OOP (Polymorphism and Encapsulation)

Polymorphism:

The word polymorphism means having many forms. In programming, polymorphism means the same function name (but different signatures) being used for different types.

Example of inbuilt polymorphic functions:

```
# len() being used for a string
print(len("Python"))

# len() being used for a list
print(len([10, 20, 30]))
```

Output:

```
6
3
```

Example of user defined polymorphic functions:

```
def add(x, y, z = 0):
  return x + y+z

# Driver code
print(add(2, 3))
print(add(2, 3, 4))
```

Output

```
5
9
```

Polymorphism with class methods

The below code shows how Python can use two different class types, in the same way. We create a for loop that iterates through a tuple of objects. Then call the methods without being concerned about which class type each object is. We assume that these methods actually exist in each class.

```
class India():
  def capital(self):
    print("New Delhi is the capital of India.")

  def language(self):
    print("Hindi is the most widely spoken language of India.")

  def type(self):
    print("India is a developing country.")
```

```
class USA():
  def capital(self):
    print("Washington, D.C. is the capital of USA.")

  def language(self):
    print("English is the primary language of USA.")

  def type(self):
    print("USA is a developed country.")

obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
  country.capital()
  country.language()
  country.type()
```

Output

```
New Delhi is the capital of India.
Hindi is the most widely spoken language of India.
India is a developing country.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
USA is a developed country.
```

Polymorphism with Inheritance

In Python, Polymorphism lets us define methods in the child class that have the same name as the methods in the parent class. In inheritance, the child class inherits the methods from the parent class. However, it is possible to modify a method in a child class that it has inherited from the parent class. This is particularly useful in cases where the method inherited from the parent class doesn't quite fit the child class. In such cases, we re-implement the method in the child class. This process of re-implementing a method in the child class is known as **Method Overriding**.

```
class Bird:
 def intro(self):
  print("There are many types of birds.")

 def flight(self):
  print("Most of the birds can fly but some cannot.")

class sparrow(Bird):
 def flight(self):
  print("Sparrows can fly.")

class ostrich(Bird):
 def flight(self):
  print("Ostriches cannot fly.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
obj_ost.flight()
```

Output

```
There are many types of birds.
Most of the birds can fly but some cannot.
There are many types of birds.
Sparrows can fly.
There are many types of birds.
Ostriches cannot fly.
```

Polymorphism with a Function and Objects

It is also possible to create a function that can take any object, allowing for polymorphism. In this example, let's create a function called "func()" which will take an object which we will name "obj". Though we are using the name 'obj', any instantiated object will be able to be called into this function. Next, let's give the function something to do that uses the 'obj' object we passed to it. In this case, let's call the three methods, viz., capital(), language() and type(), each of which is defined in the two classes 'India' and 'USA'. Next, let's create instantiations of both the 'India' and 'USA' classes if we don't have them already. With those, we can call their action using the same func() function:

```
def func(obj):
  obj.capital()
  obj.language()
  obj.type()

obj_ind = India()
obj_usa = USA()

func(obj_ind)
func(obj_usa)
```

Code: Implementing polymorphism with a function

```
class India():
  def capital(self):
    print("New Delhi is the capital of India.")

  def language(self):
    print("Hindi is the most widely spoken language of India.")

  def type(self):
    print("India is a developing country.")

class USA():
  def capital(self):
    print("Washington, D.C. is the capital of USA.")

  def language(self):
    print("English is the primary language of USA.")

  def type(self):
    print("USA is a developed country.")

def func(obj):
  obj.capital()
  obj.language()
  obj.type()

obj_ind = India()
obj_usa = USA()

func(obj_ind)
func(obj_usa)
```

Output

```
New Delhi is the capital of India.
Hindi is the most widely spoken language of India.
India is a developing country.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
USA is a developed country.
```

Encapsulation

- It describes the idea of wrapping data and the methods that work on data within one unit.
- This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data.

- To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as **private variable**.
- A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.

Protected Members

- Protected members (in C++ and JAVA) are those members of the class that cannot be accessed outside the class but can be accessed from within the class and its subclasses.
- To accomplish this in Python, just follow **the convention** by prefixing the name of the member by a **single underscore** "_".

Note: The __init__ method is a constructor and runs as soon as an object of a class is instantiated.

```
# Creating a base class
class Base:
  def __init__(self):

    # Protected member
    self._a = 2

# Creating a derived class
class Derived(Base):
  def __init__(self):

    # Calling constructor of
    # Base class
    Base.__init__(self)
    print("Calling protected member of base class: ")
    print(self._a)

obj1 = Derived()

obj2 = Base()

# Calling protected member
# Outside class will  result in
# AttributeError
print(obj2.a)
```

Ouput

```
Calling protected member of base class:
2
```

```
Traceback (most recent call last):
  File "C:/Users/RITI/AppData/Local/Programs/Python/Python39/reve.py", line 25, in
<module>
    print(obj2.a)
AttributeError: 'Base' object has no attribute 'a'
```

Private Members

Private members are similar to protected members, the difference is that the class members declared private should neither be accessed outside the class nor by any base class. In Python, there is no existence of **Private** instance variables that cannot be accessed except inside a class. However, to define a private member prefix the member name with double underscore "__". **Note:** Python's private and protected member can be accessed outside the class through

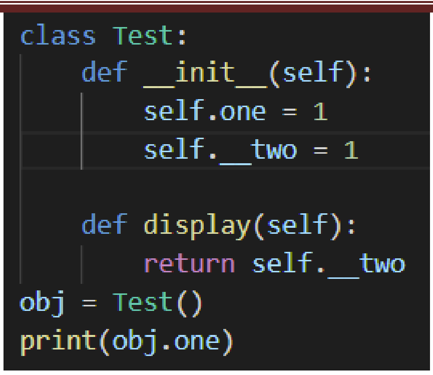
```
# Python program to
# demonstrate private members
# Creating a Base class
class Base:
    def __init__(self):
        self.a = "BennettUniversity"
        self.__c = "BennettUniversity"

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)
# Driver code
obj1 = Base()
print(obj1.a)
```

Output

```
BennettUniversity
```

Problems:

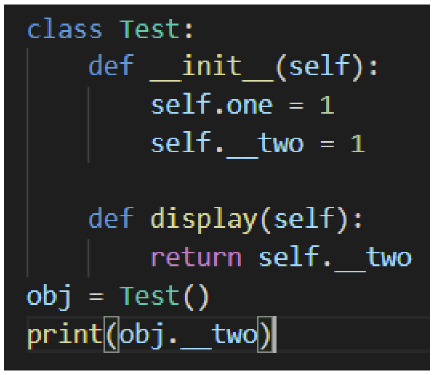
1. Identify the output of the following code:



```python
class Test:
    def __init__(self):
        self.one = 1
        self.__two = 1

    def display(self):
        return self.__two
obj = Test()
print(obj.one)
```

Ans: 1

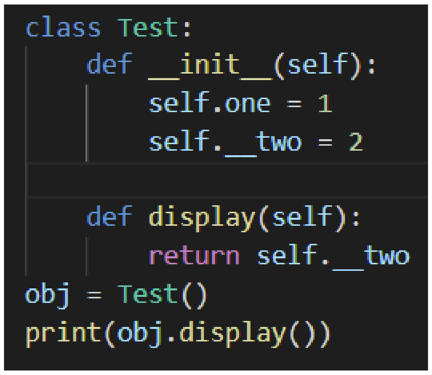
2. What will be the output of the following code:



```python
class Test:
    def __init__(self):
        self.one = 1
        self.__two = 1

    def display(self):
        return self.__two
obj = Test()
print(obj.__two)
```

Ans: The program has an error because b is private and hence can't be printed.

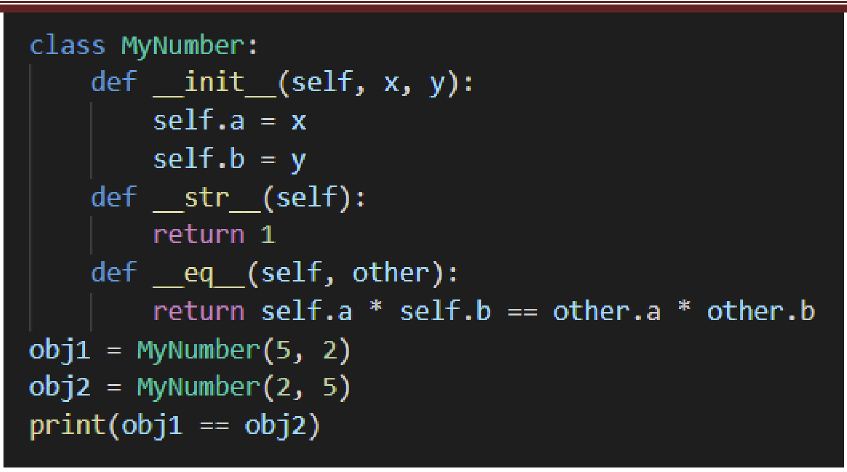
3. What will be the output of the following code:



```python
class Test:
    def __init__(self):
        self.one = 1
        self.__two = 2

    def display(self):
        return self.__two
obj = Test()
print(obj.display())
```

Ans: 2

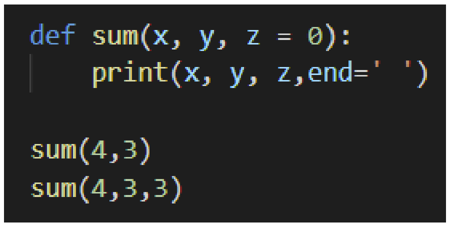
4. What will be the output of the following code:



```
class MyNumber:
    def __init__(self, x, y):
        self.a = x
        self.b = y
    def __str__(self):
        return 1
    def __eq__(self, other):
        return self.a * self.b == other.a * other.b
obj1 = MyNumber(5, 2)
obj2 = MyNumber(2, 5)
print(obj1 == obj2)
```

Ans: True

5. What will be the output of the following code:



```
def sum(x, y, z = 0):
    print(x, y, z,end=' ')

sum(4,3)
sum(4,3,3)
```

Ans: 4 3 0 4 3 3

6. Create a class that represents an Employee. The class should contain the following:
    a. Attributes for the Employee's name, hours worked and hourly rates.
        i. All attributes should be private to the class
    b. The class should contain an init method that accepts arguments for and updates all instance variables.
    c. The class should contain mutators and accessors for all instance variables.
    d. The class should contain a calc_pay method that calculates and returns the Employee's pay:
        i. An Employee's pay is equal to the number of hours worked times their hourly rate.

```
class Employee:
    def __init__(self,name,hours,rates):
        self.__name=name
        self.__hoursWorked=hours
        self.__hourlyRates=rates

    def setName(self,name):
        self.__name=name

    def getName(self):
        return self.__name

    def setHoursWorked(self,hours):
        self.__hoursWorked=hours

    def getHoursWorked(self):
        return self.__hoursWorked

    def setHourlyRates(self,rates):
        self.__hourlyRates=rates

    def getHourlyRates(self):
        return self.__hourlyRates

    def calculatePay(self):
        return self.__hourlyRates*self.__hoursWorked

E1=Employee("abc",20,200)
print("Pay is : ",E1.calculatePay())
```

7. Design a class ‘Complex ‘with data members for real and imaginary part. The class should have methods for performing arithmetic operations of two complex numbers. Overload + and – operator to add and subtract two complex numbers. Write a driver program to test your class.

```
class Complex:
    def __init__(self,r,i):
        self.real=r
        self.imaginary=i

    def __add__(self,c1):
        self.real=self.real+c1.real
        self.imaginary=self.imaginary+c1.imaginary
        return self
        # print(f"Result of addition is: {self.real} + ({self.imaginary})i")

    def __sub__(self,c1):
        self.real=self.real-c1.real
        self.imaginary=self.imaginary-c1.imaginary
        return self
        # print(f"Result of addition is: {self.real} +
({self.imaginary})i")

c1=Complex(2,3)
```

```
c2=Complex(3,4)
c3=Complex(1,2)

c4=c1+c2
print(f"Result of addition is: {c4.real} + ({c4.imaginary})i")

c5=c2-c1
print(f"Result of subtraction is: {c5.real} + ({c5.imaginary})i")
```

8. Create class Number with only one private instance variable as a float type. Include the following methods (include respective constructor) isZero( ), isPositive(), isNegative( ), isOdd( ), isEven( ), isPrime(), isArmstrong() the above methods return boolean primitive type.

```
import math
class Number:
    def __init__(self,n):
        self.__number=n

    def isZero(self):
        if self.__number==0:
            return True
        return False

    def isPositive(self):
        if self.__number>0:
            return True
        return False

    def isNegative(self):
        if self.__number<0:
            return True
        return False

    def isOdd(self):
        if self.__number%2==1:
            return True
        return False

    def isEven(self):
        if self.__number%2==0:
            return True
        return False

    def isPrime(self):
        n=self.__number
        prime_flag = 0
        if(n > 1):
            for i in range(2, int(math.sqrt(n)) + 1):
                if (n % i == 0):
                    prime_flag = 1
                    break
            if (prime_flag == 0):
                return True
            else:
                return False
        return False

    def isArmstrong(self):
        num = self.__number
        sum = 0
        temp = num
        while temp > 0:
            digit = temp % 10
            sum += digit ** 3
            temp //= 10
        if num == sum:
            return True
        return False

n=Number(11)
print("Is zero : ",n.isZero())
print("Is positive : ",n.isPositive())
print("Is negative : ",n.isNegative())
print("Is odd : ",n.isOdd())
print("Is even : ",n.isEven())
print("Is prime : ",n.isPrime())
print("Is armstrong : ",n.isArmstrong())
```

9. Write a class to represent 2 dimensional points. Your class should have all the members as private. After that write a driver function getDistance() which should compute the distance between two points. The function should have code to get the two points as input. You need to calculate distance and then print the result. Test your program using the following data: The distance between the points (3, 17) and (8, 10) is 8.6023... (lots more digits printed); the distance between (–33, 49) and (–9,–15) is 68.352.

```
import math
class Point:
    def __init__(self,x,y):
        self.__x=x
        self.__y=y

    def getX(self):
        return self.__x
    def getY(self):
        return self.__y

def getDistance(p1,p2):
    distance=math.sqrt((p2.getX()-p1.getX())**2 + (p2.getY()-p1.getY())**2)
    print("Distance is: ",distance)

p1=Point(10,20)
p2=Point(20,30)
getDistance(p1,p2)
```

10. Write a class “BankAccount” with the following private instance variables: accountNumber (an integer), balance (a floating-point number), and “ownerName” (a String). Write a constructors for this class. Constructor should take customer name and amount as arguments. If amount is not given then it should be zero. Also write methods getBalance (to print balance), add (to deposit an amount), and subtract (to withdraw an amount) and implement them. Write a driver code to test your class. Make sure that the balance should not be negative also negative amount to withdraw or deposit is not permitted.

```
import random

class BankAccount:

    def __init__(self,name,amount=0):
        self.__accountNumber=random.randint(1,100)
        self.__balance=amount
        self.__ownerName=name

    def getBalance(self):
        return self.__balance

    def add(self,amt):
        if amt<0:
            print("Negative amount is not permitted")
        else:
            self.__balance=self.__balance+amt

    def withdraw(self,amt):
        if amt<0:
            print("Negative amount is not permitted")
        else:
            if amt>self.__balance:
                print("Not enough balance")
            else:
                self.__balance=self.__balance-amt

ba=BankAccount("ABC")
print(ba.getBalance())
ba.add(100)
print(ba.getBalance())

ba1=BankAccount("CDE",100)
ba1.withdraw(101)
print(ba1.getBalance())
```

11. Write a class, Commission, which has an private instance variable, sales; an appropriate constructor; and a method, commission() that returns the commission. Now write a driver code to test the Commission class by reading a sale from the user, using it to create a Commission object after validating that the value is not negative. Finally, call the commission() method to get and print the commission. If the sales are negative, your demo should print the message "Invalid Input".

```
class Commission:
    def __init__(self,sales):
        self.__sales=sales

    def printComission(self):
        print("Commision is: ",self.__sales)

sales=int(input("Enter the sale: "))
if sales<0:
    print("Invalid input")
else:
    comission=Commission(sales)
    comission.printComission()
```

12. Create a class Voter(voterId, name, age). The message should be printed if age is less than 18 and the object should not be created. The message is "invalid age for voter ". Besides the above mentioned three members, there should be another private member named "vote". Now write driver code in which you need to conduct an election and take votes from five voters. Count the votes and declare the winner.

```
import random

class Voter:
    def __init__(self,voterId, name, age):
        if age<18:
            print("Invalid age for voter")
            raise ValueError
        self.voterId=voterId
        self.name=name
        self.age=age
        self.__vote=""

    def setVote(self,vote):
        self.__vote=vote

    def getVote(self):
        return self.__vote

voters=[]
for i in range(0,5):
    try:
        vTemp=Voter(random.randint(1,10),"A",random.randint(10,50))
        voters.append(vTemp)
    except:
        print("Invalid age of voter")

for voter in voters:
    voter.setVote(random.randint(1,2))

for voter in voters:
    firstCandidate=0
    secondCandidate=0
    if voter.getVote()==1:
        firstCandidate+=1
    elif voter.getVote()==2:
        secondCandidate+=1

if firstCandidate>secondCandidate:
    print("First candiate won")
elif secondCandidate>firstCandidate:
    print("Second candidate won")
else:
    print("There is a tie")
```

13. Create a Point class which stores x and y coordinates. Overload the + and comparison operators to implement the adding and comparison functionality for Point class.

```
class Point:
    def __init__(self,x,y):
        self.x=x
        self.y=y

    def __add__(self,other):
        x = self.x + other.x
        y = self.y + other.y
        return x,y
    def __eq__(self, other):
        if self.x==other.x and self.y==other.y:
            return "Both the points are equal"
        else:
            return "Both the points are not equal"

p1 = Point(2,3)
p2 = Point(-1,2)
print(p1 + p2)

p3=Point(5,5)
p4=Point(5,6)
print (p3==p4)
```