1. Modules and Pip

- *Module :- codes that are written by professionals, libraries*
- *pip :- using pip we can import those modules*

```python
import pandas as pd                                    # type: ignore

# pandas is package name
# pd is alias , rather then using pandas , we use pd because we
just gave a short name for the package
```

2. Variables

- *Variables are containers that store information that can be manipulated and referenced later by the programmer within the code*
- *In python, it is not needed to declare variable type*

```python
name = "IRONMAN"   #type str
age = 20            #type int
passed = True       #type bool
```

```python
type(age) # data type
```

```
int
```

Scope of variables

- *can be created in a function that is local variable*
- *can be global variable and can be used anywhere in the code*

```python
a = "HI"   # global variable
def foods():                # function fun_name():
    vegetable = "Potato"    # local variable
    fruit = "Lichi"         # local variable
    print(a + vegetable + fruit)

foods()                     # function calling
```

```
HIPotatoLichi
```

3. Data Types

- *Data Types describes what type of data is stored in variable*

```python
# numeric data
int = 1 , 6, 9
float = 5.6, 5.7

# text data
str = "Hello"

# Boolean data
bool = True, False
```

Python

- *Sequenced Data Types :-*

```python
# List
# list is an ordered collection of data with elements separated
by comma and enclosed within square brackets
# list are mutable (Modified)

list = [8, 4, 5.2, ["apple", "banana"]]
list
```

Python

```
[8, 4, 5.2, ['apple', 'banana']]
```

```python
# tuple
# tuple is ordered collection of data with elements separated by
a comma and enclosed within parentheses
# Tuples are immutable(cannot be modified)

tuple = (("parrot", "sparrow"), ("Lion", "Tiger"))
tuple
```

Python

```
(('parrot', 'sparrow'), ('Lion', 'Tiger'))
```

```python
# range
# return a sequence of numbers as specified by user , if not it
starts from 0 and incremented by 1
# range(start, end, increment by)
sequence = range(1,10,2)

for i in sequence:
    print(i)
```

Python

```
1
3
5
7
9
```

```python
# dictionary
# dictionary is an collection of data containing a key : value
pairs enclosed in curly brackets

dictionary = {"name" : "Hitesh", "age" : "22"}
dictionary
```

Python

```
{'name': 'Hitesh', 'age': '22'}
```

4. User Input

```python
a = input(("Enter your name"))
print("enter your name", a )
```

Python

```
enter your name 5
```

5. Strings

```python
# Anything that can be enclosed between single or double
quotation marks is considered a string
# string is a sequence or array of textual data

name = "Hitesh"
print("enter your name", name)
print(name[0]) # first character of string
```

Python

```
enter your name Hitesh
H
```

6. IF Else Statement

```python
# statement use to execute code based on conditions
a=18
# a > 20 :- a greater than 20
# a < 20 :- a less than 20
# a <= 20 :- a less than or equals to 20
# a == 20 :- if a equals to 20
# a != 20 :- a not equal to 20
if(a == 18):     # if a is equal to 18
    print(" number is greater")

elif(a <= 15):   # if a is below or equals to 15
    print(" number is below 15")

else:            # if non of the above condition are true
    print("number is Smaller")
```

Python

```
number is greater
```

7. For Loop

```python
# for loop is a loop that execute for a specific number of
iterations
# good for repetitive work
name = 'Hitesh'

for i in name:    # i refer each item present in name
    print(i)
```

Python

```
H
i
t
e
s
h
```

```python
color = ["Red", "green", "blue", "yellow"]

for i in color:
    print(i)
```

Python

```
Red
green
blue
yellow
```

```python
# creates a range (start, end, steps)
for i in range(0, 11, 2):
    print(i)
```

Python

```
0
2
4
6
8
10
```

8. Break and Continue

```python
 # break is to stop the loop or stop the whole iteration
for i in range(15):
    if (i == 10):
        break              # loop break when i equals to 10
    print("5 * " , i , " =", 5 *i)
```

Python

```
5 *  0  = 0
5 *  1  = 5
5 *  2  = 10
5 *  3  = 15
5 *  4  = 20
5 *  5  = 25
5 *  6  = 30
5 *  7  = 35
5 *  8  = 40
5 *  9  = 45
```

```python
# continue will not stop the whole iteration, it will just skip
the iteration
for i in range(15):
    if (i == 10):
        continue
    print("5 * " , i , " =", 5 *i)
```

Python

```
5 *  0  = 0
5 *  1  = 5
5 *  2  = 10
5 *  3  = 15
5 *  4  = 20
5 *  5  = 25
5 *  6  = 30
5 *  7  = 35
5 *  8  = 40
5 *  9  = 45
5 *  11  = 55
5 *  12  = 60
5 *  13  = 65
5 *  14  = 70
```

9. Function and Argument

- *Function is a block of code that can be called many times to reduce the work of writing a code again and again*

```python
def average(a,b):                # function function_name
    print("Average is :", (a+b)/2)

average(5,10)  # 5 will be assigned to a , 10 will be assigned
to b in the function
```

Python

```
Average is : 7.5
```

```python
def average_tuple(*number):  # this will create a tuple
    print(type(number))
    sum = 0
    for i in number:
        sum = sum + i
    print("Average is :", sum / len(number))

average_tuple(1,2)
```

Python

```
<class 'tuple'>
Average is : 1.5
```

```python
def average_tuple1(*number):  # this will create a tuple
    print(type(number))       # type of variable
    sum = 0
    for i in number:
        sum = sum + i
    return sum / len(number)  # after computing this function
    will return the value

c = average_tuple1(5,10,20,5)
print(c)
```

Python

```
<class 'tuple'>
10.0
```

10. List

- *list is mutable , can be changed*
- *list can store different data types in them like int with string and many more*

```python
li = [1, 3, 5]
type(li)
li
```

Python

```
[1, 3, 5]
```

```python
print(li[0])  # 0 index element in list
```

Python

```
1
```

```python
li1 = [5 ,50.6, "HItesh"] # can store multiple data types
together
li1[2]
```

Python

```
'HItesh'
```

```python
if 1 in li:    # to check if a element is present in list
    print("Yes")
else:
    print("No")
```

Python

```
Yes
```

```python
if "Hit" in "Hitesh": # similarly for string we can check
    print("yes")
else:
    print("no")
```

Python

```
yes
```

```python
list1 = [1,2,3,4,5,6,7,8,9]

print(list1[:3]) # from start to 3-1 index = 0,1,2
print(list1[2:5]) # from 2 index to 5-1 index = 2,3,4

print(list1[1:8:2]) # start from 1st index upto 8th index but
jump by 2 indexes, knows as jump index
```

Python

```
[1, 2, 3]
[3, 4, 5]
[2, 4, 6, 8]
```

```python
# list comprehension :- on the way generate a list
lst = [i for i in range(10)] # values from 0 to 9 in lst
print(lst)

lst1 = [i for i in range(20) if i%2==0] # can give a condition
in list comprehension , number from 0 to 20, if divisible by 2
then only they can be element of list
print(lst1)
```

Python

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- List Methods

Append() :-

```python
# append () :-  to add elements in a list
list = [1,2,3,4]
print(list)

list.append(9) # using append we can add a element at the end of
list
print(list)
```

Python

```
[1, 2, 3, 4]
[1, 2, 3, 4, 9]
```

sort() :-

```python
# sort() :- to sort the list
list0 = [11,5,4,0,3,44]
print(list0)

list0.sort() # sort the list , ascending order
print(list0)

list0.sort(reverse=True) # sort the list in descending order
print(list0)
```

Python

```
[11, 5, 4, 0, 3, 44]
[0, 3, 4, 5, 11, 44]
[44, 11, 5, 4, 3, 0]
```

count() :-

```python
# count() :- to count how many times a element is present in list

list0 = [11,5,4,0,3,44]
print(list0)

print(list0.count(4)) # count how many times 4 is present
```

Python

```
[11, 5, 4, 0, 3, 44]
1
```

copy() :-

```python
# copy() :-  to copy a list to another list
# if direct copy using = the original list will also be changed

list0 = [11,5,4,0,3,44]
print(list0)

list1 = list0.copy()
print(list1)
```

Python

```
[11, 5, 4, 0, 3, 44]
[11, 5, 4, 0, 3, 44]
```

insert() :-

```python
# insert :- to inset a element at specific index

list0 = [11,5,4,0,3,44]
print(list0)

list0.insert(2, 100) # insert 100 at index 2
print(list0)
```

Python

```
[11, 5, 4, 0, 3, 44]
[11, 5, 100, 4, 0, 3, 44]
```

extend() :-

```python
# extend() :-add the content of a list at the end of another list

list0 = [11,5,4,0,3,44]
print(list0)

list1 = [100, 200, 300]
list0.extend(list1) # elements of list1 are added at the end of
list0
print(list0)
```

Python

```
[11, 5, 4, 0, 3, 44]
[11, 5, 4, 0, 3, 44, 100, 200, 300]
```

concatenate :-

```python
# concatenate list :- to concat two list
list0 = [11,5,4,0,3,44]
print(list0)

list1 = [100, 200, 300]

list2 = list0 + list1  # concatenating list0 and list1 into list2
print(list2)
```

Python

```
[11, 5, 4, 0, 3, 44]
[11, 5, 4, 0, 3, 44, 100, 200, 300]
```

11. Dictionary

- *Dictionary :- elements are stores in key : value pair { }*

```python
dict = {
    "name" : "Hitesh",
    "age" : 21,
    "Pincode" : 6994
}

print(dict)
```

```
{'name': 'Hitesh', 'age': 21, 'Pincode': 6994}
```

```python
print(dict["name"]) # search element by its key , value will be
returned
```

```
Hitesh
```

```python
print(dict.keys()) # return all keys
print(dict.values()) # return all values
```

```
dict_keys(['name', 'age', 'Pincode'])
dict_values(['Hitesh', 21, 6994])
```

```python
print(dict.items()) # return all items
```

```
dict_items([('name', 'Hitesh'), ('age', 21), ('Pincode', 6994)])
```

- *Dictionary Methods*

```python
dict1 = {
    "name" : "Hitesh",
    "age" : 21,
    "address" : "Kolhapur"
}
dict2 = {
    "name1" : "Rishikesh",
    "age2" : 22,
    "address2" : "Satara"
}
```

```python
dict1.update(dict2) # using update we can add elements of one
dictionary to another
dict1
```

Python

```
{'name': 'Hitesh',
 'age': 21,
 'address': 'Kolhapur',
 'name1': 'Rishikesh',
 'age2': 22,
 'address2': 'Satara'}
```

```python
dict2.clear() # clear the dictionary
print(dict2)
```

Python

```
{}
```

```python
dict3 = {} # creating a empty dictionary
print(dict3)
```

Python

```
{}
```

```python
dict1.pop("name") # using pop give the key and it will be
removed from the dictionary
print(dict1)
```

Python

```
{'age': 21, 'address': 'Kolhapur', 'name1': 'Rishikesh', 'age2': 22, 'a
```

```python
dict1.popitem() #pop item will remove last element from the
dictionary
print(dict1)
```

Python

```
{'age': 21, 'address': 'Kolhapur', 'name1': 'Rishikesh', 'age2': 22}
```

```python
del dict2["name1"] # delete the specific element, if no key is
given whole dictionary will be removed
print(dict2)
```

Python

12. Exceptional Handling

- *Exception handling is the process of responding to unwanted or unexpected events when a program runs*
- *This is used when you get an error message, instead what message will be printed*

```python
a = 5
for i in range(1,11):
    print(f"{int(a)} X {i} = {int(a) * i}")
```

Python

```
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
```

```python
# If we have an integer value it will give an Value error
b ="Hitesh"
for i in range(1,11):
    print(f"{int(b)} X {i} = {int(b) * i}")
```

Python

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

```python
# we use try and except here to handle this error
# try will run the program normally
# if any error occurs, except will catch the error and print
your custom message

# it will not show any error messages, it will print the custom
message you have provided
b ="Hitesh"
try:
    for i in range(1,11):
        print(f"{int(b)} X {i} = {int(b) * i}")
except:
    print("Enter a integer number instead of a string")
```

Python

```
Enter a integer number instead of a string
```

```python
# In python there are different errors like valueerror, keyboard
error, indexerror, etc for each type of error you can provide
custom message
try:
    num = int(input("Enter the number"))
    a = [6,3]
except ValueError:
    print("entered number is not a valid integer")

except IndexError:
    print("entered index is not a valid integer")
except:
    print("None of the above error")
```

Python

13. Virtual Environment

- *Virtual environment is a tool which creates a isolated environment which is different from the global environment that is installed in computer*
- *In virtual environment you can have your own python interpreter and can have different python versions , different versions of python libraries and frame works*

- *In computer you have pandas == 1.20*
- *but in virtual environment you can have pandas == 1.10*

- *both of them are fully isolated from each other*

1.To create a virtual environment, in cmd, terminal

python -m venv 'filename'

2.Activate virtual environment:- use python interpreter present in virtual environment

filename/Scripts/Activate

3.Deactivate virtual environment:- stop using python interpreter in virtual environment

deactivate

4.install python packages

pip install -r requirements.txt # list of required packages

or

pip install 'package name'

In virtual environment you have pip and setuptools preinstalled

Rest all the packages you have to install from basic numpy also

to open python interpreter in virtual environment

1.activate virtual environment

2.python

3.import package

4.print(package.__ version __)

14. Lambda Function

- *lambda function used to create mini functions and store in a variable*
- *Use lambda when you want function in one line*
- *use lambda when you want to pass a function as argument to another function*

```python
def double(x):
    return x*2

print(double(2))
```

Python

```
4
```

```python
# above code can be written using lambda function
double1 = lambda x: x*2

# print(double1(5))

cube = lambda x: x*x*x
print(cube(3))
```

Python

```
27
```

```python
# lambda function can accept multiple arguments
average = lambda x,y: (x + y)/2
print(average(5,10))
```

Python

```
7.5
```

```python
average1 = lambda x,y,z: (x + y + z)/2
print(average1(5,10,15))
```

Python

```
15.0
```

```python
def apply(fx, value):
    return 6+fx(value)

cube = lambda x: x*x*x

print(apply(cube, 2)) # cube is passed as argument to another
function
#or

print(apply(lambda x: x*x*x,2)) #direct function as argument to
another function, in this the inner function is anonymous it
does not have any name like 1st one
```

Python

```
14
14
```

15. MAP, Filter, Reduce

- *known as higher order functions, they accept function as argument*

Map :- *a function which allows to apply function on each element of a sequence like list, dictionary, etc*

```python
# we have a function
def cube(x):
    return x*x*x

l = [1,2,3,4,5,6,7,8,9]

# we want to apply cube function on each element of list
# we can do with for loop, but it is very hectic
# instead we use map function

# map(function_name, List_name)

newlist = map(cube,l) # it gives a map object you can specify
your own data type

# map / apply cube function on each element of list and convert
the map object to list data type and store it in newlist1
newlist1 = list(map(cube, l))
print(newlist1)
```

Python

```
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Filter :- *filters a sequence of elements based on a given condition , returns elements which matches the condition*
return a Boolean value for each element and if true then keep element

```python
def filter_function(a):
    return a>200

# it return a filter object you can convert it into any datatype
you want

newlist0 = list(filter(filter_function,newlist1))

# filter the list and put only elements which match the condition

print(newlist0)
```

Python

```
[216, 343, 512, 729]
```

Reduce :- *applies a function to a sequence and return a single value*

from functools import reduce

```
# map :- apply a function to each element and return the same
number of elements

# reduce :- apply a function to each element and return single
value

from functools import reduce

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

def mysum(x,y):
    return x + y

sum = reduce(mysum, numbers)

# apply mysum function to each and every element in list and
return a single value
# it applies function again and again to adjesant elements and
perform the function on it

# x and y , 1st element as x, 2nd element as y, add them
# that added value stored in x , next value stored in y, add them
# this process will repeat again and again and it will return
single value
print(sum)
```

Python

```
45
```

16. With Statement

- *with statement in Python replaces a try-catch block with a simple way*
- *it ensures that resources are closed immediately after processing.*

```
# open a python file , model = read, store in f
# perform read operation and it will be successfully closed
# print all the components present in that file
with open('./Hitesh.py', mode='r') as f:
    data = f.read()
    print(data)
```

```
def welcome():
    print("Welcome to my code")
```