PYTHON

INTRODUCTION TO PYTHON

- Python is a general purpose programming language that is often applied in scripting roles.
- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.It's an open source language from beginning.

PYTHON CAN DO-

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

PYTHON IS USED FOR-

- Web application
- Data Science
- Software Development
- Gaming
- Analytics
- Artificial Intelligence
- Gaming

WHY PYTHON-

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

PYTHON PACKAGES FOR DATA ANALYSIS-

Being a general purpose language Python is often used beyond data analysis and data science. Abundant availability of libraries makes Python remarkably useful for working with data functionalities. The significant Python libraries that are used for working with data.

Numpy – this library provides fundamental scientific computing.

Matplotlib – used for plotting and visualization.

Pandas – applied for data manipulation and analysis.

Scikit-learn – library designed for machine learning and data mining.

StatsModels – packed with statistical modelling, testing, and analysis

Scipy-SciPy is a bunch of mathematical algorithms and convenience functions built on the Numpy extension of Python.

Seaborn-Seaborn is mostly used for the visualization of statistical models.

Plotly-a web-based toolbox for constructing visualizations.

PYTHON SYTNTAX-

print("Hello, World!")

PYTHON IDENTATION-

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

```
if 7 > 3:
  print(“Seven is greater than three!")
```

Note: Python will give you an error if you skip the indentation

Error: Without Indentation it will give error like below

```
if 7 > 2:
print(" Seven is greater than three!")
```

Note: The number of spaces is up to you as a programmer, but it has to be at least one

```
if 7 > 2:
 print(“Seven is greater than three!")
```

```
if 7 > 2:
      print(“Seven is greater than three!")
```

Note: You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Syntax Error:

```
if 7> 2:
 print(“Seven is greater than three!")
      print(“Seven is greater than three!")
```

PYTHON VARRIABLE-

CREATING VARRIABLE

x = 10

y = “Fingertips"

print(x)

print(y).

- Variables are containers for storing data values.:
- Unlike other programming languages, Python has no command for declaring a variable
- A variable is created the moment you first assign a value to it.

- Variables do not need to be declared with any particular type and can even change type after they have been set.

x = 7 # x is of type int

x = “Fingertips" # x is now of type str

print(x)

- String variables can be declared either by using single or double quotes

x = “Rishi"

is the same as

x = ‘Rishi

VARRIABLE NAME-

- A variable can have a short name (like x and y) or a more descriptive name (age, companyname, total_profit).
- Let see Rules for Python variables
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (male, Male and MALE are three different variables

Legal variable names:

myvar = “Data"

my_var = “Data"

_my_var = "Data"

myVar = "Data"

MYVAR = "Data"

myvar2 = "Data"

#Illegal variable names:

2myvar = “Data"

my-var = “Data"

my var = “Data"

ASSIGN VALUE

Python allows you to assign values to multiple variables in one line:

a, b, c = “mango", “apple", “banana"

print(x)

print(y)

print(z)

And you can assign the same value to multiple variables in one line

```
a=b=c="mango"
print(a)
print(b)
print(c)
```

OUTPUT VARRIABLE-

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the +character:

x = “gocorona"

print("Python is " + x)

We can also use the + character to add a variable to another variable

```
x = “Go corona "
y = “asap"
z =  x + y
print(z)
```

The + character works as a mathematical operator

x = 15

y = 10

print(x + y

If we try to combine a string and a number, Python will give you an error

```
x = 5
y = "John"
print(x + y)
```

In Python, variables are created when you assign a value to it:

```
x = 5
```

```
y = "Hello, World!"
```

DATA TYPE-

- STRING
- NUMERIC
- SEQUENCE
- MAPPING
- SET
- BOOLEAN
- BINARY

GETTING DATA TYPE

You can get the data type of any object by using the type() function

```
A = 10
```

```
print(type(A))
```

NUMBER

A numeric value is any representation of data which has a numeric value.

Python identifies three types of numbers:

INTEGER

FLOT

COMPLEX NUMBER

Variables of numeric types are created when you assign a value to them:

```
x = 1    # int
```

```
y = 2.8  # float
```

```
z = 1j   # complex
```

To verify the type of any object in Python,use the type() function.

```
print(type(x))
```

```
print(type(y))
```

print(type(z)))

INTEGER-

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

x = 1

y = 35656222554887711

z = -3255522

print(type(x))

print(type(y))

print(type(z))

FLOT

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

x = 1.10

y = 1.0

z = -35.59

print(type(x))

print(type(y))

print(type(z))

STRING-

String literals in python are surrounded by either single quotation marks, or double quotation marks.

You can display a string literal with the print() function.

'hello' is the same as "hello".

print("Hello")

print('Hello')

ASSIGN STRINGS TO VARRIABLE

Assigning a string to a variable is done with the variable name followed by an equal sign and the string.

```
a = "Hello"

print(a)
```

MULTIPLE STRINGS

You can assign a multiline string to a variable by using three quotes:

```
a = """Hello."""

print(a)
```

Or three single quotes:

```
a = '''Hello'''
```

BOOLEANS-

- In programming you often need to know if an expression is True or False.
- You can evaluate any expression in Python, and get one of two answers True or False.
- When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
print(10 > 9)

print(10 == 9)

print(10 < 9)
```

SET-

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Sets are unordered, so you cannot be sure in which order the items will appear.

```
thisset = {"apple", "banana", "cherry"}

print(thisset
```

ACCESS ITEMS

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

```
thisset = {"apple", "banana", "cherry"}


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

CHECK

- Check if "banana" is present in the set
- Once a set is created, you cannot change its items, but you can add new items

thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)

ADD ITEMS

To add one item to a set use the add() method.

To add more than one item to a set use the update() method

thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)

_____

thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

print(thisset)

LENGTH AND REMOVE

To determine how many items a set has, use the len() method.

Remove "banana" by using the remove() method.

thisset = {"apple", "banana", "cherry"}

print(len(thisset)

_____

thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

```
print(thisset)
```

JOIN SET-

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another.

There are other methods that joins two sets and keeps ONLY the duplicates, or NEVER the duplicates, check the full list of set methods in the bottom of this page..

```
set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}


set3 = set1.union(set2)

print(set3)
```

_____

```
set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}


set1.update(set2)

print(set1)
```

SEQUENCE TYPE-

Sequence identifies three types of

1. LIST
2. TUPLE
3. RANGE

LIST

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Sets are unordered, so you cannot be sure in which order the items will appear.

```
thislist = ["apple", "banana", "cherry"]

print(thislist)
```

ACCESS ITEM-

You access the list items by referring to the index number:

```
thislist = ["apple", "banana", "cherry"]

print(thislist[1])
```

To verify the type of any object in Python,use the type() function.

```
print(type(x))

print(type(y))

print(type(z)))
```

You access the list items by referring to the index number:

```
thislist = ["apple", "banana", "cherry"]

print(thislist[1])
```

NEGATIVE INDEXING-

Negative indexing means beginning from the end,-1refers to the last item -2 refers to the second last item etc.

```
thislist = ["apple", "banana", "cherry"]

print(thislist[-1])
```

RANGE OF INDEXES –

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

print(thislist[2:5])
```

Note: The search will start at index 2 (included) and end at index 5 (not included)..

Remember that the first item has index 0.By leaving out the start value, the range will start at the first item:

RANGE OF INDEXES EX.

This example returns the items from the beginning to "orange":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

print(thislist[:4])
```

Example
This example returns the items from "cherry" and to the end:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Range of Negative Indexes
Specify negative indexes if you want to start the search from the end of the list:

Example
This example returns the items from index -4 (included) to index -1 (excluded)
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

Change Item Value
To change the value of a specific item, refer to the index number:

Example
Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

List Length
To determine how many items a list has, use the len() function:
Example
Print the number of items in the list:
```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Add Items
To add an item to the end of the list, use the append() method:
Example
Using the append() method to append an item

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

To add an item at the specified index, use the insert() method:
Example
Insert an item as the second position:
```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
```

```
thislist.insert(1, "orange")
print(thislist)
```

Remove Item

There are several methods to remove items from a list:

Example

The remove() method removes the specified item:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Example

The pop() method removes the specified index, (or the last item if index is not specified):

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

Example

The del keyword removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

Example

The del keyword can also delete the list completely

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Example

The clear() method empties the list:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Copy a List

Example

Make a copy of a list with the copy() method:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Another way to make a copy is to use the built-in method list()

```
thislist = ["apple", "banana", "cherry"]
```

```
mylist = list(thislist)
print(mylist)
```

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

Or you can use the extend() method, which purpose is to add elements from one list to another list:

Example

Use the extend() method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

List Methods

Python has a set of built-in methods that you can use on lists.

METHOD	DESCRIPTION
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list

METHOD	DESCRIPTION
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Assignment:List+ Loop

Loop Through a List

You can loop through the list items by using a for loop:

Example:

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
print(x)
```

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

Example

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
print("Yes, 'apple' is in the fruits list")
```

Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
```

```
for x in list2:
list1.append(x)


print(list1)
```

Tuple

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple
```

You can access tuple items by referring to the index number, inside square brackets:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

NEGATIVE INDEXING

Negative indexing means beginning from the end,-1 refers to the last item,-2 refers to the second last item etc.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

RANGE OF INDEXES

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items

Note:

The search will start at index 2 (included) and end at index 5 (not included)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

CHANGE TUPLE VALUES

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
```

```
x = tuple(y)

print(x)
```

LOOP THROUGH A TUPLE

You can loop through the tuple items by using a for loop.

```
thistuple = ("apple", "banana", "cherry")

for x in thistuple:

  print(x)
```

CHECK IF ITEM EXISTS

To determine if a specified item is present in a tuple use the in keyword.

```
thistuple = ("apple", "banana", "cherry")

if "apple" in thistuple:

  print("Yes, 'apple' is in the fruits tuple")
```

TUPLE LENGTH

To determine how many items a tuple has, use the len() method.

```
thistuple = ("apple", "banana", "cherry")

print(len(thistuple))
```

ADD ITEM

```
thistuple = ("apple", "banana", "cherry")

thistuple[3] = "orange" # This will raise an error

print(thistuple)
```

CREATE TUPLE WITH ONE ITEM

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)

print(type(thistuple))

#NOT a tuple

thistuple = ("apple")

print(type(thistuple))
```

REMOVE ITEMS

Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:**.**

The del keyword can delete the tuple completely

thistuple = ("apple", "banana", "cherry")

del thistuple

print(thistuple) #this will raise an error because the tuple no longer exists

JOIN TWO TUPLES-

To join two or more tuples you can use the + operator:

tuple1 = ("a", "b" , "c")

tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2

print(tuple3)

THE TUPLE() CONSTRUCTOR

It is also possible to use the tuple() constructor to make a tuple.

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets

print(thistuple))

TUPLE METHODS

Python has two built-in methods that you can use on tuples

METHOD	DESCRIPTION
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

DICTIONARY

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

ACCESSING ITEMS

You can access the items of a dictionary by referring to its key name, inside square brackets:

```
x = thisdict["model"]
```

There is also a method called get() that will give you the same result:

```
x = thisdict.get("model")
```

CHANGE VALUES-

You can change the value of a specific item by referring to its key name

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

LOOP THROUGH A DICTIONARY

We can loop through a dictionary by using a **for** loop

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

```
or x in thisdict:
  print(x)
for x in thisdict:
  print(thisdict[x])
```

ACCESSING ITEMS

We can also use the value() method to return values of a dictionary:

```
for x in thisdict.values():
  print(x)
```

Loop through both keys and values, by using the items() method

```
for x, y in thisdict.items():
  print(x, y)
```

CHECK IF KEY EXISTS

To determine if a specified key is present in a dictionary use the in keyword:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

DICTIONARY LENGTH

To determine how many items (key-value pairs) a dictionary has, use the len() function.

```
print(len(thisdict))
```

ADDING ITEMS

Adding an item to the dictionary is done by using a new index key and assigning a value to it.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

REMOVING ITEMS

There are several methods to remove items from a dictionary

The pop() method removes the item with the specified key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

The del() keyword removes the item with the specified key name

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

The clear() method empties the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

COPY A DICTIONARY

You cannot copy a dictionary simply by typing dict2=dict1,because dict2 will only be a reference to dict1 and changes made in dict1 will automatically also be made in dict2

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

A dictionary can also contain many dictionaries, this is called nested dictionaries

```
myfamily = {
  "child1" : {
    "name" : "Emil",
```

```
  "year" : 2004
 },
 "child2" : {
  "name" : "Tobias",
  "year" : 2007
 },
 "child3" : {
  "name" : "Linus",
  "year" : 2011
 }
}
```

NESTED DICTIONARIES

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

```
child1 = {
 "name" : "Emil",
 "year" : 2004
}
child2 = {
 "name" : "Tobias",
 "year" : 2007
}
child3 = {
 "name" : "Linus",
 "year" : 2011
}
myfamily = {
 "child1" : child1,
 "child2" : child2,
 "child3" : child3
```

}

The dict() Constructor

It is also possible to use the dict() constructor to make a new dictionary:

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
print(thisdict)
```

DICTIONARY METHODS

Python has a set of built-in methods that you can use on dictionaries.

METHOD	DESCRIPTION
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair

METHOD	DESCRIPTION
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs