**PYTHON LANGUAGE PROGRAMMING**

1. Demonstrate the working of ‘id’ and ‘type’ functions

z=10

y=z

k=int(z)

**if** id(z)==id(y):

print(**True**)

**else**:

print(**False**)

**if** id(k)==id(y):

print(**True**)

**else**:

print(**False**)

OUTPUT:-

True

True

2. To find all prime numbers within a given range.

n,m=map(int,input().split())

**for** i **in** range(n,m):

c=0

**for** j **in** range(2,i-1):

**if** i%j==0:

c=c+1

**if**(c<1):

print (I)

OUTPUT:-

2 20

2

3

5

7

11

13

17

19

3. To print ‘n terms of Fibonacci series using iteration.

**def** fibo(n):

**if**(n==0):

print(**"0"**)

**elif**(n==1):

print(**"0 \n** **1"**)

**else**:

a=0

b=1

print(str(a)+**"\n"**+str(b))

a=int(a)

b=int(b)

**for** i **in** range(0,n-2):

c=a+b

print(str(c)+**" "**)

a=b

b=c

n=int(input(**"enter no"**))

fibo(n)

OUTPUT:-

enter no8

0

1

1

2

3

5

8

13

4. To demonstrate use of slicing in string

string=**"i am a progrmmer"**

print(string[1:])

print(string[:-1])

print(string[-4:-1])

print(string[1:4])

print(string[1:8:2])

print(string[5:8])

print(string[-8:-4])

print(string[-1:])

print(string[9:10])

OUTPUT:-

am a progrmmer

i am a progrmme

mme

am

map

a p

rogr

r

o

5. a. To add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged.

Sample String : 'abc'

Expected Result : 'abcing'

Sample String : 'string'

Expected Result : 'stringly'

k=input()

n=len(k)

**if** n>=3:

l=k[-3:]

**if**(l==**'ing'**):

print(k.replace(**"ing"**, **"ly"**, 1))

**else**:

k = k + **"ing"**

print(k)

**else**:

print(k)

OUTPUT:-

reading

readly

b. To get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.

l=input()

s=l[0]

l=l.replace(s,**"$"**)

l=l.replace(**"$"**,s,1)

print(l)

OUTPUT:-

ababababaaaaabb

ab$b$b$b$$$$$bb

6. a. To compute the frequency of the words from the input. The output should output after sorting the key alphanumerically.

*# Python code to find frequency of each word*

**def** freq(str):

str = str.split()

str2 = []

**for** i **in** str:

**if** i **not in** str2:

str2.append(i)

d={}

**for** i **in** range(0,len(str2)):

d[str.count(str2[i])]=str2[i]

**for** i **in** sorted(d): *# dictionary sort*

print((i, d[i]), end=**" "**)

str = **'apple mango apple orange orange apple guava mango mango apple'**

freq(str)

OUTPUT:-

(1, 'guava') (2, 'orange') (3, 'mango') (4, 'apple')

b. Write a program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.

h=input();

l=h.split(**","**)

l.sort()

k=**","**.join(l)

print(k)

OUTPUT:-

i,am,proud,to,bean,Indian

Indian,am,bean,i,proud,to

7.Write a program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically.

**def** remove\_duplicates (list):

output = []

**for** value **in** list:

**if** value **not in** output:

output.append(value)

**return** output

x=input(**"enter the word seperated with white spaces "**)

list=x.split(sep=**" "** )

result = remove\_duplicates(list)

result.sort()

print(result)

OUTPUT:-

enter the word seperated with white spaces aa dd s s vv eee ww q qq

['aa', 'dd', 'eee', 'q', 'qq', 's', 'vv', 'ww']

8.To demonstrate use of list &amp; related functions.

list=[1,4,5,3,2]

*#indexing*

print(list[0])

print(list[-1])

*#sorting function*

list.sort()

print(list)

*# append function*

list.append(9)

print(list)

*# extend function*

list.extend([10,11])

print(list)

**del**(list[2])*#delete element from list*

print(list)

list.reverse()*#reverse*

print(list)

print(3 **in** list)*#membership*

print(sum(list))*#sum function*

OUTPUT:-

1

2

[1, 2, 3, 4, 5]

[1, 2, 3, 4, 5, 9]

[1, 2, 3, 4, 5, 9, 10, 11]

[1, 2, 4, 5, 9, 10, 11]

[11, 10, 9, 5, 4, 2, 1]

False

42

9.To demonstrate use of Dictionary& related functions

d={1:[1,2,3,4],2:[74,2,4],3:**'hello'**,4:(**'singh'**,56,89.3)}

print(d)

print(d[4])

citypop={11:**'new york'**,22:**'uttar pradesh'**,13:**'dehli'**,4:**'jummu kashmir'**}

k=len(citypop)

**del** citypop[11]

print(citypop)

citypop[7]=**'kanpur'**

citypop.pop(22)

citypop[5]=**'nagpur'**

citypop.popitem()

citypop[4]=**'nagpur'***#replace*

print(citypop)

city={18:**'lko'**,9:**'gzb'**,10:**'hjs'**}

citypop.update(city)

print(citypop)

print(city)

empty={}

em={}

empty=citypop.copy()

print(citypop.keys())

em=citypop.keys()

l=[]

print(empty)

**for** i **in** sorted (citypop) :*#dictionary sort*

print ((i, citypop[i]), end =**" "**)

print(citypop)

OUTPUT:-

{1: [1, 2, 3, 4], 2: [74, 2, 4], 3: 'hello', 4: ('singh', 56, 89.3)}

('singh', 56, 89.3)

{22: 'uttar pradesh', 13: 'dehli', 4: 'jummu kashmir'}

{13: 'dehli', 4: 'nagpur', 7: 'kanpur'}

{13: 'dehli', 4: 'nagpur', 7: 'kanpur', 18: 'lko', 9: 'gzb', 10: 'hjs'}

{18: 'lko', 9: 'gzb', 10: 'hjs'}

dict\_keys([13, 4, 7, 18, 9, 10])

{13: 'dehli', 4: 'nagpur', 7: 'kanpur', 18: 'lko', 9: 'gzb', 10: 'hjs'}

(4, 'nagpur') (7, 'kanpur') (9, 'gzb') (10, 'hjs') (13, 'dehli') (18, 'lko') {13: 'dehli', 4: 'nagpur', 7: 'kanpur', 18: 'lko', 9: 'gzb', 10: 'hjs'}

10.To demonstrate use of tuple, set& related functions.

a. TUPUL DEMOSTRATION

tup1 = ('physics', 'chemistry', 1997, 2000)

tup2 = (1, 2, 3, 4, 5, 6, 7 )

print ("tup1[0]: ", tup1[0])

print ("tup2[1:5]: ", tup2[1:5])

tup3 = tup1 + tup2

print("tup3", tup3)

tup = ('physics', 'chemistry', 1997, 2000);

print (tup);

del (tup);

print (" deleting tup ");

#print (tup);

tup = ('physics', 'chemistry', 1997, 2000);

print(tup[:])

print(tup[1:])

print(tup[:-1])

print(tup[:-2])

print(tup[3:])

lod=(464,291,93,3789)

k=len(lod)

print(k)

h=max(lod)

print(h)

l=min(lod)

print(l)

OUTPUT:-

tup1[0]: physics

tup2[1:5]: (2, 3, 4, 5)

tup3 ('physics', 'chemistry', 1997, 2000, 1, 2, 3, 4, 5, 6, 7)

('physics', 'chemistry', 1997, 2000)

deleting tup

('physics', 'chemistry', 1997, 2000)

('chemistry', 1997, 2000)

('physics', 'chemistry', 1997)

('physics', 'chemistry')

(2000,)

4

3789

93

b.SET DEMOSTRATION

Days=set(["Mon","Tue","Wed","Thu","Fri","Sat"])

Months={"Jan","Feb","Mar"}

Dates={21,22,17}

print(Days)

print(Months)

Days.add("Sun")

print(Days)

Days.discard("Sun")

print(Days)

Days.discard("Wed")

Days.discard("Mon")

Days.discard("Sat")

daysA=set(["Mon","Tue","Sat"])

union=daysA|Days

print(union)

inter= daysA & Days

print(inter)

diff = daysA - Days

print(diff)

Days.add("Sun")

Days.add("Wed")

Days.add("Mon")

Days.add("Sat")

SubsetRes = daysA <= Days

SupersetRes = Days >= daysA

print(SubsetRes)

print(SupersetRes)

OUTPUT:-

{'Tue', 'Fri', 'Thu', 'Wed', 'Mon', 'Sat'}

{'Jan', 'Feb', 'Mar'}

{'Tue', 'Fri', 'Thu', 'Wed', 'Mon', 'Sat', 'Sun'}

{'Tue', 'Fri', 'Thu', 'Wed', 'Mon', 'Sat'}

{'Tue', 'Fri', 'Mon', 'Thu', 'Sat'}

{'Tue'}

{'Mon', 'Sat'}

True

True

**11.To implement stack using list**

class stack:

def \_\_init\_\_(self):

self.list=[]

def push(self,key):

self.list.append(key)

def pop(self):

self.list.pop()

def topitem(self):

print(self.list[-1])

def isempty(self):

if not list:

print("empty")

else:

print("not empty")

def display(self):

print(self.list)

# main program

author = stack()

while True:

print('write push <value> for push, pop for pop ,quit for stop process')

do = input('what u want ').split()

operation = do[0].strip().lower()

if operation == 'push':

author.push(int(do[1]))

elif operation == 'pop':

print("stack is popoed ", author.pop() )

elif operation == 'quit':

break

print('list is',end=" ")

author.isempty()

author.display()

print("top element is",end=" ")

author.topitem()

OUTPUT:-

write push <value> for push, pop for pop ,quit for stop process

what u want push 4

write push <value> for push, pop for pop ,quit for stop process

what u want push 67

write push <value> for push, pop for pop ,quit for stop process

what u want push 23

write push <value> for push, pop for pop ,quit for stop process

what u want pop

stack is popoed None

write push <value> for push, pop for pop ,quit for stop process

what u want push 55

write push <value> for push, pop for pop ,quit for stop process

what u want quit

list is not empty

[4, 67, 55]

top element is 55

**12.To implement queue using list**

class stack:

def \_\_init\_\_(self):

self.list=[]

def push(self,key):

self.list.append(key)

def pop(self):

self.list.pop(0)

def topitem(self):

print(self.list[-1])

def isempty(self):

if not list:

print("empty")

else:

print("not empty")

def display(self):

print(self.list)

# main program

author = stack()

while True:

print('write push <value> for push, pop for pop ,quit for stop process')

do = input('what u want ').split()

operation = do[0].strip().lower()

if operation == 'push':

author.push(int(do[1]))

elif operation == 'pop':

print("stack is popoed", author.pop())

elif operation == 'quit':

break

print('list is',end=" ")

author.isempty()

author.display()

print("top element is",end=" ")

author.topitem()

OUTPUT:-

write push <value> for push, pop for pop ,quit for stop process

what u want push 56

write push <value> for push, pop for pop ,quit for stop process

what u want push 32

write push <value> for push, pop for pop ,quit for stop process

what u want pop

stack is popoed None

write push <value> for push, pop for pop ,quit for stop process

what u want push 44

write push <value> for push, pop for pop ,quit for stop process

what u want quit

list is not empty

[32, 44]

top element is 44

**13.To read and write from a file.**

**Theory:-**

* We declared the variable f to open a file named textfile.txt. Open takes 2 arguments, the file that we want to open and a string that represents the kinds of permission or operation we want to do on the file
* Here we used "w" letter in our argument, which indicates write and the plus sign that means it will create a file if it does not exist in library
* The available option beside "w" are "r" for read and "a" for append and plus sign means if it is not there then create it.
* f.read() is for reading the content of file.
* f.close() is for closing the instance of file.

afile=open(**'hello.txt'**,encoding=**'utf-8'**,mode=**'w'**)

afile.write(**"India is Great"**)

afile.close()

afile=open(**'hello.txt'**,encoding=**'utf-8'**)

print(afile.read())

afile.seek(5)

print(afile.read(5))

print(afile.tell())

afile.close()

afile=open(**'hello.txt'**,**'r'**)

print(afile.read())

afile.close()

f=(open(**'ravi.txt'**,**'w'**))

f.write(**"ravi is good"**)

f.close()

f=open(**'ravi.txt'**,**'r'**)

print(f.read())

f.close()

f=open(**"ravi.txt"**,**'w'**)

f.write(**"ravi likes to sell milk"**)

f.close()

f=open(**"ravi.txt"**,**'r'**)

print(f.read())

OUTPUT:-

India is Great

is G

10

India is Great

ravi is good

ravi likes to sell milk

14. **To copy a file.**

**Theory:-In this program we copy the content of one file into another and print that content from another file.**

**f=open("ravi.txt",'r')**

f1=open(**"ravi1.txt"**,mode=**"w"**)

f1.write(f.read())

f1.close()

f1=open(**"ravi1.txt"**,**"r"**)

print(f1.read())

f1.close()

f.close()

OUTPUT:-

ravi likes to sell milk

15.**To demonstrate the working of class and objects.**

**Theory:-**

* Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.
* **We can create multiple different objects that are of the same class(have the same variables and functions defined). However, each object contains independent copies of the variables defined in the class.**

**class Student:**

nos = 0 *#class variable*

**def** \_\_init\_\_ (self,rollno,name): *#instance variables*

self.rollno = rollno

self.name =name

**def** prints(self):

print(self.rollno)

print(self.name)

Student.Nos = 1

s1 = Student(1,**"mayank"**)

s1.Nos =2

s2 = Student(2,**"rahul"**)

print(s1.\_\_dict\_\_)

print(s2.\_\_dict\_\_)

OUTPUT:-

{'rollno': 1, 'name': 'mayank', 'Nos': 2}

{'rollno': 2, 'name': 'rahul'}

**16.To demonstrate class method and static method.**

**Theory**:-**A class method** is a method which is bound to the class and not the object ofthe class. They have the access to the state of the class as it takes a class parameter thatpoints to the class and not the object instance.It can modify a class state that would apply across all the instances of the class.For example it can modify a class variable that will be applicable to all the instances.

A class method can access or modify class state while a static method can’t access or modify it.In general, static methods know nothing about class state. They are utility typemethods that take some parameters and work upon those parameters. On the other hand class methods must have class as parameter. We use @classmethod decorator in python to create a class method and weuse @staticmethod decorator to create a static method in python.

**Static Method:-A static method does not receive an implicit first argument. A static method is also a method which is bound to the class and not the objectof the class.A static method can’t access or modify class state. It is present in a class because it makes sense for the method to be present in class.**

**class Student:**

nos = 0 *#class variable*

**def** \_\_init\_\_ (self,rollno,name): *#instance variables*

self.rollno = rollno

self.name =name

**def** prints(self):

print(self.rollno)

print(self.name)

@classmethod

**def** increments(cls):

cls.nos=cls.nos +1

@staticmethod

**def** numbers(i):

D={1:**"one"**,2:**"two"**,3:**"three"**,4:**"four"**,5:**"five"**}

print(D[i])

s1=Student.increments()

s1 = Student(1,**"mayank"**)

s2=Student.increments()

s2 = Student(2,**"rahul"**)

s3=Student.increments()

print(s1.\_\_dict\_\_)

print(s2.\_\_dict\_\_)

Student.numbers(4)

**OUTPUT:-**

{'rollno': 1, 'name': 'mayank'}

{'rollno': 2, 'name': 'rahul'}

four

17.**To demonstrate constructors**

**Theory:-** Constructor is used for initializing the instance members when we create the object of a class. As we have seen in the above example that a constructor always has a name init and the name init is prefixed and suffixed with a double underscore(\_\_). We declare a constructor using def keyword, just like methods.

**Types of constructors :**

* **default constructor :**The default constructor is simple constructor which doesn’t accept any arguments.It’s definition has only one argument which is a reference to the instance being constructed.

**parameterized constructor :**constructor with parameters is known as parameterized constructor.The parameterized constructor take its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

**class rectangle:**

**def** \_\_init\_\_(self, length, width):

self.length=length

self.width=width

**def** area(self):

print(**"area of rectangle ="**+str(self.length\*self.width))

h=rectangle(3,4)

h.area()

OUTPUT:-

area of rectangle =12

18.**To demonstrate Inheritance.**

**Theory:-**Inheritance is the capability of one class to derive or inherit the properties from some another class. The benefits of inheritance are:

1. It represents real-world relationships well.
2. It provides **reusability** of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
3. It is transitive in nature, which means that if class B inherits from another class A,then all the subclasses of B would automatically inherit from class A.

The specialization and generalization relationships are both reciprocal and hierarchical. They are reciprocal because specialization is the obverse side of the coin from generalization. Thus, Dog and Cat specialize Mammal, and Mammal generalizes from Dog and Cat.

**class** Person:

**def** \_\_init\_\_(self, name, age):

self.name = name

self.age = age

**def** introduceyourself(self):

print(**"My name is "** + self.name)

print(**"My age is "** + str(self.age))

**class** Teacher(Person): *# this class inherits the class above!*

**def** \_\_init\_\_(self, name, age):

self.courses = [] *# initialise a new variable*

super().\_\_init\_\_(name, age) *# call the init of Person*

**def** stateprofession(self):

print(**"I am a student!"**)

**def** introduceyourself(self):

super().introduceyourself() *# call the introduceyourself() of the Person*

self.stateprofession()

print(**"I learn "** + str(self.nrofcourses()) + **" course(s)"**)

**for** course **in** self.courses:

print(**"I learn "** + course)

**def** addcourse(self, course):

self.courses.append(course)

**def** nrofcourses(self):

**return** len(self.courses)

author = Teacher(**"Mayank"**,19)

author.stateprofession()

author.addcourse(**"Python"**)

author.introduceyourself()

OUTPUT:-

I am a student!

My name is Mayank

My age is 19

I am a student!

I learn 1 course(s)

I learn Python

19. **To demonstrate aggregation and composition**

**Theory:-**Composition is a way of aggregating objects together by making some objects attributes of other objects. we can express a relationship between two classes using the phrase has-a, it is a composition relationship. According to some formal definitions the term composition implies that the two objects arequite strongly linked – one object can be thought of as belonging exclusively to the other object. If the owner object ceases to exist, the owned object will probably cease to exist as well. If the link between two objects is weaker, and neither object has exclusive ownership of the other, it can also be called aggregation.

Association is relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.  
In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. **Composition** and **Aggregation** are the two forms of association.

COMPOSITON--→AGGREGATION----→ASSOCIATION

class Math:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def add(self):

return self.x + self.y

def substract(self):

return self.x - self.y

class Math2:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def multiply(self):

return self.x \* self.y

def divide(self):

return self.x / self.y

class Math3:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

self.m1 = Math(x,y)

self.m2 = Math2(x,y)

def power(self):

return self.x \*\* self.y

def add(self):

return self.m1.add()

def substract(self):

return self.m1.substract()

def multiply(self):

return self.m2.multiply()

m1 = Math3(5,3)

print("add", m1.add())

print("multiply", m1.multiply())

print("substract", m1.substract())

print("power", m1.power())

OUTPUT:-

add 8

multiply 15

substract 2

power 125