 'python' to check whether python is installed on your Windows or not.

**STEP 3:**

Once you type 'python' then press 'Enter', the screen will look like this:



```

C:\Users\Hira>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>

```

This line shows that Python is ready to be interpreted and you are inside the python shell now.

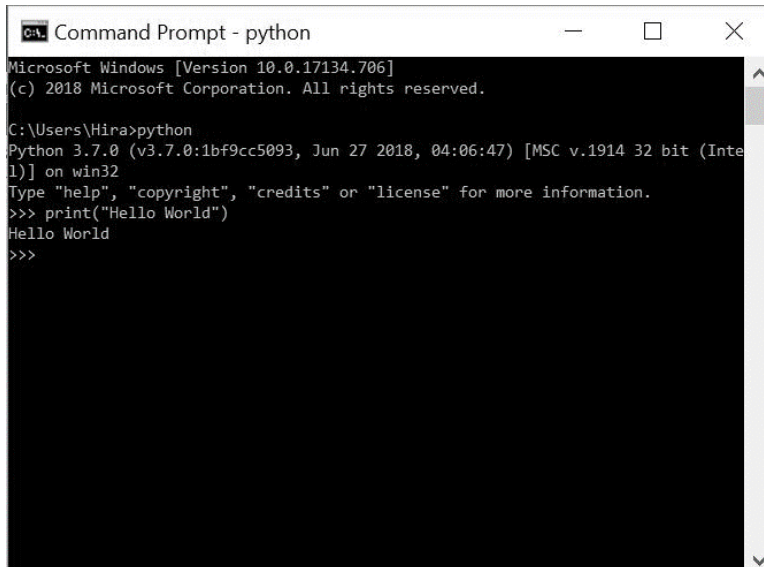
If python is present on your system then you don't have to download it, you can easily get started to try bits of python code, however, if Python is not present on your system then you might want to take a watch this tutorial [here](#) to get a beginner-friendly python environment along with a code editor that will enhance your options to code in Python.

## Writing Your First Hello World Program on Windows

As we know that python comes with an interpreter that runs on command prompt which allow you to try bits of python code. In this section, we are going to write a simple hello world program which is going to display “**hello world**” as an output. Once you are inside the command prompt and you have initiated python as interpreter then simply type:

```
>>> print("Hello World!")
Hello World
```

The output will look like this:



```
Command Prompt - python
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Hira>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

The purpose of writing this simple hello world program is that if it runs correctly on your system then any Python program could work just fine. We will look into how to write the same program on OS X.

## Running Python on OS X Terminal:

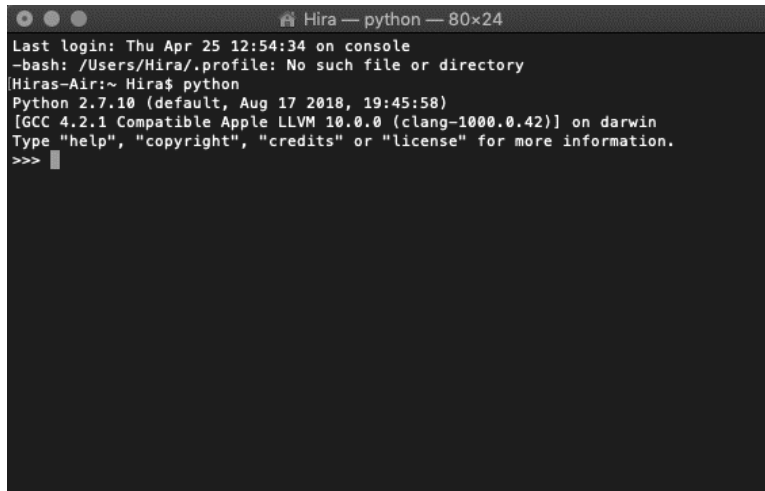
Python is already installed on most OS X systems. You just need to figure out whether the latest version of Python is installed on your system or not. In order to check which of python is installed on your machine,

### STEP 1:

Go to **Applications > Utilities > Terminal**. You can also press *cmd + spacebar* and type ‘**Terminal**’ inside the spotlight search bar.

**STEP 2:**

Once the terminal opens, type 'python' and press Enter. Your screen will show you the installed version of python on your system.



```

Hira — python — 80x24
Last login: Thu Apr 25 12:54:34 on console
-bash: /Users/Hira/.profile: No such file or directory
Hiras-Air:~ Hira$ python
Python 2.7.10 (default, Aug 17 2018, 19:45:58)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.0.42)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>

```

Since we want to work with the latest python version, watch how to download the latest version of python and install a code editor to interpret it [here](#).

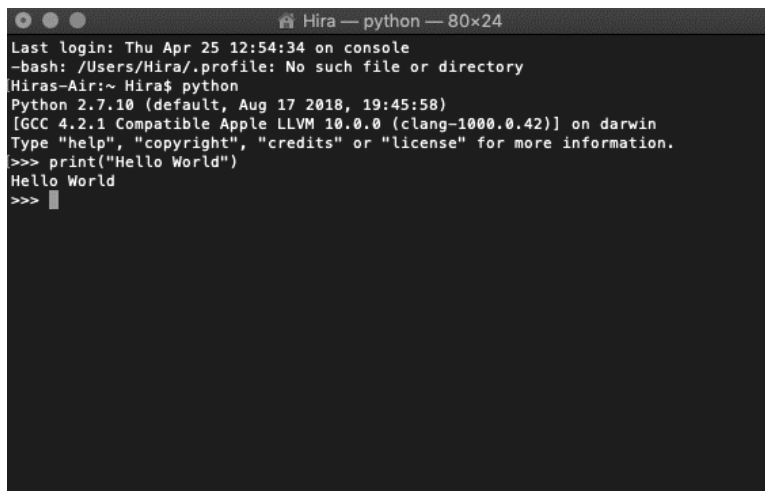
***Writing Your First Hello World Program on OS X:***

```

print("Hello World!")
Hello World

```

You will see your output rightly printed on the screen once you press Enter.



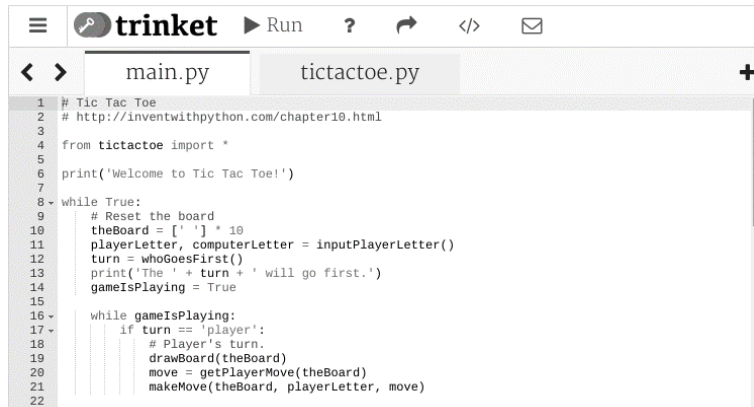
```

Hira — python — 80x24
Last login: Thu Apr 25 12:54:34 on console
-bash: /Users/Hira/.profile: No such file or directory
Hiras-Air:~ Hira$ python
Python 2.7.10 (default, Aug 17 2018, 19:45:58)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.0.42)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>

```

## Running Python Externally on a Web Browser:

If you want to skip installing python on your system then you can simply run python on your web browser by the most preferred online python editor known as [trinket](#)



```

1 # Tic Tac Toe
2 # http://inventwithpython.com/chapter10.html
3
4 from tictactoe import *
5
6 print('Welcome to Tic Tac Toe!')
7
8 while True:
9     # Reset the board
10    theBoard = [' '] * 10
11    playerLetter, computerLetter = inputPlayerLetter()
12    turn = whoGoesFirst()
13    print('The ' + turn + ' will go first.')
14    gameIsPlaying = True
15
16    while gameIsPlaying:
17        if turn == 'player':
18            # Player's turn.
19            drawBoard(theBoard)
20            move = getPlayerMove(theBoard)
21            makeMove(theBoard, playerLetter, move)
22

```

Trinket supports Python 3 which makes it an exclusive online code playground for python. Its enormous support for python libraries among NumPy, Pandas and Matplotlib makes it easy to compute scientific data.

# 3

## Variables and Data Types

### *Variables*

We have previously created a hello world program and saved it as a 'hello\_world.py' file in our projects file directory and now it's time to add a variable to our hello world program. Just add a new line to the beginning of the program and store it in a random variable

For Example:

```
result = "Hello World"
print(result)
```

OUTPUT:

```
Hello World
```

Run this program and you will see the same result as our first hello world program. But this time we have stored our string inside a variable named 'result'. A variable is holding the value associated to it. In this case, the 'result' variable is holding the "hello world" value. The advantage of using variable in your code is that you can use the variable and associate to more than one value at a time.

For Example:

```
result = "Hello World"
print(result)
```

```
result = "Hello World, this is second line"
print(result)
```

OUTPUT:

```
Hello World
Hello World, this is second line
```

If you print the variable multiple times just like the code above then python takes the code line by line and prints the value accordingly. But if the variable has more than one value associated to it and if you only print the variable once, then python is going to print the latest value the variable is holding.

For Example:

```
result = "Hello World"
result = "Hello World, this is second line"
print(result)
```

OUTPUT:

```
Hello World, this is second line
```

## Rules to Use Variables

When you use variables, you need to follow a set of rules and guidelines while writing them to avoid errors. These rules will help you to write your code correctly that is easier to read and understand.

- Variables cannot be started with a number, you can start a variable with an alphabet, underscore. The variable can contain a number but cannot start with it. For example, you can write `'formula_1'` but not `'1_formula'` otherwise error will be generated.
- Using spaces in your variable name will generate an error, it is better to use underscores if you want to show some break in your variable names. For example, `'final result'` will cause an error but `'final_result'` won't.
- Since Python has built-in functions and keywords so don't use those functions or keyword names as those names are already reserved, if you try to use keyword or function names as your variable names then you will encounter an error. For example, the keyword `'print'` is reserved for printing only, you cannot use it as a variable name.
- It's better to use lowercase letters in python, you won't get any error if you use uppercase letters, but it is always a good practice to use lowercase letters.

It will take some time for you to understand using variables rightly especially when your programs become bigger and more complicated. You have to come up with name that are unique, easily readable and short but descriptive. Remember the code you will write is not just limited to your code editor, maybe you want to share it with other coders in the future. So, you have to make sure that your code is easy to understand.

## Strings

A string is simply a set of characters. Anything that is inside single or double quotes is called a string.

For Example:

```
print('I am a string')
print("I am also a string")
```

OUTPUT:

```
I am a string
I am also a string
```

Depending on the type of sentence that you want to write, you can use single and double quotes together.

For Example:

```
print('He said, "I am not going to make it today"')
```



```
print("The 'tomatino' festival of France serves tones of tomatoes")
```

OUTPUT:

```
He said, "I am not going to make it today"
The 'tomatino' festival of France serves tones of tomatoes
```

## Printing a Substring

You can print a part of a string as well. Let's say that you want to print a few words out of the string but not the whole string.

For Example:

```
print("It's raining cats and dogs")
```

Now, let's say that out of the whole string you want to only print *cats and dogs*, so basically you are printing a **substring** which is a segment of a string, in order to print cats and dogs, you will get the characters position and slice the string accordingly

```
a = ("It's raining cats and dogs")
print(a[13:])
```

OUTPUT:

```
cats and dogs
```

The position '13' defines that the string will start printing from the position 'c' and goes till the very end. Note: you don't have to define an end character position when you are printing the substring till the very end.

## Removing Whitespace From Strings

Sometimes while creating a string, we create unwanted spaces which may affect the output. So in order to minimize this issue, python has a method known as `strip()` which eliminates all the white spaces from left and right.

For Example:

```
c = "      I love Python      "
print(c.strip())
```

OUTPUT:

```
I love Python
```

## ***Changing Letter Cases in Python***

The upper() method will print all the letters in the string in uppercase

```
For Example:
c = "I love Python"
print(c.upper())
```

```
OUTPUT:
I LOVE PYTHON
```

The lower() method will print all the letters in the string in lowercase

```
c = "I love Python"
print(c.lower())
```

```
OUTPUT:
i love python
```

The title() method will print the first letter of the words present in a string in an uppercase letter.

```
c = "I love Python"
print(c.title())
```

```
OUTPUT:
I Love Python
```

## ***Finding the Length of the String***

You can find the length of the string by using the len() method

```
c = "I love Python"
print(len(c))
```

```
OUTPUT:
13
```

## ***Combining or Concatenating More Than One String***

So far, we have learned about what is a string and substring, how to write it, apply different methods to it etc. Now, we will learn about how to combine two strings together. Let's us look at:

```
For Example:
String1 = "I love Python"
String2 = "I love Strings"
```

Now in order to combine these two strings, we are going to use '+' between them to print the final output

```
Final_result = String1 + String2
print(Final_result)
```

```
OUTPUT:
I love PythonI love Strings
```

The combining method of strings is called concatenation in Python and programming in general.

### ***Adding Newlines and Tabs in Python***

We know the concept of adding newlines and tabs in a simple word document, we can do the same in python as well. In order to add a new line in python string, all you need to do is add '\n' backslash and 'n' which stands for newline.

For Example:

```
c = "I love Python\nand I love Strings"
print(c)
```

OUTPUT:

```
I love Python
and I love Strings
```

Now in order to add a tab in python string then you need to add a '\t' backslash and a 't' which stands for tab, it behaves similar to the function of tab key present on the keyboard to give indentation to the text.

For Example:

```
b = "March is: \n\t-Beautiful\n\t-Cool\n\t-Colorful"
print(b)
```

OUTPUT:

```
March is:
    -Beautiful
    -Cool
    -Colorful
```

### ***Numbers***

Numbers are used in programming language to perform scientific calculations, arithmetic operations or just doing simple math. There are two main types of numbers used in python:

- Integers
- Float

You can apply simple arithmetic operations on integers like addition, subtraction, multiplication, division in Python.

```
add = 4+2
subtract = 3-2
multiply = 4*3
divide = 12/2
```

```
print(add)
print(subtract)
print(multiply)
print(divide)
```

OUTPUT:

```
6
1
12
6.0
```

You can apply the same arithmetic operations to float numbers as well but float values will always have a decimal. For example, 3.12 and 0.0003 are all float numbers.

## Comments

Writing comments is very important in python as it enables coder to take notes and bookmark the code especially when the code gets bigger and complicated. Note: A comment is never part of the code but it just hints the coder about a particular block of code to classify. You can write comments in multiple ways in Python:

Use hashtag (#) in the beginning of the comment, the line will be treated as a comment in python.

```
#This is a comment
```

Using triple single quotes (""") in the beginning and at the end is also used to write comments in Python.

```
'''This is a comment'''
```

You can use docstrings in python as well, A docstring is simply a multi-line string, that is not assigned to anything. It is specified in source code that is used to document a specific segment of code. You can represent a docstring by using triple double quotes in the beginning and at the end of the string.

```
"""This is a docstring"""
```

## The Zen of Python

One of the coolest features of python is its libraries because without these libraries Python won't be so powerful. We are just starting off in Python, but there are going to be times, when the python gets bigger and complicated so it needs external tools to calculate massive calculations. We won't be importing any major libraries at the moment, but just to get a taste of how importing works in Python, we will be using a simple *import this* in our code editor to see a strange output:

```
hello_world.py x
56
57 import this
58
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

# 4

## Comments, Indentations and Statements

In this tutorial we are going to revise the basic syntax of python along with the addition of Statements and Comments along with some examples.

### *Python Comments and Statements*

```
>>> a*=3
```

### *Multiline Python Statement*

In Python, every statement ends with a newline character. But like we have seen, it is possible to span a statement over multiple lines. We do this using the `\` character.

```
>>> a=\
    10\
    +20
>>> a
```

OUTPUT:  
30

```
>>> "Hi\  
There"
```

OUTPUT:  
'HiThere'

But you can also use a set of parentheses for this.

```
>>> b=(  
        10+  
        20)  
>>> b
```

OUTPUT:  
30

## ***Multiple Python Statement in One Line***

You can easily put multiple statements in python on one line.

```
>>> a=7; print(a)
```

You can also do this to an if-statement.

```
>>> if 2>1: print("2")
```

## ***Strings Python Statements***

To declare strings in python, you may use single or double quotes.

```
>>> "Hello 'user'"
```

OUTPUT:  
Hello 'user'

If you use double quotes outside, use single quotes wherever you need to use a quote inside.

## ***Blank Lines Python Statements***

The interpreter simply ignores blank lines.

## ***Python Indentation***

Unlike C++ or Java, Python does not use curly braces for indentation. Instead, Python mandates indentation. At this point, our inner monsters are laughing at the lazy programmers around us.

```
>>> if 2>1:
    print("2")
```

OUTPUT:  
2

There are no strict rules on what kind of Python indentation you use. But it must be consistent throughout the block. Although, four whitespaces are usually preferred, and tabs are discouraged. Let's take an example with an inconsistent indentation in python.

```
>>> if 2>1:
    print("1")
    print("2")
```

OUTPUT:  
SyntaxError: unexpected indent



## Python Comment

Python Comment is a programmer's tool. We use them to explain the code, and the interpreter ignores them. You never know when a programmer may have to understand code written by another and make changes to it. Other times, give your brain a month's time, and it might forget what it once had conjured up in your code. For these purposes, good code will hold comments in the right places.

In C++, we have `//` for single-lined comments, and `/* ... */` for multiple-lined comments. Here, we only have single-lined python comment.

To declare a Python comment, use the octothorpe (hash) (`#`).

```
>>> #This is a comment
>>>
```

## Multiline Python Comment

To have multiline python comment in your code, you must use a hash at the beginning of every line of your comment in python.

```
>>> #Line 1 of comment
>>> #Line 2 of comment
>>> #Line 3 of comment
```

You can also use triple quotes (`''' '''` or `""" """`) for this purpose.

```
>>> """This comment
is spanned across
multiple lines"""
```

This comment is spread on multi lines. The only difference between a `#` and triple quotes is that `#` is always spanned on a single line, whereas if your comment is more than one line then you must use triple (`"""`) quotes at the start and at the end of the comment.

While triple quotes are generally used for multiline python comment, we can conveniently use them for python comment as well.

Triple quotes will also preserve formatting.

```
>>> print("""Hello  
Hi""")
```

## ***Indentation***

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

Generally, four whitespaces are used for indentation and is preferred over tabs. Here is an example:

```
for i in range(1,11):  
    print(i)  
    if i == 5:  
        break
```

OUTPUT:

```
1  
2  
3  
4  
5
```

## ***Conclusion***

Hope we have tried to connect you with the basics of how the comments, statements and indentations work in Python. You can try your own combinations to test further.

# 5

## Basic Operators

Python supports basic operations since it is built on logic, a number of operations can be applied. Python operators are divided into eight categories that you can use to do certain operations:

### *Arithmetic Operator*

Just as the name goes, arithmetic operators are used to do simple calculations like addition, subtraction, division, modulo, exponential power and multiplication

OPERATOR	FUNCTION	EXAMPLE
+	Add two numbers or operands	$x + y$
-	Subtract two operands	$x - y$
*	Multiply two operands	$x * y$
/	Divide two operands	$x / y$
//	Divide(floor) two operands	$x / y$
%	Modulus returns the remainder of the operands divided by each other	$x \% y$
**	Creates exponential power of the first operand	$x ** y$

```
#Taking two numbers a and b
a = 3
b = 5

# Addition
addition = a + b
# Subtraction
subtraction = a - b
# Division
division = a / b
# Multiplication
multiplication = a * b
# Exponential Power
exponential = a**b
# Modulo
modulo = a % b

# Printing the variables
print(addition)
print(subtraction)
print(multiplication)
print(division)
print(exponential)
print(modulo)
```

**OUTPUT:**

```
8
-2
15
0.6
243
3
```

## Comparison Operator

Comparison operators are used to compare values and return the result by checking the nature of the output as True or False

OPERATOR	FUNCTION	EXAMPLE
==	equals to sign, returns true when both the operands are equal to each other	x == y
>	Greater than sign, turns true when the left operand is greater than the right operand	x > y
<	Less than sign, turns true when the left operand is less than the right operand	x < y
>=	Greater than or equals to sign, turns true when the left operand is greater or equals to the right operand	x >= y
<=	Less than or equals to sign, turns true when the left operand is less than or equals to the right operand	x <= y
!=	Not equals to sign, turns true when both the operands aren't equal to each other	x != y

```
# Comparison operators
x = 10
y = 4

# When x > y
print(x > y)

# When x < y
print(x < y)

# When x == y
print(x == y)

# when x != y
print(x != y)
```

```
# when x >= y  
print(x >= y)
```

```
#when x <= y  
print(x <= y)
```

**OUTPUT:**

```
True  
False  
False  
True  
True  
False
```

## Assignment Operator

Assignment operators are used to assign values (operands) to the variables. There are a lot of assignment operators used in python:

OPERATOR	FUNCTION	EXAMPLE
=	Equals to, sets the variable equals to the operand.	x = 5
+=	Adds the right operand value with the left operand value and then assigns it to the left operand.	x += y x = x + y
-=	Subtracts the right operand value from the left operand value and then assigns to the left operand.	x -= y x = x - y
*=	Multiplies the right operand value with the left operand value and then assigns it to the left operand.	x *= y x = x * y
/=	Divides the right operand value with the left operand value and then assigns it to the left operand.	x /= y x = x / y
//=	Divides (floor) the right operand value with the left operand value and then assigns it to the left operand.	x //= y x = x // y
%=	Takes the remainder of the right operand value from the left operand value and then assigns it to the left operand.	x %= y x = x % y
**=	Calculate exponent(raise power) of left value with right operand and assign value to left operand	x **= y x = x ** y

## Logical Operator

Logical operators perform AND, NOT and OR functions between operands

OPERATOR	FUNCTION	EXAMPLE
and	Turns true if both the values are true	x and y
or	Turns true if one or both of the operands is true	x or y
not	Turns true if the operand is false	not x

## Bitwise Operator

As the name goes by, bitwise operators perform operations bit by bit.

OPERATOR	FUNCTION	EXAMPLE
&	AND, sets the bit to one if both the operands are one	x & y
	OR, sets the bit to one if one or both the operands are one	x   y
^	XOR, sets the bit to one only if one of the 2 operands is one, if numbers are identical then there is going to be a zero	x ^ y
~	Inverts all the bits and prints the compliment of the number	~x
<<	Shift the bits on the left side by adding number of zeros of the right operand	x << y
>>	Shift the bits on the right side by adding number of zeros of the left operand	x >> y



## Special Operators

- Identity operators: `is` and `is not` are used to see if two operands are present on the same memory location or not

OPERATOR	FUNCTION	EXAMPLE
<code>is</code>	Turns true if two operands are identical	<code>x is y</code>
<code>is not</code>	Turns true if two operands aren't identical	<code>x is not y</code>

- Membership operators: `in` and `not in` are used to check whether a value or operand is present in a sequence or not

OPERATOR	FUNCTION	EXAMPLE
<code>in</code>	Turns true if operand is found in the sequence	<code>x in y</code>
<code>not in</code>	Turns true if operand is not found in the sequence	<code>x not in y</code>

# 6

## If Else and Elif Statements

Sometimes we need to set some conditions for our code to run a certain block of code upon meeting the condition. These are called using conditional statements in programming. Python also supports logical conditional statements

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

### *If Statement*

We test a simple condition by using the 'if' keyword and then present a certain condition.

```
x = 400
y = 200
if x > y:
    print("x is greater than y")
```

OUTPUT:

```
x is greater than y
```

### *Else Statement*

The else statement is used when there are two conditions to be checked within a code block. For example:

```
x = 100
y = 200
if x > y:
    print("x is greater than y")
else:
    print("y is greater than x")
```

OUTPUT:

y is greater than x

## ***Elif Statement***

Elif conditions are used when there are more than two conditions involved, for example if two values doesn't meet the condition then the third condition might work.

For example:

```
x = 300
y = 200
if x > y:
    print("x is greater than y")
elif x == y:
    print("both values are equal")
else:
    print("y is greater than x")
```

OUTPUT:

x is greater than y

# 7

## Loops

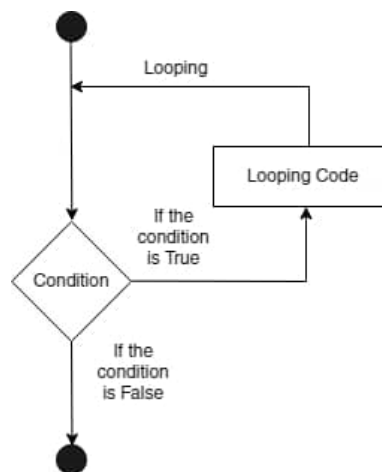
We need looping or 'loops' in python especially when we want a certain block of code to run unless the condition is met. Let's say that we want python to print natural numbers from 1 to 10 and then stop. So, instead of writing the print statement for every number, we will create a loop. Now, there are two types of loops in python:

- While Loop
- For Loop
- Nested Loops

### ***While Loop***

A while loop in python continues to execute the code as long as the given condition is true, if it becomes false, then the loop stops.

Another reason to use while loop is when we aren't sure about how many times to iterate.



## Syntax of While Loop

```
while expression:
    statements
```

In the above syntax, a while loop starts by checking the **expression**, if the **expression** is **true**, the loop continues to **statements**. After making an iteration, it goes back to expression and continues to execute through the **statements** until the expression becomes **False**.

For example:

```
n = 1
while n < 10:
    print(n)
    n += 1
```

OUTPUT:

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

In the above example, the variable `n` is already set to 1, and a while loop is set to a condition where `n` is less than 10, then on line 3 the print statement prints `n` where currently 1 is stored, on line 4 the `n` is incremented by 1 and becomes 2 and `n` now holds 2. The loop continues to execute and increment until `n` becomes 10; when `n` becomes 10 the expression becomes false, hence the loop stops.

Note: The while loop will run infinite times if 'n' doesn't get an increment, hence it will make your program unstoppable and may cause errors.

### While Loops with Else:

Python allows us to use else statement with the while loop, let's suppose that if the while expression turns out to be False then we may use an 'else' statement to indicate the user that the program will no longer run further.

For Example:

```
n = 1
while n < 10:
    print(n)
    n += 1
else:
    print("The program has ended since n is greater than 10")
```

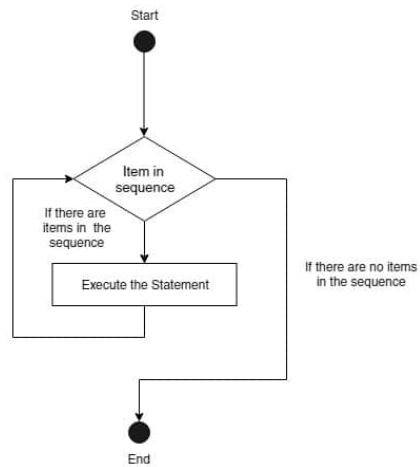
OUTPUT :

1  
2  
3  
4  
5  
6  
7  
8  
9

The program has ended since n is greater than 10

## ***For Loop***

For loop is a python loop which repeats a group of statements for a given number of times. It is always a good practice to use the for loop when we know the number of iterations.



## ***Syntax of For Loop***

```

for variable in sequence:
    statement 1
    statement 2
    ...
  
```

Here is an example of a list containing fruits, let us use **for loop** to go through each known number of fruits:

```
For Example:
fruits = ['apple','grapes', 'oranges', 'bananas']
for fruit in fruits:
    print(fruit)
```

OUTPUT:

```
apple
grapes
oranges
bananas
```

### ***Using the range() function in for Loop***

We can use the range() function using for loops and in that range we can define the number of times we want to iterate; for example if we want to print natural numbers up to 10 then we have to define a range of 11, since the range() function starts at index 0.

```
For Example:
#print natural numbers upto 10
for n in range(11):
    print(n)
```

OUTPUT:

```
0
1
2
3
4
5
6
7
8
9
10
```

You can use the for loop to print each letter of a string as well.

```
For Example:
#print python tricks each character
for character in 'python tricks':
    print(character)
```

OUTPUT:

p  
y  
t  
h  
o  
n  
  
T  
r  
i  
c  
k  
s

### ***Using the if-else statements with For Loop:***

Python language supports the usage of if-else statements with for loop. If-else statements comes handy if you want to execute the loop with an applied condition, as long as the condition is met the loop continues to execute; else the loop gets terminated.

For Example:

```
#Checking which toppings are available using if-else with for loop
icecream_toppings = ['chocolate', 'strawberry', 'blueberry', 'cotton candy', 'cherry']
available_toppings = ['chocolate','blueberry']

for toppings in icecream_toppings:
    if toppings in available_toppings:
        print("Available toppings are: " + toppings)
    else:
        print("Sorry we don't have " + toppings)
print("finished making your icecream")
```

### ***Explanation***

At first, we define a list of total **icecream\_toppings** in this example. Next we made a list of **available\_toppings** that are currently present. Now if we find anything that is not available currently like 'strawberry', 'cotton candy' and 'cherry' then we loop through the list of **available\_toppings**. Inside the loop we check to see if the **available\_toppings** is present in the list of **icecream\_toppings**, if it is then we add the available toppings on the ice cream. If the **available\_toppings** is not in the list of **icecream\_toppings**, the else block will run. The else block will tell which toppings are not currently in stock.



## ***Nested Loops***

Python supports using the nested loop that is a loop within a loop. The following example is an explanation of how a nested loop works.

### ***Syntax of Nested Loops:***

A nested loop of for loop looks like this:

```
for variable in sequence:
    for variable in sequence:
        statement 1
        statement 2
```

A nested loop for while loop looks like this:

```
while expression:
    while expression:
        statement 1
        statement 2
```

Let's look at an example of a nested loop containing for loops inside it.

```
For Example:
#Printing tables of 1 to 10 using nested loops
for i in range(1,11):
    for j in range(1,11):
        print(i*j)
    print("\n")
```

See output [here](#):

# 8

## Control Statements

Let's learn about the control statements in python now, control statements change the way we execute a loop from its normal behavior. There are many types of control statements in python that you can use to control the loops:

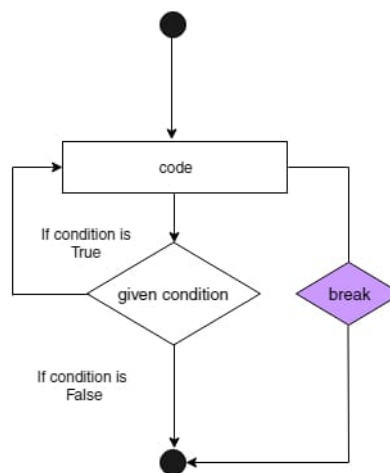
### ***Break Statement***

Break statement is used to terminate or abandon the loop. Once the loop breaks then the next statement after the **break** continues. The main reason we could use the break statement is to take a quick exit from a nested loop (i.e. a loop within a loop). The **break** statement is used with both **while** and **for** loop.

### ***Syntax***

```
break
```

This is how a break statement works within a loop:



Let's look at how to use a **break** statement to exit a **while** loop:

For Example:

```
country_name = "\nPlease enter the name of the country you love: "
while True:
    country = input(country_name)
    if country == 'quit':
        break
    else:
        print("My favorite country is: " + country.title())
```

OUTPUT:

See output [here](#)

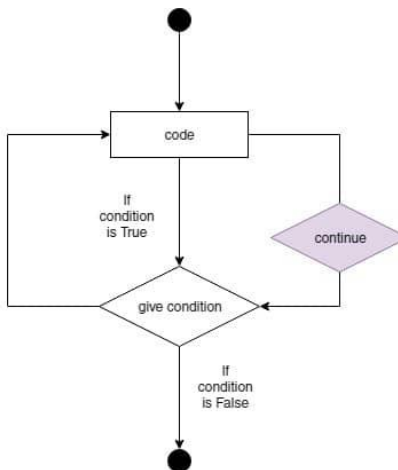
## Continue Statement

Continue statement in Python is used to continue running the program even after the program encounters a break during execution. The continue statement enables the code to proceed inside a loop and move on skipping the particular condition.

## Syntax

```
continue
```

This is how a **continue** statement works within a loop:



Let's look at how to use a **continue** statement inside a loop:

```

For Example:
for letter in 'Py thon':
    if letter == ' ':
        continue
    print ('Letters:', letter)

```

```

OUTPUT:
Letters: P
Letters: y
Letters: t
Letters: h
Letters: o
Letters: n

```

The execution will skip the 'space' character and continue to print the rest of the loop without any interruptions.

## ***Pass Statement***

The pass statement is used when comment cannot be printed to show the program's status to the programmer. It is always going to be a null operation. For example, if a programmer has written a comment within the code, then that comment will be ignored but the pass statement won't be ignored. It will be used as a placeholder.

## ***Syntax***

```
pass
```

Let's look at how to use a **pass statement** in Python:

```

For Example:
for letter in 'Python':
    if letter == 'h':
        pass
    print("We are still printing letters")
print ('Letters:', letter)

```

```
OUTPUT:  
Letters: P  
Letters: y  
Letters: t  
We are still printing letters  
Letters: h  
Letters: o  
Letters: n
```

# 9

## Number Types

There are three main types of numbers in Python:

- Integer
- Decimal
- Complex

These different numbers can be assigned to the variables and to verify the type of any object in Python, use the `type()` function:

For Example:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

```
print(type(x))
print(type(y))
print(type(z))
```

OUTPUT:

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

The difference between integers and float numbers is that a float number is separated by a decimal. So '10' is an integer and '10.5' is a float number. Python gives us the freedom to store the integer, floating and complex numbers and also lets us do conversion operations between them.



## ***Exponential numbers***

You can write an exponential number using the letter 'e' between the mantissa and the exponent.

For Example:  

```
print(2e5)
```

OUTPUT:  
 200000.0

Remember that this is power of 10. To raise a number to another's power, we use the \*\* operator. If you face difficulty in Python number types, please comment.

## ***Float in Python***

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

For Example:  

```
from math import pi
print(pi)
```

OUTPUT:  
 3.141592653589793

A float value is only accurate upto 15 decimal places. After that, it rounds the number off.

## ***Complex Numbers in Python***

Complex numbers are written with a "j" as the imaginary part:

For Example:  

```
a=2+3j
print(a)
```

OUTPUT:  
 (2+3j)

## ***Coefficient to the imaginary part***

Here, 2 is the real part, and 3j is the imaginary part. To denote the irrational part, however, you can't use the letter 'i', like you would do on paper.

For Example:  

```
a=2+3i
print(a)
```



OUTPUT:  
SyntaxError: invalid syntax

Also, it is mandatory to provide a coefficient to the imaginary part.

### ***Operations on complex numbers***

Finally, you can perform the basic operations on complex numbers too.

For Example:  
a=2+3j  
b=2+5j  
print(a+b)

OUTPUT:  
(4+8j)

In this tutorial, we learnt about Python number types. We looked at int, float, and complex numbers in detail. You can try out your own combination and test accordingly.

# 10

## Strings

In this tutorial we will learn about strings in python. We will learn about how to create, edit and delete strings, also we will learn about the associate operations and functions of strings.

### ***What are Strings***

Strings are one of most popular aspects of python. We can simply create a string by adding single or double quotes around characters. Now, to create your first string, all you need to do is:

```
string1 = 'Hello, I am single quotes'  
string2 = "Hello, I am double quotes"
```

**Note:** Strings are not limited to be enclosed in single or double quotes, rather you can use triple quotes as well to create a multiline string:

For Example:

```
#single quotee  
string1 = 'Hello, I am single quotes'  
print(string1)  
  
#double quotes  
string2 = "Hello, I am double quotes"  
print(string2)  
  
#triple quotes for multilines  
string3 = """Hey, I am triple quotes, and  
you are?"""  
print(string3)
```

OUTPUT:

```
Hello, I am single quotes  
Hello, I am double quotes  
Hey, I am triple quotes, and  
you are?
```

## How Strings are Indexed?

Like any other character in python, characters used in strings are also indexed number wise, where each character takes up a 1 byte of memory and has an address in terms of indexing. For example, if I have a string consisting of the word 'python' then each character of 'python' has a separate index and memory. Let's say that I want to access p from the string 'python', then I would simply write:

```
For Example:
string = 'python'
print(string[0])
```

Since, indexing always starts from 0, so the first letter will be 'p' in this case which is stored at 0 index. The output of the above code will be:

```
OUTPUT:
p
```

## Slicing a String

You can also slice a string by choosing the index number of characters that you want to slice through the slicing method. Let's slice p from python by omitting index '0' from the rest of the index numbers of characters:

```
For Example:
#slicing p from python
string = 'python'
print(string[1:6])
```

```
OUTPUT:
ython
```

## Positive and Negative Indexing of Strings

We have learned that strings can be indexed positively through numbers starting from 0 onward. However, strings can be indexed negatively as well in the reverse order. For example, to view the last character of the word 'python' you can enter the index '-1' to view letter n from 'python'.

0	1	2	3	4	5
P	Y	T	H	O	N
-6	-5	-4	-3	-2	-1

## Updating a String Through Slicing

You can update a current string with some other characters using the slicing method:

For Example:

```
#updating a string through slicing
string = 'Python Tricks'
print("New String: " + string[0:6] + " Love")
```

OUTPUT:

```
New String: Python Love
```

## Deleting a String

You can delete a string too but remember that deleting a character from a string is not possible however you can delete an entire string using `del` function:

For Example:

```
#updating a string through slicing
string = 'python'
del string
print(string)
```

```
#The output will be null as we have used the del statement to delete the string
```

OUTPUT:

```
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
    print(string)
NameError: name 'string' is not defined
```

## String Operations in Python

There are many string operations that you can do with different data types and variables in Python as we have studied previously as well.

### The Concatenate Operation

When you join or 'add' to strings together to form one new string, that process is called concatenation.

The plus(+) symbol is used to concatenate or join two strings together:

For Example:

```
string1 = 'I love'
string2 = ' Python Programming'
adding = string1 + string2
print(adding)
```

OUTPUT:

```
I love Python Programming
```

You can also concatenate a string by using the \* sign i.e. if you want to print a same string multiple times then you need to multiply the string with that particular number.

For Example:

```
#Adding & multiplying two strings together
string1 = 'I love'
string2 = ' Python Programming '
adding = string1 + string2
multiplying = string1 * 3
print(adding)
print(multiplying)
```

OUTPUT:

```
I love Python Programming
I loveI loveI love
```

### ***Using Iteration in Strings***

You can use a for loop to through the characters in your string quite easily. For example, if you want to count the number of characters present in your string then you can simply:

For Example:

```
#Counting the number of characters of a string using a for loop
string = 'I love Python'
count = 0
for letter in string:
    count = count + 1
print(count)
```

OUTPUT:

```
13
```

### ***Testing Sub Strings within a String***

You can know easily whether a sub string exists (or doesn't exist) within a string by using the `in` and `not in` methods. If a sub string exist within a string then the result will be in boolean form resulting `True`. Similarly, if a sub string doesn't exist within a string then the boolean value will be `False`.

For Example:

```
#Checking whether a substring is present or not inside a string
if 'p' in 'python':
    print(True)
elif 'p' not in 'python':
    print(False)
```

OUTPUT:

```
True
```

## ***Built-in Functions in Strings***

Python consists of tons of built-in functions for strings. These functions/method enable strings to produce different result for a particular query. For example, if you want to play around with the text cases of your string, you can create lower(), upper(), title() and capitalize() functions to your strings.

For Example:

```
string = "I love python tricks"

#Creating all letters of your string to lowercase
print("Lower case letters:")
print(string.lower())

#Creating all letters of your string to uppercase
print("Upper case letters:")
print(string.upper())

#Creating first letter of each word capital
print("Title case letters:")
print(string.title())

#Creates first letter of your string capital
print("Capital case letters:")
print(string.capitalize())
```

OUTPUT:

```
Lower case letters:
I love python tricks
Upper case letters:
I LOVE PYTHON TRICKS
Title case letters:
I Love Python Tricks
Capital case letters:
I love python tricks
```

## ***Formatting Strings in Python***

While programming, in python you may encounter syntax errors as well caused by the arrangements of characters within a string. To avoid that, we require some string formatting to encounter the relevant issues to solve errors. For example, if you use single quotes and double quotes together in a string, then that might cause an error, unless you tell python to do otherwise. For example, if you use double quotes twice along with a single quote (apostrophe) then the output will be a syntax error:

For Example:

```
string = "He asked me, "What's your name?"
print(string)
```

OUTPUT:

```
string = "He asked me, "What's your name?"
SyntaxError: invalid syntax
```

Now, there are multiple ways to tackle the issue of this syntax error caused by using double quotes or single quotes together. You can use triple quotes around single and quotes to remove this issue or you can use escape sequence (\) to remove the issue.

For Example:

```
#Escape single quotes
print('He asked me, "What\'s your name?")

#Escaping double quotes
print("He asked me, \"What's your name?\")

#Using triple quotes
print('\'\'He asked me, "What's your name?\'\'')
```

OUTPUT:

```
He asked me, "What's your name?"
He asked me, "What's your name?"
He asked me, "What's your name?"
```

### ***How to Ignore Escape Sequence?***

So, previously we have learned about how to use escape sequence to format our strings. Now we will learn about how to use escape sequence as a character within a string by using **r** or **R** to ignore escape sequence.

For Example:

```
#Using escape sequence
print("This is another line: \nI am a new line")

#Ignoring escape sequence
print(r"This is another line: \nI am a new line")
```

OUTPUT:

```
This is another line:
I am a new line
This is another line: \nI am a new line
```

# 11

## Lists

Python list is used to store the sequence of different data types. Python, however, includes six kinds of data types that can store the sequences, but list is the most prevalent and reliable form.

A list may be defined as a collection of different types of values or items. The items in the list are divided by the comma (,) and items are encapsulated inside the square brackets [].

You can define lists as follows:

```
For Example:
names = ["Hira", 302, "Paris"]
numbers = [1, 2, 3, 4, 5, 6]
random = [3, "John"]
```

```
print(names)
print(numbers)
print(random)
```

```
OUTPUT:
['Hira', 302, 'Paris']
[1, 2, 3, 4, 5, 6]
[3, 'John']
```

### *Indexing and Splitting in Lists*

Indexing is processed in the same manner as the strings are processed. Using the slice operator [], the list components can be accessed. The index begins at 0 and ends at -1. The list's first component is placed in the 0th index, the list's second component is placed in the 1st index, and so on.

List = [1,2,3,4,5,6]

1	2	3	4	5	6
---	---	---	---	---	---

List[0] = 1

List[0:] = 1,2,3,4,5,6

List[1] = 2

List[0:3] = 1,2,3

List[2] = 3

List[1:4] = 2,3,4



Python gives us the flexibility to use negative indexing as well, unlike other languages. From the right, negative indexes are counted. The list's last component (most right) has the index-1, its next element is placed in the index-2 (from right to left) and so on until most of the left component is found.

0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

## Updating an Item in List

Lists are mutable, meaning that unlike string or tuple, their elements can be changed. To alter an item or a variety of items, we can use assignment operator (=).

```
For Example:
numbers = [0,1,2,3,4,5]
print(numbers)
numbers[2] = 4;
print(numbers)
numbers[1:3] = [89, 78]
print(numbers)
```

```
OUTPUT:
[0, 1, 2, 3, 4, 5]
[0, 1, 4, 3, 4, 5]
[0, 89, 78, 3, 4, 5]
```

You can also delete the queue components by using the **del** keyword.

```
For Example:
string = ['p','y','t','h','o','n']
del string[2]
print(string)
```

```
OUTPUT:
['p', 'y', 'h', 'o', 'n']
```

Also, Python provides us with the **remove()** method or a **pop()** method if we don't know which component to remove from the list. The **pop()** method removes the last element of the list if the index is not provided.

```
For Example:
string = ['p','y','t','h','o','n']
string.remove('p')
string.pop()
print(string)
```

```
OUTPUT:
['y', 't', 'h', 'o']
```

## ***Adding elements to the list***

You can add new elements to an existing list by using the **append()** method. The only condition with **append()** method is that you can only append or add new elements to the end of the list.

For Example:

```
fruits = ['apple', 'banana', 'orange', 'kiwi']
fruits.append('grapes')
print(fruits)
```

OUTPUT:

```
['apple', 'banana', 'orange', 'kiwi', 'grapes']
```

## ***Iterating a List***

You can iterate through a list as well by using a *for loop*

For Example:

```
numbers = [1,2,3,4,5,6,7,8]
for n in numbers:
    print(n)
```

OUTPUT:

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

# 12

## Tuples

Using Python Tuple, the sequence of immutable python objects is stored. Tuple is similar to lists since it is possible to change the value of the items stored in the list while the tuple is immutable and it is not possible to change the value of the items stored in the tuple.

A tuple comprises of set of comma-separated objects encapsulated inside the parentheses. The following can be described as a tuple:

```
For Example:
tuple1 = (230, "James", 123)
tuple2 = ("red", "green", "blue")
```

### *Using a For Loop in Tuples*

You can also iterate a tuple using a for loop:

```
For Example:
tuple= ("apple", "orange", "lemon")
for n in tuple:
    print(n)
```

```
OUTPUT:
apple
orange
lemon
```

### *Adding Items to Tuple*

Since Tuples are immutable meaning that you cannot add the values to a tuple once it is created just like it is possible in lists.

```
For Example:
tuple1= ("apple", "orange", "lemon")
tuple1[3] = "banana" # It's an error
print(tuple1)
```

banana won't be created as a new element since tuple doesn't support it. Hence, the output will be:

```
OUTPUT:
Traceback (most recent call last): File "/tmp/sessions/4ee7c8963287ca22/main.py", line
2, in <module> tuple1[3] = "banana" TypeError: 'tuple' object does not support item
assignment
```

## ***Negative Indexing in Tuple***

Just like lists, tuples also have negative indexing starting from the most right (-1) to the most left.

```
For Example:  
tuple1= ("apple", "orange", "lemon")  
print(tuple1[-1])
```

```
OUTPUT:  
lemon
```

## ***Changing Tuple Values***

As we know that tuples are immutable, means that we cannot change the values inside a tuple.

However, we can convert a tuple into a list and then change the values of that list and again change the particular list into a tuple again.

```
For Example:  
tuple1= ("apple", "orange", "lemon")  
tuple2 = list(tuple1)  
tuple2[1] = "banana"
```

```
tuple1 = tuple(tuple2)  
print(tuple1)
```

We were able to convert the tuple into a list by using the **list()** constructor and later we converted the list into a tuple again by using the **tuple()** constructor.

```
OUTPUT:  
( 'apple', 'banana', 'lemon' )
```

# 13

## Dictionaries

A dictionary is an unordered, mutable, and indexed collection. Dictionaries are created by using curly brackets in Python, and they consist of key value pairs. Let's create a simple dictionary to understand more:

```
book= {  
    "pages": "277",  
    "name": "Gone Girl",  
    "year": 2007  
}  
print(book)
```

In the above example we have seen that we have created a dictionary known as **book**, which consists of three key-value pairs where **pages**, **name** and **year** are the keys and **277**, **Gone Girl** and **2007** are their respective values.

OUTPUT:  
{'pages': '277', 'name': 'Gone Girl', 'year': 2007}

### *Accessing Elements inside a Dictionary*

We can always access the elements inside the dictionary by using the 'key' name to access the 'value' of a dictionary.

For Example:  
book= {  
 "pages": "277",  
 "name": "Gone Girl",  
 "year": 2007  
}  
x = book['name']  
print(x)

OUTPUT:  
Gone Girl

## Changing Elements inside a Dictionary

You can change the values of a particular by referring the item to its key

For Example:

```
book= {
    "pages": "277",
    "name": "Gone Girl",
    "year": 2007
}
book['year'] = '2010'

print(book)
```

## Using Loop to print keys and values

You can print all the keys by using a simple for loop, for example:

```
book= {
    "pages": "277",
    "name": "Gone Girl",
    "year": 2007
}
for n in book:
    print(n)
```

You can also print values separately which is not possible in using a regular for loop (which only prints keys), so in order to print values you can use the values() method at the end of the dictionary's name,

For Example:

```
book= {
    "pages": "277",
    "name": "Gone Girl",
    "year": 2007
}
for n in book.values():
    print(n)
```

OUTPUT:

```
Gone Girl
277
2007
```

If you want to print both keys and values together as a key-value pair using a for loop, you can do that as well by using two variables n and m that will get stored in items() methods,

```
for example:
book= {
    "pages": "277",
    "name": "Gone Girl",
    "year": 2007
}
for n, m in book.items():
    print(n,m)
```

```
OUTPUT:
pages 277
name Gone Girl
year 2007
```

## ***Removing Items from a Dictionary***

You can pop items in a dictionary as well by using the pop() function. For example if you want to pop out name of the book, then you can do that by naming 'name' inside the pop() parentheses.

```
book= {
    "pages": "277",
    "name": "Gone Girl",
    "year": 2007
}
book.pop('name')
print(book)
```

```
OUTPUT:
{'pages': '277', 'year': 2007}
```

You can use the del function as well to delete an item inside a dictionary or the whole dictionary as well:

For Example:

```
book= {
    "pages": "277",
    "name": "Gone Girl",
    "year": 2007
}
del book['pages']
print(book)
```

```
OUTPUT:
{'name': 'Gone Girl', 'year': 2007}
```

Similarly, you can delete the whole dictionary as well by using the del keyword:

For Example:

```
book= {
    "pages": "277",
    "name": "Gone Girl",
    "year": 2007
}
del book
print(book)
```

```
OUTPUT:
Traceback (most recent call last):
  File "/tmp/sessions/11ab166edc475687/main.py", line 7, in
    print(book)
NameError: name 'book' is not defined
```

# 14

## Functions

This tutorial will guide you through the Python function. It enables you know how to build user-defined functions and use them to compose Python module programs.

Python enables us to split a big program into the chunks of function-friendly buildin. The function includes the collection of { } curly brackets. A function can be called several times to provide the python program with reusability and modularity.

### *How to define a Function?*

In order to define a function all you need to do is follow a few guidelines:

- First, to define a python function you will use the keyword 'def' before the describing the name of the function. 'def' stands for 'define'. Once you have added **def** followed by the function name (which should follow the same rules that we used while defining the variables in python), and after defining the function name add parentheses as well with colon in the end. Syntax of writing a function: `def function_name():`
- Next, you will add the function body, this part decides what your function is going to do. Add valid python code here.
- You can add a return statement if you want. However, that is completely optional.

A normal function would look like this:

```
def function_name(parameter):  
    function_body  
    return
```



## How to Call a Function?

After defining, naming and specifying the parameters and arranging the structure of code blocks then it's time to call the function. You can easily call the function directly inside the python program.

```
For Example:
def name_printing():
    print("My name is Python Tricks")

name_printing()
```

OUTPUT:  
My name is Python Tricks

## Adding Parameters in Function Calling

We can add parameters inside the parentheses of the function, we can add number of parameters to be separated by comma. If you change the value of the parameter, then this will affect during the function calling as well.

```
For Example:
def change_name(name, age):
    print ("My name is " + name + " and my age is " + str(age))

change_name("Hira", 25)
change_name("Smith", 30)
```

OUTPUT:  
My name is Hira and my age is 25  
My name is Smith and my age is 30

## Setting a Default Parameter

A default argument (parameter) stands for a value that is given inside the function definition. For example, you don't want to define a during function calling then you may want to set a default argument in the function definition, like:

```
For Example:
def change_name(name, age=25):
    print ("My name is " + name + " and my age is " + str(age))

change_name("Hira")
```

OUTPUT:  
My name is Hira and my age is 25

### ***\*args or Multiple Arguments***

Sometimes while writing a function, you may notice that there would be more arguments required than what has been decided earlier in the function definition. To tackle this issue you just need to enter the name of a parameter with an **asterisk (\*)** in the beginning.

For Example:

```
def my_colors( *colors ):
    print ("Output is: ")

    for color in colors:
        print (color)

my_colors('red','green','yellow')
```

OUTPUT:

```
red
green
yellow
```

### ***\*\*kwargs or Multiple Keyword Arguments***

The **\*\*kwargs** in python function definition is used to pass keyword, variable-length argument list. You can think of this as a dictionary with key-value pairs. It is represented with double **asterisk (\*\*)**.

For Example:

```
def person_info(**my_info):
    print ("Output is: ")

    for key, value in my_info.items():
        print (key, value)

person_info(name="hira", age=25)
```

OUTPUT:

```
name hira
age 25
```

You can add as many key-value pairs as you want.

# 15

## Modules

### *What is a Module?*

A module is a python file with a '.py' extension which consists of functions, statements and variables. The reason we use modules is to break down big programs into simpler and smaller ones. We can reuse module files anytime as well.

Let's create a module file and name it as '**my\_code.py**', and create a function inside it.

```
def my_function():  
    print("This is a module file")  
  
my_function()
```

### *Importing a Module*

Now that we created a module file, let's save it in the same directory where our **main** python file is stored. Now let's suppose we are inside our **main.py** file; now it's time to use the **import** statement to import the **my\_code.py** file:

```
import my_code  
  
my_code.my_function()
```

When the python interpreter sees an import statement, it imports the module file if the module is present in the same directory, otherwise it will show an error. Once the module is imported then you can call the relevant function name that you want to use.

```
my_code.my_function()
```

**Note:** Whenever you want to use the function from your module file, then you have to follow the syntax: `modulename.functionname()`.

## 'From' in Import Statement

Sometimes the module file is gonna have more than one block of code so Python lets you import a specific block from your module, let's say that you have a file named **info\_modules.py** and it has two dictionaries named 'person\_1' and 'person\_2' and you want to import 'person\_1' in your main program.

```
person_1 = {
    'name': "hira",
    'age': '29',
    'gender': 'Female',
}

person_2 = {
    'name': "Smith",
    'age': '29',
    'gender': 'Male',
}
```

Importing person\_1 in the main.py file using **from** statement

```
from my_code import person_1
print(person_1)
```

Output will be:

```
{'name': 'hira', 'age': '29', 'gender': 'Female'}
```

**Note:** once you have specifically imported the function, variable or statement in your main program then you don't have use `modulename.functionname()` syntax while calling it in your main file. Just call the name directly with from statement.

## Renaming a Module using Alias

You can rename a module as well in python. Let's say you want to call your info\_modules as info as a short form so that you can have flexibility, for example:

```
import info_modules as info
print(info.person_1)
```

OUTPUT:

```
{'name': 'hira', 'age': '29', 'gender': 'Female'}
```

## Built in Modules

There is an extensive list of all the built-in modules that python supports, each caters a different functionality and usage depending on what we want to calculate. In order to see the list of modules python supports just type:

```
print(help('modules'))
```

OUTPUT:

```
IPython
__future__
_abc
_ast
_asyncio
_bisect
_blake2
_bootlocale
_bz2
_codecs
_codecs_cn
_codecs_hk
_codecs_iso2022
_codecs_jp
_codecs_kr
_codecs_tw
_collections
_collections_abc
_compat_pickle
_compression
_contextvars
_csv
_ctypes
_ctypes_test
_datetime
_decimal
_distutils_findvs
_dummy_thread
_elementtree
_functools
_hashlib
_heapq
_imp
_io
_json
_locale
_lsprof
_lzma
_markupbase
_md5
_msi
_multibytecodec
_multiprocessing
_opcode
_operator
_osx_support
_overlapped
_pickle
_weakrefset
_winapi
abc
aifc
antigravity
argparse
array
ast
asynchat
asyncio
asyncore
atexit
audioop
autoreload
backcall
base64
bdb
binascii
binhex
bisect
builtins
bz2
cProfile
calendar
cgi
cgitb
chunk
cmath
cmd
code
codecs
codeop
collections
colorama
colorsys
compileall
concurrent
configparser
contextlib
contextvars
copy
copyreg
crypt
csv
ctypes
curses
cythonmagic
dataclasses
heapq
hmac
html
http
idlelib
imaplib
imghdr
imp
importlib
ind
inspect
io
ipaddress
ipython_genutils
itertools
jedi
json
keyword
lib2to3
linecache
locale
logging
lzma
macpath
mailbox
mailcap
marshal
math
mimetypes
mmap
modulefinder
msilib
msvcrt
multiprocessing
netrc
ntplib
nt
ntpath
nturl2path
numbers
opcode
operator
optparse
os
parser
parso
pathlib
pdb
secrets
select
selectors
setuptools
shelve
shlex
shutil
signal
simplegeneric
site
six
smtpd
smtplib
sndhdr
socket
socketserver
sqlite3
sre_compile
sre_constants
sre_parse
ssl
stat
statistics
storemagic
string
stringprep
struct
subprocess
sunau
symbol
sympyprinting
symtable
sys
sysconfig
tabnanny
tarfile
telnetlib
tempfile
test
tests
textwrap
this
threading
time
timeit
tkinter
token
tokenize
```

_py_abc	datetime	pickle	trace
_pydecimal	dbm	pickleshare	traceback
_pyio	decimal	pickletools	tracemalloc
_queue	decorator	pip	traitlets
_random	difflib	pipes	tty
_sha1	dis	pkg_resources	turtle
_sha256	distutils	pkgutil	turtledemo
_sha3	doctest	platform	types
_sha512	dummy_threading	plistlib	typing
_signal	easy_install	poplib	unicodedata
_sitebuiltins	email	posixpath	unittest
_socket	encodings	pprint	urllib
_sqlite3	ensurepip	profile	uu
_sre	enum	prompt_toolkit	uuid
_ssl	errno	pstats	venv
_stat	faulthandler	pty	warnings
_string	filecmp	py_compile	wave
_strptime	fileinput	pyclbr	wcwidth
_struct	fnmatch	pydoc	weakref
_symtable	formatter	pydoc_data	webbrowser
_testbuffer	fractions	pyexpat	winreg
_testcapi	ftplib	pygments	winsound
_testconsole	functools	queue	wsgiref
_testimportmultiple	gc	quopri	xdrlib
_testmultiphase	genericpath	random	xml
_thread	getopt	re	xmlrpc
_threading_local	getpass	replib	xxsubtype
_tkinter	gettext	rlcompleter	zipapp
_tracemalloc	glob	rmagic	zipfile
_warnings	gzip	runpy	zipimport
_weakref	hashlib	sched	zlib

# 16

## File Handling

### *File in Python*

We all know about what is a file in conventional terms but let's talk about the file in python. It is just like a normal file that is stored in the conventional way inside the hard disk because things stored in RAM are temporary and are likely to be missing once the computer gets turn off.

Let's say we want to read from or write a file in python so we will have to open the file first. And after performing particular task we need to close the file as well. So, we have to follow all the steps in order to handle files in python. Python file operations is divided into four parts:

- Opening the file
- Reading or writing the file
- Append the file
- Delete the file

### *Opening the File*

In order to open a file in Python, we will use the `open()` built-in function. This function will return python the file object. The `open()` function will have two parameters: filename and mode

The `open()` function takes two parameters; **filename**, and **mode** separated by the commas (filename, mode)

- Filename is going to be the name of the file that you want to open.
- Mode is going to decide that in which specification we want to open the file. There are four different types of modes that you can use:
  - "r" - Read Mode - It is the default value when there is no mode defined. 'r' is used to open the file in read mode
  - "a" - Append Mode - It opens to file for appending, and if there is none then it creates one.
  - "w" - Write Mode - It opens the file for writing, This mode can overwrite or create new content in the file.
  - "x" - Create Mode - It is only validated as long as there is no existing file. If the file already exists in the directory, then it shows an error.
  - "t" - Text Mode - Default text view mode. The file in python always opens in the text mode unless you change it.
  - "b" - BinaryMode - If your file is an image then you may want to utilize this mode to open image files.

To open the file, you need to use the following syntax in a python file, let's suppose that file is **main.py**, here you will enter the following:

```
f = open("new.txt")
```

**Note:** The above statement opens the file named 'new' in the default read ('r') and text mode ('t'). If the file doesn't exist in the directory, then you will get an error.

## Reading or Writing the File

Now let's assume that we have a file named 'new.txt' in the same folder where our **main.py** file is, once we have **opened** the file and returned it then we will start reading the contents of the file in python so for that we will use the **read()** method:

```
f = open("new.txt")
print(f.read())
```

Once the **read()** method is printed then all the contents present in the 'new.txt' file will get printed on the screen, so the output will be:

```
Hello, I love Python Programming
I am learning Python
Wish me best of luck
```

## Reading Parts of File

You can read parts of the characters present in your 'new.txt' file too if you want, let's say that you want to read the first 10 characters present in your file then you can use the method **read()**, here you can see that you have specified 10 as a number to be read only. For Example:

```
f = open("new.txt")
print(f.read(10))
```

Output will be:

```
Hello, I l
```

## Reading Lines of File

You can start reading the lines too, let's say that you have more than one lines in your file, then you can read one line at a time. For Example:

```
f = open("new.txt")
print(f.readline())
```

Output:

```
Hello, I love Python Programming
```

If you run the **readline()** function twice then python will execute the second line and on.



```
f = open("new.txt")
print(f.readline())
print(f.readline())
```

**OUTPUT:**

Hello, I love Python Programming

I am learning Python

***Closing the File***

Another important thing to remember is to use the **close()** method to close the files. Sometimes the file that we read on python still goes on buffering in the background, so we need to make sure that we close the file in order to save disk space and processing errors.

```
f = open("new.txt")
print(f.readline())
f.close()
```

***Writing to a File***

Once we learned about reading the file, now it is time use the **write()** method in 'w' mode. This will erase all previous content and overwrite new one.

**For Example:**

```
f = open("new.txt", 'w')
f.write("Where did the text go?")

f.close()
```

```
f = open('new.txt')
print(f.read())
```

**OUTPUT:**

Where did the text go?

So, the new text has replaced the old one, and all previous data is erased.

***Creating a New File***

Let's assume that you don't have a text file in your directory to open but you still can create a new file in python by sing 'x' parameter, store data in it and read from it later on.

**For Example:**

```
f = open("new_word.txt", 'x')
```

Run the code once, in order to write something to it, remove the previous code and enter the following:

```
f = open("new_word.txt", "w")
f.write("I have created a new file named new_word.txt")

f.close()

f = open("new_word.txt")
print(f.read())
```

**OUTPUT:**

I have created a new file named new\_word.txt

***Appending to A File***

You can append text at the end of the existing content as well by using the 'a' as your parameter.

**For Example:**

```
f = open("new_word.txt", "a")
f.write("\nI have appended some text in new_word.txt")

f.close()

f = open("new_word.txt")
print(f.read())
```

**OUTPUT:**

I have created a new file named new\_word.txt  
I have appended some text in new\_word.txt

***Deleting a File***

You can delete a file in python as well by using the **import os** statement

```
import os
os.remove("new.txt")
```

If you will check your directory, you will see that the 'new' file isn't there anymore.

# 17

## Exception Handling

### ***What is Error?***

Error occurring is an unfortunate event in coding that happens when something in the code is not right. It could be a human mistake in coding structure or calculations.

For Example:

```
a = 4
b = 2
if a < b
    print("a is less than b")
else:
    print("b is less than a")
```

OUTPUT:

```
File "C:\Users\Hira\Desktop\Python Projects\Test\main.py", line 61
  if a < b
    ^
SyntaxError: invalid syntax
```

### ***What is Exception Handling?***

Before we jump into knowing about Exception Handling in Python, we must learn about why we need to handle exceptions in the first place. When you are coding in python or any other programming language, you are bound to have errors and some of those errors might terminate your whole program. But, what if, you as a developer is able to tackle those errors with exception handling methods? So yes! there are exception handling methods in python for developers to deal with recurring errors.

## ZeroDivision Exception

Let's look at a quick example of the zero-exception error that happens when a number is divided by 0.

It's impossible to do so, see below:

```
print(10/0)
```

OUTPUT:

```
Traceback (most recent call last):
  File "C:\Users\Hira\Desktop\Python Projects\Test\main.py", line 1, in
    print(10/0)
ZeroDivisionError: division by zero
```

Above, you can see that there is a ZeroDivisionError: division by zero, python stops the program right there unless we put an exception to prepare ourselves for this kind of error.

## FileNotFoundError Exception

When we are working with files, it is very common to have missing files issues. This may occur due to a different storage location of the file or maybe you have misspelled the file name or the file may not exist at all. Let's try an example and read a file named 'popeye.txt'. Now the below program is trying to read the file 'popeye.txt' but since we haven't saved the file let's see what happens:

```
f = open("popeye.txt")
print(f.read())
```

OUTPUT:

```
Traceback (most recent call last):
  File "C:\Users\Hira\Desktop\Python Projects\Test\main.py", line 57, in
    f = open("popeye.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'popeye.txt'
```

The output reports a FileNotFoundError exception. To minimize this exception, the **try-except** block will be useful so that the program doesn't generate any error.

## Try-Except Code Blocks

When you think that there are chances of having an error then you can use a try-except block to handle it. You can tell python what to 'try' and what not 'except' a certain exception.

For Example

```
try:
    print(10/0)
except ZeroDivisionError:
    print("You had a zero error")
```

OUTPUT:

```
You had a zero error
```

You can use exceptions to prevent programs from crashing. This is very reason why we need exceptions in the first place.

## ***Else Block***

To make more sense of the try-except block, we can introduce an else statement with it as well to or calculate the answer if the given condition is met otherwise.

For Example:

```
x = 10
```

```
y = 0
```

```
try:
```

```
    result = x/y
```

```
except ZeroDivisionError:
```

```
    print("You have a zero division error")
```

```
else:
```

```
    print(result)
```

**OUTPUT:**

You have a zero-division error

## Finally Exception

After trying out `try-except` and `try-except-else` blocks, it is time to learn about the `finally` block. This block will always execute no matter whether `try` turns out to be true or false.

For Example:

```
x = 10
y = 0
try:
    result = x/y
except ZeroDivisionError:
    print("You have a zero division error")
else:
    print(result)
finally:
    print("The exception handling is finished")
```

OUTPUT:

```
You have a zero-division error
The exception handling is finished
```

## More Built-in-Exceptions

Here is a list of more python built-in-exceptions that can help you minimize errors in your programs. Check out more on [Python Exceptions Docs](#)

EXCEPTION	DESCRIPTION
AssertionError	Raised when the assert statement fails.
AttributeError	Raised on the attribute assignment or reference fails.
EOFError	Raised when the input() function hits the end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raised when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when the index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits the interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in the local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when a system operation causes a system-related error.

EXCEPTION	DESCRIPTION
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by the next() function to indicate that there is no further item to be returned by the iterator.
SyntaxError	Raised by the parser when a syntax error is encountered.
IndentationError	Raised when there is an incorrect indentation.
TabError	Raised when the indentation consists of inconsistent tabs and spaces.
SystemError	Raised when the interpreter detects internal error.
SystemExit	Raised by the sys.exit() function.
TypeError	Raised when a function or operation is applied to an object of an incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translation.
ValueError	Raised when a function gets an argument of correct type but improper value.
ZeroDivisionError	Raised when the second operand of a division or module operation is zero.

# Conclusion

This book may not contain all your answers regarding Deep Python, but will surely make you familiar with the initial programming concepts and syntax of how simple Python is to start with as this book is a beginner's guide to launch in Python. You can always refer to the online resources like [StackOverflow](#), [Quora](#) and [Python.org](#) to learn more about other branches (Libraries) that will help you learn python even in more depth.

