Python Features

Python is a general purpose,dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming.

1.Easy to Code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

2.Free and Open Source

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

3.Object-Oriented Language:

One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

4.GUI Programming Support:

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

5.Python is Portable language:

Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

6.Large Standard Library

Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

7.Dynamically Typed Language:

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don’t need to

specify the type of variable.

Python history

Python was created in the early 1990s by **Guido van Rossum** at Stichting Mathematisch Centrum (CWI) in the Netherlands as a successor of a language called ABC.

It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

https://thelead.io/data-science/companies-that-uses-python

Core Python Programming

Add Comments

In [ ]:
```
#single line comment -1
#single line comment -2
""" Multiline Comments
Line 1
Line 2
Line 3 """
print("Hello World!")
```

Printing Basic Data Types

In [ ]:
```
print("Hello")
print("Welcome "+ "to Data Science training") # string data type
```

In [ ]:
```
print(1900+69) # integer data type
```

In [ ]:
```
print(55/34.0) # float data type
```

In [ ]:
```
print(True or False) # boolean data type
print(True and False)
```

Variables and Inputs

In [ ]:
```
school = "MIT"
print(school)
print(type(school)) # school variable belongs to string data type
print(id(school)) # unique id for the variable
```

In [ ]:
```
another_school="Stanford"
print(another_school)
print(id(another_school))
```

In [ ]:
```
print("I studied at " +school )
```

In [ ]:
```
my_age=44
print(type(my_age))
```

In [ ]:
```
my_salary= 10500.50
```

```
print(type(my_salary))
```

In [ ]:
```
is_manager= True # boolean Variable
print(type(is_manager))
print(is_manager)
```

In [ ]:
```
myname = input("enter your name ")
```

In [ ]:
```
print(myname+" is my name and i am studying at " + school)
```

Data Type Conversion

In [ ]:
```
#hours_per_day = input("How many hours do you study per day?")
hours_per_day = int(input("How many hours do you study per day?"))

print(hours_per_day)
print(type(hours_per_day))
```

In [ ]:
```
hours_per_week = hours_per_day * 7
print(hours_per_week)
```

In [ ]:
```
#print("i am studying at " + school + "and i study" + hours_per_week + "every week")
print("i am studying at " + school + " and i study " + str(hours_per_week) + " hours
```

Operators

In [ ]:
```
a=13
b=5
print("a=",a,"b=",b)
```

Arithemetic Operators

In [ ]:
```
print("Addition: a+b = ", a+b)
print("Subraction: a-b = ", a-b)
print("Multiplication: a*b = ", a*b)
print("Division: a/b = ", a/b)
print("Integer Division: a//b = ", a//b)
print("Modulus: a%b = ", a%b)
print("Exponent: a**b = ", a**b)
```

In [ ]:
```
# Write a program to get 2 numbers from the user and divide them to find the quotien
```

Assignment Operators

In [ ]:
```
x,y,z=20,10,5
print("x=",x,"y=",y,"z=",z)
```

In [ ]:
```
x,y,z=20,10,5
z=z*2
z*=2
print(z)
```

Unary minus Operator

In [ ]:
```
a=-1
print("a=",a)
```

Relational Operators

In [ ]:
```
a=13
b=5

print("a>b = ",a>b)
print("a>=b = ",a>=b)

print("a<b = ",a<b)
print("a<=b = ",a<=b)

print("a==b = ",a==b)
print("a!=b = ",a!=b)

#Relational Operators can be chained
print("0<a<=20 = ",0<a<=20)
print("0<b<1 = ",0<b<1)
```

Logical Operators

In [ ]:
```
### Logical Operators are useful to construct the compound conditions. A compound co
### is a combination of more than one simple condition

x=100
y=100
print("x>100 and y>100 ="  , x>99 and y>200)
print("x>100 and y>100 ="  , x>99 or y>200)
print("not(x>100 and y>100) ="  , not(x>99 or y>200))
```

In [ ]:
```
science_mark=int(input("Enter your science mark : "))
maths_mark=int(input("Enter your maths mark : "))

if science_mark > 35 : # relational Operator
    science_pass = True
else:
    science_pass = False

if maths_mark > 35 :
    maths_pass = True
else:
    maths_pass = False

if(science_pass and maths_pass ):# Logical Operators
    print("Passed")
else:
    print("failed")

if(science_pass or maths_pass ):# Logical Operators
    print("Passed")
else:
    print("failed")
```

Boolean Operators

In [ ]:
```
science_pass= True
maths_pass= False

### Boolean operators acts on boolean type literals( True , False) and returns Boole
print("science_pass and maths_pass =", science_pass and maths_pass)
print("science_pass or maths_pass =", science_pass or maths_pass)
```

Membership Operators

In [ ]:
```
### The membership operators are useful to test for membership in a sequence like st
### membership operators are
### in
### not in

names=["Mark","Bill","Thomas","Edison"]

myname="Mark"

yourname = "Bill"

print(myname in names)

print(yourname not in names)
```

Identity Operators

In [ ]:
```
### These operators compare the memory location of two objects
### it is possible to find out whether 2 variables pointing to the same object or no

### Identity Operators are
###  is ---> both objects are the same
###  is not --> if both objects are different

a=25
b=25

print(a is b)

print("ID of a =", id(a))
print("ID of b =", id(b))
```

In [ ]:
```
b=26

print(a is b)

print("ID of a =", id(a))
print("ID of b =", id(b))
```

Other Mathematic Functions

In [ ]:
```
### Operators are very handy when we do the fundamental operations.
### but we can use built in functions given in python for various advanced operation

import math
print("the square root of 16 =", math.sqrt(16))

import math as m
print("the square root of 25 =", m.sqrt(25))

from math import sqrt # only sqrt function will be imported from that library
print("the square root of 36 =", sqrt(36))
```

Operator Precedence and Associativity

https://www.programiz.com/python-programming/precedence-associativity

Exercise#1

Write a program to get ENGLISH,GERMAN,MATHS,SCIENCE,HISTORY marks from a student and print the total Marks and Percentage.

Input and Output Statements

To provide input to a computer, python provides some statements which are called input statements similarly to display the output it provides some output statements

print()

In [ ]:
```
### Output statements are

print()
print("Python")
```

In [ ]:
```
firstname,lastname="Nikola","Tesla"
print(firstname,lastname)
print(firstname,lastname,sep="~")
print()
```

In [ ]:
```
### to print the above in the same line
print("Welcome")
print("to")
print("Core Python")
```

In [ ]:
```
#print object
myLst=["Edison","Einstein","Newton"]
print(myLst)
print()
```

In [ ]:
```
#print formatted statement
# print("formatted string" %(variable list))
# %i or %d - integer
# %s - String
# $f - floatVa
fname = "Thomas"
lname = 'Edison'
print("Name = %s %s:" %(fname,lname))
print()
```

In [ ]:
```
x=10

print(x,fname)

# we can display single character from the string
print("my x value = %s and my fname = %s " %(x,fname))
```

In [ ]:
```
salary = 156999.8344556
print("Salary=%f:" %(salary))
```

In [ ]:
```
# formatted string with replacement field
# print("formatted string {0} with replacement".format(values))

fname = "Thomas"
lname = 'Edison'
print("Scientist Name :  {0} Alva {1}".format(fname,lname))
```

input()

In [ ]:
```
# Input Statements
# use input() to accepts the value from the keyboard and returns it as string

salary = float(input("Enter your Salary : "))
print("Salary = %14.2f" %(salary))
```

In [ ]:
```
# use the below syntax to accept more than one input values from the user

fname,lname = [str(name) for name in input("Enter Full Name :").split(',')]
print("fname =",fname )
print("lname =",lname )
```

In [ ]:
```
x,y = [int(x) for x in input("Enter 2 Numbers :").split(',')]
print("x+y=",x+y )
```

In [ ]:
```
Total = sum([int(x) for x in input("Enter 2 Numbers :").split(',')])
print(Total)
```

In [ ]:
```
mylist = list([int(x) for x in input("Enter 2 Numbers :").split(',')])
print(mylist)
```

In [ ]:
```
mylist = list([str(x) for x in input("Enter multiple names :").split(',')])
mylist.reverse()
print(mylist)
```

In [ ]:
```
# use the eval() along with input function to accept string from the user and execut
x = eval(input("Enter an expression : "))
print("Result = ", x)
```

Command Line Arguments

In [ ]:
```
# command line arguments are passed to the program from outside. All the arguments a
# a list with the name "argv" which is available in the sys module.
# argv[0] - name of the program
# argv[1] - first argument
# argv[2] - second argument...

# len(argv)-1 -> number of arguments passed by the user

# parsing command line arguments using argparse module
# python args.py 2 3
'''
import argparse

parser = argparse.ArgumentParser()

parser.add_argument('nums',nargs=2)

args=parser.parse_args()

print("Number=",args.nums[0])
print("Its Power", args.nums[1])

result= float(args.nums[0])**float(args.nums[1])

print("Results =",result)
'''
```

Control Statements

In python, usually the statements in the program are normally executed one by one from top to bottow. this type of execution is called as "Sequential execution". it may be suitable for simple programs But for complex program we should be able to change the flow of execution as we needed.i.e we should be able to repeat the group of statements multiple times or we may want to directly jump from one statement to another. for this purpose we have control statements

if

In [ ]:
```
science_mark=int(input("Enter your science mark : "))

if science_mark > 35 : # relational Operator
    print("You passed in Science")
    print("Congrats !!!")

print("************Program Ends***********")
```

if...else

In [ ]:
```
science_mark=int(input("Enter your science mark : "))

if science_mark > 35 : # relational Operator
    print("You passed in Science")
    print("Congrats !!!")
else:
    print("You Failed in Science")
    print("Better Luck next time")

print("************Program Ends***********")
```

if...elif...else

In [ ]:
```
science_mark=int(input("Enter your science mark : "))

if science_mark > 80 :
    print("You passed in Science with GRADE A")
    if science_mark > 90:
        print("super")
    else:
        print("Excellent")
elif science_mark > 60:
    print("You passed in Science with GRADE B")
    print("Very Good Marks")
elif science_mark > 36:
    print("You passed in Science with GRADE C")
    print("Try to score more")
else:
    print("You failed")
    print("Better Luck next time")


print("************Program Ends***********88")
```

while

it is useful to execute set of statements multiple times

1. while loop - it will gets executed unless the condition become false

2. for loop - it will execute the statements repeatedly depending upon the number of elements in the sequence

In [ ]:
```
#while loop example
# to display all numbers between 2 numbers

start_num= int(input("Enter the start number : "))
end_num= int(input("Enter the end number : "))

while start_num < end_num :
    print(start_num)
    start_num= start_num + 1


print("************Program Ends***********")

#Note: Beware of INFINITE loop while using WHILE Loops
```

for

In [ ]:
```
#for loop example
# program to display and find the sume of a list of numbers
mystring= input("Enter the string : ")
print_num= int(input("how many time you want to print ? : "))
for x in range(print_num):
    print(mystring)
print("outside")
print("************Program Ends***********")
```

break

In [ ]:
```
#while loop example
# to display all numbers between 2 numbers

start_num= int(input("Enter the start number : "))
end_num= int(input("Enter the end number : "))

while True :
    print(start_num)
    start_num= start_num + 1
    if start_num > end_num :
        break

print("************Program Ends***********")



#Note: Beware of INFINITE loop while using WHILE Loops
```

for...else with break

In [ ]:
```
myLst=[1,2,3,4,5,21,23]
num1=211
for x in myLst:
    if x==num1:
        print("Number found in the list")
        break;
else:
    print("Number not found in the list")
print("program ends")
```

continue

In [ ]:
```
# CONTINUE - this statement is used in a loop to go back to beginning of the LOOP.
# PASS - this statement does not do anything. it is used with the if statement or
# for statement inside a loop to do NO operation

#CONTINUE  example

mystring= input("Enter the string : ")
print_num= int(input("how many time you want to print ? : "))

myList = [1,2,-4,5,-8,2]
for num in myList:
    if num <= 0 :
        continue # moves to next iteration
    print(str(num)+" : " + mystring)

print("************Program Ends************")
```

In [ ]:
```
# CONTINUE - this statement is used in a loop to go back to beginning of the LOOP.
# PASS - this statement does not do anything. it is used with the if statement or
# for statement inside a loop to do NO operation

#CONTINUE  example

mystring= input("Enter the string : ")
print_num= int(input("how many time you want to print ? : "))

for num in range(print_num):
    if num % 2 == 0 :
        continue # moves to next iteration
    print(str(num+1)+" : " + mystring)

print("************Program Ends************")
```

Functions

Function is similar to program consist of group of statements to perform a specific tasks.

built-in functions --> print()

user-defined functions -- you can write your functions

Functions can be resused across the main program. it avoids code redundancy, easy to modify and improve maintenance.

"return" values from the function. in java we can return only one value from the function but in python we can return multiple values

Topics

- Multiple parameters
- Nested Functions
- Formal Parameters/Actual arguments
- pass by value/pass by reference
- Actual arguments - 4 types
  - positional args
  - keyword args

    - default args
    - variable length args
- function local and global variables-
- Anonymous function - Lambdas
    - with filter function
    - with Map function
    - with reduce function
- Modules

Define function

In [ ]:
```
def findMax(n1,n2):# formal args
    if n1 > n2 :
        return n1;
    else:
        return n2

#Main Program

input_num1 = int(input("Enter the First Number "))
input_num2 = int(input("Enter the Second Number "))

# calling the function from Main
print("Maximum Number is : ",findMax(input_num1,input_num2))
```

In [ ]:
```
def welcomeMsg(studentName): # fu
    WelcomeStr ='Hi '+studentName + ', Welcome to our School'
    return WelcomeStr


#Main Program Starts
student = input("Your Name : ")
print(welcomeMsg(student)) # calling the function
```

Functions are first class objects

in python, functions are considered as **objects**. infact when we define a function, python will create an object. so we can pass function to another function as we pass object. and also we can **return** a function from another function

In [ ]:
```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")

# main program starts
ip_name = input("Enter your name : ")
greet(ip_name)
```

Return statement

The return statement is used to exit a function and go back to the place from where it was called.

In [ ]:
```
def odd_even(num):
    """This function returns the absolute
    value of the entered number"""

    if num %2 == 0:
        return "Even"
    else:
        return "Odd"


print(odd_even(2))
print(odd_even(3))
```

Return Multiple Values

"return" values from the function. in java we can return only one value from the function but in python we can return multiple values

In [ ]:
```
# example for returning multiple values
def division(dividend,divisor):# formal args
    quotient = dividend//divisor
    remainder= dividend%divisor
    return quotient,remainder

#Main Program

i_dividend = int(input("Please enter the dividend : "))

i_divisor = int(input("Please enter the divisor : "))

o_quotient,o_remainder = division(i_dividend,i_divisor)

print("Quotient ={0}  and  Remainder = {1} ".format(o_quotient,o_remainder))
```

Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function are not visible from outside the function. Hence, they have a local scope. The lifetime of variables inside a function is as long as the function executes. They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Here is an example to illustrate the scope of a variable inside a function.

In [ ]:
```
def print_number():
    first_num = 1
    # Print statement 1
    print("The first number defined is: ", first_num)

print_number()
# Print statement 2
#print("The first number defined is: ", first_num)
```

Recursive and Nested Functions

In [ ]:
```
def factorial(number):

    def inner_factorial(number):
        if number <= 1:
```

```
            return 1
        return number*inner_factorial(number-1)
    return inner_factorial(number)

# Call the outer function.
print(factorial(4))
```

In [ ]:
```
# you can define function inside another function. this is called nested function
from datetime import *

def Welcome(f_name):
    def greetMsg():

        tdm = datetime.today()

        print("Time Now is : ",tdm)

        if tdm.strftime("%p")== "AM":
            return ". Good Morning"
        else:
            return ". Good Evening"

    return "Hello, " + f_name + greetMsg()
```

In [ ]:
```
#Formal and Actual arguments
# when we define function, we mention some parameter to receive data from outside. t
# when we CALL function, we pass some parameters to the function and these are calle

def sum(a,b): # a and b are formal arguments
    c=a+b
    print(c)
#main
x,y=10,15
sum(x,y) # x and y are actual arguments
```

Pass by Value and References

In [ ]:
```
# pass object by reference
#in java and c we can pass variables to a function either Pass by Value or Pass by r
#But in python, everything is objects like integer, string, float..they are immutabl
#mutable so object  can be modified.

# Integer Class is IMMUTABLE
# when we try to modify the integer object inside the function, new object will get
# samething will happen to float, string and tuple

def update_Age(age):

    age=16
    print("Inside age=",age,"ID=",id(age))

#Main Program Starts
age= 40
print("Outside Before Age=",age,"ID=",id(age))

# calling function
update_Age(age)

print("Outside After x=",age,"ID=",id(age))
print()
```

In [ ]:
```
# Since list is mutable object. passed object can be modified.
def modifyList(ageL):
```

```
    print("Inside age=",ageL,"ID=",id(ageL))
    ageL[0]=16
    print("After update Inside age=",ageL,"ID=",id(ageL))

ageL=[40]
print("Outside Before age =",ageL,"ID=",id(ageL))

# calling function
modifyList(ageL)

print("Outside After age=",ageL,"ID=",id(ageL))
print()
```

Function Arguments

In [ ]:
```
#Formal and Actual arguments
# when we define function, we mention some parameter to receive data from outside. t
# when we CALL function, we pass some parameters to the function and these are calle

def sum(a,b): # a and b are formal arguments
    c=a+b
    print(c)

#main
x,y=10,15
sum(x,y) # x and y are actual arguments
```

Actual arguments are of 4 types

1. **positional** --attach('New','york') - order is important
2. **keyword** -- grocery (item='sugar',price=50.75)--use the formal arguments name, order is not important
3. **default** arguments -- def grocery (item,,price=50.75)--when we define use some default value,if actual arguments is not there then default value will be used
4. **variable** length arguments--def add(farg,*args)---it can accept any number of arguments

In [ ]:
```
# for Positional Arguments
def grocery(itemParam,priceParam):
    print(str(itemParam) + " price is " + str(priceParam))

#Main
item = "Sugar"
price = 100
grocery(item,price)
grocery(price,item)
```

In [ ]:
```
#example for keyword arguments

def grocery(itemParam,priceParam):
    print(str(itemParam) + " price is " + str(priceParam))
#Main Program

item='Sugar'
price=100.45
grocery(itemParam=item,priceParam=price)
grocery(priceParam=price,itemParam=item)
```

In [ ]:
```
#example for Default arguments
def grocery(itemParam,priceParam=150.50):
    print(str(itemParam) + " price is " + str(priceParam))
```

```
item='Sugar'
grocery(item)# passing only one argument
```

In [ ]:
```
# example for variable length arguments

def add(farg,*args):
    print("First Argument : ",farg,"\nremaining arguments in Tuple : ",args)
    sum=farg
    for i in args:
        sum+=i
    print("Total =",sum)

#main program
add(4,3)
print("----------")
add(4,3,5)
print("----------")
add(1,2,4,3,5)
```

Local and Global Variables

when we declare a variable inside a function it becomes local variable. scope is limited to that function only. when we declare a variable outside a function it becomes GLOBAL variable. it can be accessed from entire program written below. if the Same variable name given inside a function, then GLOBAL keyword can be prefixed to access global varible from inside a function

In [ ]:
```
# example for local variable
def displayStudentDetails():

    global studentAge
    studentAge=20
    print("Student Age inside Function = " + str(studentAge))

#main program starts
displayStudentDetails()
print(studentAge)
```

Anonymous Functions or Lamdas

A function without name is called Lambda functions. functions are defined using Lambda (not using def)

-Python code to illustrate cube of a number -showing difference between def() and lambda().

def cube(y): return yyy;

g = lambda x: xxx print(g(7)) print(cube(5))

normal function returns values. but Lamda function returns "FUnction" so it should be assigned to function variable

Without using Lambda : Here, both of them returns the cube of a given number. But, while using def, we needed to define a function with a name cube and needed to pass a value to it. After execution, we also needed to return the result from where the function was called using the return keyword.

Using Lambda : Lambda definition does not include a “return” statement, it always contains an expression which is returned. We can also put a lambda definition anywhere a function is expected, and we don’t have to assign it to a variable at all. This is the simplicity of lambda functions.

In [ ]:
```
sqr= lambda x: x**2 # step#1 defining the function which returns the function ptr
print(sqr(5)) # call the function
```

In [ ]:
```
# max of 2 numbers
max=lambda x,y :x if x>y else y # step#1 defining the function which returns the fun
print(max(8,7))# call the function
```

In [ ]:
```
# use lambda function with filter function
# syntax for filter function is filter(function_name,sequence)
# it applies functions to all the elements inthe sequence

#filter the even numbers

mylist=[2,3,4,5,6,7,8] # mylist is a sequence

for i in mylist:
    if i%2==0:
        print(i)
```

In [ ]:
```
#print(list(filter(lambda x:(x%2==0),mylist))) # only divisible by 2 is filtered

print(list(filter(lambda x:(x%2==0),mylist)))
```

In [ ]:
```
# lamda with filter - example #2
# Take a list of numbers.
my_list = [12, 65, 54, 39, 102, 339, 221, 50, 70, ]

# use anonymous function to filter and comparing
# if divisible by 13 or not
result = list(filter(lambda x: (x % 13 == 0), my_list))

# printing the result
print(result)
```

In [ ]:
```
# using lambda with Map function
# the map function is similar to filter function but it acts on each element of the

# map( function,sequence)'

# lambda to return the squares
# map(lamdafunction, list)


#map(lambda x:x**2,lst1 )

lst1=[10,2,3,4,5]
lst2= list(map(lambda x:x**2,lst1 ))
print(lst2)
#the difference between map and filter. map applies to all elements of the sequence
#filter applies a condition to all elements and return only element which satisified
```

In [ ]:
```
# using lambda to reduce function
# the reduce function reduces a sequences to single value by processing the elements
#reduce(function,sequence)

import functools
```

```
lst=[1,2,3,4,5,6,7,8,9]
i=functools.reduce(lambda x,y: x*y,lst)
print(i)
```

In [ ]:
```
# Tag Function Name into another variable

def printMsg(msg):
    print(msg)


# main program starts
printMsg("From PrintMsg")

duplicate = printMsg

duplicate("From duplicate")
```

In [ ]:
```
# function as parameter
def inc(x):
    return x + 1

def dec(x):
    return x - 1

def operate(func, x):
    result = func(x)
    return result

#Main Program Starts
print(operate(inc,5))

print(operate(dec,5))
```

Modules and Packages

Modules in Python are simply Python files with the .py extension, which implement a set of functions. Modules are imported from other modules using the import command. Before you go ahead and import modules, check out the full list of built-in modules in the Python Standard library.

When a module is loaded into a running script for the first time, it is initialized by executing the code in the module once. If another module in your code imports the same module again, it will not be loaded twice but once only - so local variables inside the module act as a "singleton" - they are initialized only once.

If we want to import module math, we simply import the module:

In [ ]:
```
# import the library
import math
#use it (ceiling rounding)
math.ceil(2.4)
```

Exploring built-in modules While exploring modules in Python, two important functions come in handy - the dir and help functions. dir functions show which functions are implemented in each module. Let us see the below example and understand better.

In [ ]:
```
print(dir(math))
```

When we find the function in the module we want to use, we can read about it more using the

help function, inside the Python interpreter:

In [ ]:
```
help(math.ceil)
```

In [ ]:
```
import sys
print(sys.path)
```

In [ ]:
```
'''
import mymodule as my
my.greeting("Tesla")
'''
```

In [ ]:
```
'''
from mymodule import *
greeting("Tesla")
'''
```

Sequences Data Types

Arrays

In [ ]:
```
# creating an integer array

import array
a = array.array('i',[4,6,2,9])


print(a[0])
```

In [ ]:
```
# creating array from another array
import array as ar

arr1=ar.array('d',[1.5,2.5,3.5])

arr2=ar.array(arr1.typecode,(a*3 for a in arr1))

for i in arr2:
    print(i)
```

In [ ]:
```
# slicing operation

import array as ar
x=ar.array('i',[10,20,30,40,50,60,70,80,90,100])
print(x[1:4])
print(x[:4])
print(x[1:])
print(x[-3:-2])
print(x[1:5:2])
```

Strings

String represents group of characters str represents String datatype.

"Apple",'Ball' or """Cat""" can be used to string assignment.

In [ ]:
```
mystr = "Python"
print(mystr)
```

```
mystr = "Python"
```

In [ ]:
```
for i in mystr:
    print(i)
```

In [ ]:
```
#Length of the string
n= len(mystr)
print("length of the string", n)
```

In [ ]:
```
mystr = '1234567890'
print(mystr[0:5:2])
```

In [ ]:
```
# slicing the string

# stringname[start:stop:stepsize]

mystr = 'core python'

print("output is ", mystr[0:10:2])
print("output is ", mystr[0:])
print("output is ", mystr[0:11:2])
print("output is ", mystr[::-1])
print("output is ", mystr[-6::])
```

In [ ]:
```
mystr = 'Edison'
print("output of mystr[::-1]is ", mystr[::-1])
```

In [ ]:
```
mystr = 'Edison'
print("output of mystr[::-1] is ", mystr[-1:-3:-1])
```

In [ ]:
```
# String Concatenation

mystr1 = 'core'
mystr2 = 'Python'
mystr = mystr1 + mystr2

print(mystr)
```

In [ ]:
```
# Repeating the string

mystr = 'corepython'
print(mystr*5)
```

In [ ]:
```
# comparing string

s1 = "Thomas"
s2 = "Thomas"

if s1==s2:
    print("Both Strings are same")
else:
    print("Both are different")
```

In [ ]:
```
# Removing spaces from string .lstrip(), .rstrip() and strip()

s1 = " Thomas "
s2 = "Thomas"

if s1.strip()==s2:
    print("Both Strings are same")
else:
    print("Both are different")
```

In [ ]:
```python
# Finding substring using find(),rfind(),index(),rindex()
mystr = " welcome to the core programming core"
n=mystr.find("core",20,40)
print("substring found in the location",n)
```

In [ ]:
```python
# using count method
mystr=" welcome to core python core"
n= mystr.count("core")
print(n)

n= mystr.count("core",0,22)
print(n)
```

In [ ]:
```python
# using index
try:
    n=mystr.index("cores",0,len(mystr))
    print("substring found in the location",n)
except ValueError:
    print("Substring not found")
```

In [ ]:
```python
# strings are immutable
# the content of the string character by character cannot be changes after it got as

mystr= "Welcome to core python programming"
#mystr[11]="J" #this is not supported..you cannot modify the content by character

mystr = "python" #you can reassign with new string.
mystr = "java"
```

In [ ]:
```python
# Splitting and Joining Strings

mystr = "Welcome,to, core,python,programming"
mylist = mystr.split(",")

print(mylist)
```

In [ ]:
```python
#using join method of string
newstr = ":".join(mylist)
print(newstr)
```

In [ ]:
```python
# changing the case of a string .upper(), .lower(), .swapcase

mystr="python is the future"

print(mystr.upper())
print(mystr.lower())
print(mystr.swapcase())
print(mystr.title())
```

In [ ]:
```python
# startswith and endswith

print(mystr.startswith("python"))

print(mystr.endswith("future")) # case sensitive
```

In [ ]:
```python
## String testing methods

#isalnum(),#isalpha(),#isdigit(),#islower(),#isupper(),#istitle(),#isspace()

mystr=input("enter your number : ")

if mystr.isdigit() and len(mystr)==10:
```

```
    print("valid number")
else:
    print("invalid number")
```

List

Lists is very similar to array but with one main difference list can store different types of elements.

In [ ]:
```
#Creating a list
# using square brackets
mylist=[]
print(type(mylist))
```

In [ ]:
```
mylist=[10,20,10.5,2.55,"Tom","Bill"]
print("List created using square brackets",mylist)
```

In [ ]:
```
# list function
mylist=list(range(4,9,2))
print("List created using range function",mylist)
```

In [ ]:
```
#Accessing one by one
mylist=[10,20,10.5,2.55,"Tom","Bill"]
for element in mylist:
    print(element)
```

In [ ]:
```
#retrieve the elements in the reverse order
mylist=[10,20,10.5,2.55,"Tom","Bill"]
print(mylist)
mylist.reverse()
print(mylist)
```

In [ ]:
```
# updating the elements of a list

# lists are Mutable i.e you can insert, update and delete the elements of a list
mylist=[10,20,10.5,2.55,"Tom","Bill"]
```

In [ ]:
```
mylist.append(35)
print("Appended 35", mylist)
```

In [ ]:
```
#update 1st element
mylist[0]=15
print("updated the first element ",mylist)
```

In [ ]:
```
#deleting an element using del statement
del mylist[1:4]
print("deleted 2nd element using del command")
```

In [ ]:
```
#deleting elements using remove method. for remove method you have to pass value and
# if there is duplicate values in the list then it removes only  the first occurence
try:
    mylist.remove(35)
    print("deleted element which contains 35 in first occurence",mylist)
except:
    print("not exists")
```

In [ ]:
```
mylist.insert(1,45)
print("Appended 35", mylist)
```

In [ ]:
```
# Concatenation of two list
x=[10,20,30,40,50]
y=[110,120,130]
print("concatenated two list x + y" , x+y)
```

In [ ]:
```
# Repetition of list using *
print("repetition of list x*2 = ", x*2)
```

In [ ]:
```
# membership in list
mylist=[10,20,30,40,50]

num=20

if num in mylist:
    print(" Num found in the list")
else:
    print(" Num not found in the list")
```

In [ ]:
```
#ALiasing and cloning list
mylist_alias=mylist

if mylist is mylist_alias:
    print("both mylist and mylist_alias are same")
else:
    print("both are not same")

mylist[1]=22

print(mylist)
print(mylist_alias)
```

In [ ]:
```
#ALiasing and cloning list
mylist_alias=mylist.copy()

if mylist is mylist_alias:
    print("both mylist and mylist_alias are same")
else:
    print("both are not same")

mylist[1]=32

print(mylist)
print(mylist_alias)
```

In [ ]:
```
### List Methods with examples

x=[10,20,30,40,50]

n = len(x)
print("using command len(x) is ",n)
```

In [ ]:
```
#### index()- returns the index of the element which has given value x
print("using x.index(30) is ", x.index(30))
```

In [ ]:
```
#### insert(i,x)- add one element with value x in the given index(i)
x.insert(3,35)
print("using x.insert(3,35) is ",x )
```

In [ ]:
```
#### extend(list1)- append the list1 to the list
y=[70,80,85]
```

```
x.extend(y)
print("using x.extend(y) is ",x )
```

In [ ]:
```
#### count() - number of  times the given value in the list
print("using x.count(30) is ",x.count(30) )
```

In [ ]:
```
#### remove(x)- remove the first element which has x value
try:
    x.remove(30)
    print("using x.remove(30) is ",x )
except:
    print("already removed")
```

In [ ]:
```
#### pop()- remove the last element
x.pop()
print("using x.pop() is ",x )
```

In [ ]:
```
#### reverse() - reverse the sequences of the elements
x.reverse()
print("after x.reverse() is ",x )
```

In [ ]:
```
#### sort()- sort the elements of list in ascending order
x.sort()
print("after x.sort() is ",x )

x.sort(reverse=True)
print("after x.sort(reverse=True) is ",x )
```

In [ ]:
```
#### clear()- delete all the elements from the list
x.clear()
print("after x.clear() is ",x )
```

Tuples

tuples is similar to lists but they are immutable. after its creation we cannot modify its elements. so we cannot perform the insert, append, delete, remove, pop, clear on the tuples.

In [ ]:
```
# Tuples Creation
mytuple=(10,20.-30.2,40.5,"India","China")
print(mytuple)
print(type(mytuple))
```

In [ ]:
```
# it is also possible to create tuple from the list
mylist=[10,20,30,"Mark"]
mytuple = tuple(mystr)
print(mytuple)
```

In [ ]:
```
another_tuple=(mytuple,11)
print(another_tuple)
print(another_tuple[0][0])
```

In [ ]:
```
#if we dont mention any brackets, by default it will take it as tuple
tup5=10,20,30,"Sara"
print(tup5)
```

In [ ]:
```
##### Accessing Tuple elements- very similar to string, array and list slicing.

mytuple=(50,60,70,80,90,100)
```

```
print(mytuple[5:0:-1])
```

Mapping DataTypes

Dictionary

A dictionary represents a group of elements arranged in the form of key-value pair. First element is considered as KEY and immediate next element as VALUE. Key and Vaue is separated by : All the key-Value pairs are inserted within ex: mydict = { key1:value1,Key2-Value2 }

In [ ]:
```
# Create Dictionary with employee details and retrieve it

mydict = {'EmpName':'Edison','EmpId':200,'EmpSalary':9502.50}
mydict["Age"] = 50

mydict["Age"] = 45

mydict["mylist"] = mylist

print(mydict)
#Access value by dictionary key
print("Name of the Employee:",mydict['EmpName'])
print("ID of the Employee:",mydict['EmpId'])
print("Salary of the Employee:",mydict['EmpSalary'])
```

In [ ]:
```
mydict = {'EmpName':'Edison','EmpId':200,'EmpSalary':9502.50}
# len function
print("Using Len function" , len(mydict))
```

In [ ]:
```
# Insert a new key value
mydict["Dept"]="IT"
print("After inserting the dept value", mydict)
```

In [ ]:
```
# Deleting a key value pair

del mydict["EmpSalary"]

print("After the deleting the Salary value",mydict)
```

In [ ]:
```
# items()
print(mydict.items())
```

In [ ]:
```
### keys and values
print(mydict.keys())
print(mydict.values())
```

In [ ]:
```
# A python function accepts dictionary and display its elements

def fun(dict):
    for i,j in dict.items():
        print(i,"---",j)

fun(mydict)
```

Sets

Set is an unordered collection of elements much like a set in mathematics. The order of elements is not in the sets.Sets does not accept duplicates

1. set datatype : can be modified

In [ ]:
```
s={10,20,30,40,50,20}
print(s)

ch= set("Hello")
print(ch)

# Set is Mutable
ch.remove("H")
print(ch)
```

In [ ]:
```
ch.add('H')
print(ch)
```

Object Oriented Programming

C,Pascal,Fortran are called procedure oriented programming languages the main task in the program is divided in to several subtasks and each sub tasks is represented as procedure or function

C++, Java and python uses classes and objects the main tasks is divided in to multiple sub tasks and these are represented as Classes. Each class can perform several interrelated tasks for which several methods are written in a class

when program becomes bigger, then more task need to be achieved for that more code will be written and less reusabllity.

Another Approach Object Oriented approach is very closer to human being point of view. this approach will be reuse the code and manage them easily.

Class and Objects

In [ ]:
```
class Employee:
    def __init__(self):
        self.name = "sara"
        self.age = 34
        self.salary = 10000

# main program starts
emp1 = Employee()
print(emp1.name,emp1.age,emp1.salary)
```

In [ ]:
```
class Employee_new:
    def __init__(self,name,age,salary):
        self.name = name
        self.age = age
        self.salary = salary

# main program starts
emp1 = Employee_new("vignesh",24,14000)
emp2 = Employee_new("Ravi",35,15000)
emp3 = Employee_new("Bala",36,16000)
```

```
print(emp1.name,emp1.age,emp1.salary)
print(emp2.name,emp2.age,emp2.salary)
print(emp3.name,emp3.age,emp3.salary)

emp2.salary = 25000

print(emp2.name,emp2.age,emp2.salary)
```

In [ ]:
```
class Employee:
    # class variables
    #but these variable carried over to objects and you can access object variable u

    def __init__(self,name,age,sal): # constructor method
        self.name=name
        self.age=age
        self.sal=sal

    def talk(self):
        print(self.name,self.age,self.sal)

#main program starts

emp1 = Employee("Tom",50,12000) # instanstiation emp1 is object
emp2 = Employee("Bill",52,15000) # instanstiation emp2 is object
emp3 = Employee("Mark",52,25000) # instanstiation emp3 is object

print(emp1.talk)
print(emp2.talk)
```

In [ ]:
```
class Employee:
    #but these variable carried over to objects and you can access object variable u
    objCount=0

    def __init__(self,name,age,sal):
        self.name=name
        self.age=age
        self.sal=sal
        self.bonus = 0.0
        Employee.objCount+=1


    def calculateBonus(self):
        self.bonus = self.sal*.5 + self.sal
        print(self.bonus)
```

In [ ]:
```
emp1 = Employee("Tom",50,12000) # instanstiation emp1 is object
emp2 = Employee("Bill",52,15000) # instanstiation emp2 is object
emp3 = Employee("Mark",52,25000) # instanstiation emp3 is object
```

In [ ]:
```
print("EMP1.NAME =",emp1.name)
print("EMP2.NAME =",emp2.name)
print("EMP3.NAME =",emp3.name)
print("Object Created=",Employee.objCount)
```

In [ ]:
```
# calling object methods
emp1.calculateBonus()
```

Encapsulation

Encapsulation is a mechanism where the data(variables) and the code (method) that act on the data are bind together All the member variables and functions are Public by default.(Uniform Access Principle in python)

In [ ]:
```
class Student:
    # class variables
    #but these variable carried over to objects and you can access object variable u

    def __init__(self,name,age,rollno):
        self.name=name
        self.age=age
        self.rollno = rollno

    def displayInfo(self):
        print(self.name,self.age,self.rollno)
```

In [ ]:
```
stud1 = Student("Tom",50,"87BL38") # instanstiation student1 is object
stud2 = Student("Bill",52,"87BL33") # instanstiation student2 is object
stud3 = Student("Mark",52,"87BL39") # instanstiation student3 is object
```

In [ ]:
```
# All the member variables and functions are Public by default.(Uniform Access Princ
stud1.name
```

In [ ]:
```
# Public methodss can be accessed from outside
stud1.displayInfo()
```

Abstraction

Class may contain many data.but the user may not need all the data. so programmer can hide some unnecessary data from the user. this is called an abstraction

example: car, dashboard and engine. user wants to view the dashboard but not the engine details

example. bank clerk wants to view the customer account details. but he should not see some other details like credit card number or any other account holder critical info.

In [ ]:
```
class BankAccount:
    def __init__(self,accno,name,balance):
        self.accno=10
        self.name= name
        self.__balance=10000.00
    def displaytoClerk(self):
        print(self.accno,self.name,self.__balance)
```

In [ ]:
```
account1 = BankAccount("ACC12345","Tom",10000.00)
account2 = BankAccount("ACC56789","Bill",20000.00)
```

In [ ]:
```
# public variable can be accessed from outside
account1.name
```

In [ ]:
```
# private variable cannot be accessed from outside
try:
    account1.__balance
except:
    print("cannot can access private variable")
```

In [ ]:
```

```

In [ ]:
```
# private variables can be accessed through methods
account1.displaytoClerk()
```

In [ ]:
```
## Inheritance

class Employee: # Parent Class or Super Class
    name="Alice" # ist variable
    age=50  # 2nd variable


    def display(self):
        print(self.name,self.age)

class Salesman(Employee):
    target=100000  #age=50  # 3rd variable


    def disptarget(self):
        print(self.target)

s1=Salesman()

#print(s1.name,s1.age,s1.target)

s1.display() ## it displays 3 values name, age and Target

s1.disptarget() ## it displays 3 values name, age and Target
```

Inheritance

In [ ]:
```
class Student:
    def __init__(self,id,name,city): # Constructor
        print("entering base class constructor")
        self.id = id
        self.name = name
        self.city = city
    def display(self): # instance method
        print("ID = {} , NAME is  = {} , CITY = {} ".format(self.id, self.name, self
```

In [ ]:
```
# main program
# creating/instantiation an object (Student1)
student1 = Student(1,"Edison","Paris")
student1.display()
```

In [ ]:
```
# EngineeringStudent class derived from Student class
## Student class is called as BASE CLass or SUPER class
## EngineeringStudent class is called as derived class or SUB class

# Note: if we there is NO constructor in the derived class then base class construct
# automatically. but if we defined derived class constructor then we need to call ba

class EngineeringStudent(Student):
    def __init__(self,id,name,city,marks):
        print("entering Derived class constructor")
        super().__init__(id,name,city) # calling the base class constructor
        print("entering back to Derived class constructor")
        self.marks=marks
    def display(self):
        super().display()
        print(" Marks = {}".format(self.marks))
```

In [ ]:
```
# main program
```

```
EngineeringStudent1 = EngineeringStudent(1,"Edison","Paris",450)
EngineeringStudent1.display()
```

Polymorphism

In [ ]:
```
class Animal:
    def speak(self):
        print("speaking")


class Dog(Animal):
    def speak(self):
        print("barking")


class Cat(Animal):
    def speak(self):
        print("Meowing")

mydog = Dog()
mycat= Cat()

animals=[mycat,mydog]

for i in animals:
    i.speak()
```

Abstract method and Abstract Class

- Abstract method is a method which not defined. it is method written without body and that body will be redefined in the subclass
- Abstract class is a class which contains some abstract methods
- use decorator @abstractmethod
- PVM cannot create objects for the abstract class

In [ ]:
```
from abc import ABC, abstractmethod

# defining Abstract class by defining Abstract Method
class Shape(ABC):
    @abstractmethod
    def draw(self):
        pass
    @abstractmethod
    def paint(self):
        pass

class circle(Shape):
    def draw(self):
        print("drawing the circle")
    def paint(self):
        print("painting Circle")


class Square(Shape):
    def draw(self):
        print("drawing a Square")
    def paint(self):
```

```
        print("painting Square")


circleObj= circle()
circleObj.draw()
circleObj.paint()

squareObj= Square()
squareObj.draw()
squareObj.paint()
```

Interfaces

In [ ]:

```
# Python program showing
# abstract base class work
# Abstract Base classes(ABC)

from abc import ABC, abstractmethod

class Polygon(ABC):

        # abstract method
        def noofsides(self):
                pass

class Triangle(Polygon):

        # overriding abstract method
        def noofsides(self):
                print("I have 3 sides")

class Pentagon(Polygon):

        # overriding abstract method
        def noofsides(self):
                print("I have 5 sides")

class Hexagon(Polygon):

        # overriding abstract method
        def noofsides(self):
                print("I have 6 sides")

class Quadrilateral(Polygon):

        # overriding abstract method
        def noofsides(self):
                print("I have 4 sides")


myPolygon = []

# Driver code
myPolygon.append(Triangle())
myPolygon.append(Quadrilateral())
myPolygon.append(Pentagon())
myPolygon.append(Hexagon())

for shape in myPolygon:
    shape.noofsides()
```

Exceptions

Generally we classify the error in the program

1. Compile time errors
2. Runtime errors
3. Logical errors

CompileTime errors

Syntax errors if you forget the ":" after the conditional statements then during compilation, compiler would throw this error with line number and error message.

Compile Time Errors

In [ ]:
```
# Compile time errors example
for x in range(10):# colon missing
    print(x) # indendation missing
```

Run Time Error

When PVM cannot execute the byte code then it flags the runtime error. Note: Runtime errors are not detected by compiler.

In [ ]:
```
# Runtime error example -1 --> invalid literal for int() with base 10: 's'
x = int(input("Enter first value : "))
y = int(input("Enter the second value : "))
```

In [ ]:
```
# Runtime error example -2  --> IndexError: list index out of range

animal=["dog","cat","elephant","horse"]

print(animal)
# print(animal[4])
```

logical errors

flaw in the logic of the program ex. programmer used the wrong formula. both compiler and pVM cannot detects this error

In [ ]:
```
# logical error example
# incrment the employee salary by 15% and display the new salary
current_salary= float(input("Enter the employee salary:"))

new_salary =  current_salary*.5
# correct logic
# new_salary = current_salary + current_salary*.15

print("the new salary with 15% increase is :", new_salary)
```

In [ ]:
```
try:
    x=-1
    y= x/0

except ZeroDivisionError:
    print("Dont enter Zero")
```

Exception Handling

compile time errors and logical errors can be corrected by modifying the code.but runtime error bound to happen, so programmer should know which type of error might occur and handle them using exception handling.

In [ ]:
```
try:
    x = int(input("Enter first value : "))
    y = int(input("Enter the second v2alue : "))
except ValueError :
    print("you cannot enter characters ")
except:
    print("some other error")
```

In [ ]:
```
f=open("myfile.txt","w")
a,b=[int(x) for x in input("enter 2 numbers : ").split(',')]
c=a/b
f.write("writing %d into myfile" %c)
f.close()
print("file closed")
```

All exceptions are represented as classes in python. The exception which are already available are called as "built in" exceptions

In [ ]:
```
try:
    x=int(input("Enter a number between 1 and 150 : "))
    assert x>=1 and x<=150
except AssertionError :
    print("the number should be within 1 and 150")
except:
    print("Some error. contact Administrator")
finally:
    print("closing files")
```

File Handling

There are 2 types of files

1. Text files- stores in ASCII characters
2. Binary Files - in bytes ex. it can be used to store images, audio and video

File Handler = open("file name", "open mode","buffering")

r - read mode - file pointer at the beginning of the file

w - write mode - creates file or delete the content of the existing file.

a - append mode - adding to the end of the file. if file does not exist then it creates it.

w+ - write and read mode; previous file content will be deleted

r+ - read and write : previous content will not be deleted: pointer will be at the beginning

a+ - append and read: file pointer will be at the end of file if the file exist. else create new file.

default buffering is 4096 bytes.

Mounting Google Drive

In [ ]:
```
from google.colab import drive
```

```
drive.mount('/content/drive', force_remount=True)
```

Reading Text Files

In [ ]:
```
with open('/content/drive/My Drive/fromcolab.txt','w') as f:
    f.write('Hello Google Drive!\n')
    f.write('Hello Python!\n')
    f.write('Hello Machine Learning!\n')
```

In [ ]:
```
f = open('/content/drive/My Drive/fromcolab.txt', 'r')
print(f.read())
```

In [ ]:
```
f = open('/content/drive/My Drive/fromcolab.txt', 'r')
print(f.readline())
print(f.readline())
print(f.readline())
```

Reading XML File

In [ ]:
```
#Python code to illustrate parsing of XML files
import xml.etree.ElementTree as ET
tree = ET.parse('/content/drive/My Drive/00-MASTER/DATA/Sample-XML-Files.xml')
root = tree.getroot()
for child in root:
    print("entering")
    for country in root.findall('country'):
        print("next level")
        rank = country.find('rank').text
        year = country.find('year').text
        name = country.get('name')
        print(name,rank,year)
```

Reading PDF files

In [ ]:
```
! pip install PyPDF2
```

In [ ]:
```
# importing required modules
import PyPDF2

# creating a pdf file object
pdfFileObj = open('/content/drive/My Drive/00-MASTER/DATA/Blog.pdf', 'rb')

# creating a pdf reader object
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)

# printing number of pages in pdf file
print(pdfReader.numPages)

# creating a page object
pageObj = pdfReader.getPage(0)

pageObj.extractText()

# extracting text from page
print(pageObj.extractText())

# closing the pdf file object
pdfFileObj.close()
```

Reading json files

In [ ]:
```
# Python program to read
# json file


import json
# returns JSON object as
# a dictionary

data = json.load(open('/content/drive/My Drive/00-MASTER/DATA/Emp.json', encoding='u

# Iterating through the json
# list
print(type(data))

for i,j in data.items():
    print(i,j)
```

OS Library

In [ ]:
```
import os
os.listdir(os.getcwd())
```