OOP(Class, Object, Members, inheritance, polymorphism)

1. Predict the output:

```
class IOString():
    def __init__(self):
        self.str1 = ""

    def get_String(self):
        self.str1 = input()

    def print_String(self):
        print(self.str1.upper())

str1 = IOString()
str1.get_String()
str1.print_String()
```

1. Output:

2. Predict the output:

```
class py_solution:
    def rwords(self, s):
        return ' '.join(reversed(s.split()))

pt = py_solution()
print(pt.rwords('hello .py'))
```

2. Output:

3. Predict the output:

```
class py_solution:
   def press(self, x, n):
        if x==0 or x==1 or n==1:
            return x
        if x==-1:
            if n%2 ==0:
                return 1
            else:
                return -1
        if n==0:
            return 1
        if n<0:
            return 1/self.press(x,-n)
        val = self.press(x,n//2)
        if n%2 ==0:
            return val*val
        return val*val*x
print(py_solution().press(2, 3))
```

3. Output:

4. Predict the output:

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('Hello, my name is', self.name)
p = Person('Subham')
p.say_hi()

class Test:
    def fun(self):
        print("Hello")
obj = Test()
obj.fun()
```

4. Output:

5. Predict the output:

```
class CSStudent:
    stream = 'cse'
    def __init__(self, roll):
        self.roll = roll
    def setAddress(self, address):
        self.address = address
    def getAddress(self):
        return self.address

a = CSStudent(101)
a.setAddress("Noida, UP")
print(a.getAddress())
```

5. Output

6. Predict the output:

```
class MyClass:
    hiddenVariable = 10

myObject = MyClass()
print(myObject.hiddenVariable)

class MyClass:
    __hiddenVariable = 0

    def add(self, increment):
        self.__hiddenVariable += increment
        print (self.__hiddenVariable)

myObject = MyClass()
myObject.add(2)
myObject.add(5)
```

6. Output

7. Predict the output: (Inheritance)

```
class Person(object):

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

    def getName(self):
        return self.name

    def isEmployee(self):
        return False

class Employee(Person):

    def isEmployee(self):
        return True

emp = Person("student1")
print(emp.getName(), emp.isEmployee())

emp = Employee("student2")
print(emp.getName(), emp.isEmployee())
```

7. Output:

8. Predict the output:

How to check if a class is subclass of another.

```
class Base(object):
    pass    # Empty Class

class Derived(Base):
    pass

print(issubclass(Derived, Base))
print(issubclass(Base, Derived))

d = Derived()
b = Base()

# check is b is an instance of Derived ?
print(isinstance(b, Derived))

# check d is an instance of Base ?
print(isinstance(d, Base))
```

8. Output:

9. Predict the output:

Python example to show working of multiple inheritances.

```
class Base1(object):
    def __init__(self):
        self.str1 = "student1"
        print ("Base1")

class Base2(object):
    def __init__(self):
        self.str2 = "student2"
        print ("Base2")

class Derived(Base1, Base2):
    def __init__(self):
```

```
        # Calling constructors of Base1
        # and Base2 classes
        Base1.__init__(self)
        Base2.__init__(self)
        print ("Derived")

    def printStrs(self):
        print(self.str1, self.str2)

ob = Derived()
ob.printStrs()
```

9. Output:

10. Predict the output:

How to access parent members in a subclass.

```
class Base(object):

    def __init__(self, x):
        self.x = x

class Derived(Base):

    def __init__(self, x, y):
        Base.x = x
        self.y = y

    def printXY(self):

       print(Base.x, self.y)

d = Derived(10, 20)
d.printXY()
```

10. Output:

11. Predict the output:

```
class Solution:
   def solve(self, words):
      maxlength = 0
      curr_letter, curr_length = None, 0
      for word in words:
         if not curr_letter or curr_letter != word[0]:
            maxlength = max(maxlength, curr_length)
            curr_letter, curr_length = word[0], 1
         else:
            curr_length += 1
      return max(maxlength, curr_length)
ob = Solution()
words = ["small", "she", "scorn", "on", "the", "shore"]
print(ob.solve(words))
```

11. Output:

12. Predict the output:

```
class Rectangle():
    def __init__(self, l, w):
        self.length = l
        self.width  = w

    def rectangle_area(self):
        return self.length*self.width

newRectangle = Rectangle(12, 10)
print(newRectangle.rectangle_area())
```

12. Output:

13. Predict the output: (*Inherited or Subclass*)

```
class Person(object):
    def __init__(self, name):
        self.name = name

    def getName(self):
        return self.name

    def isEmployee(self):
        return False

class Employee(Person):
    def __init__(self, name, eid):

        super(Employee, self).__init__(name)
        self.empID = eid

    def isEmployee(self):
        return True

    def getID(self):
        return self.empID

emp = Employee("student", "E101")
print(emp.getName(), emp.isEmployee(), emp.getID())
```

13. Output:

14. Predict the output:

```
class Solution:
  def solve(self, nums):
    e=[i for i in nums if i%2==0]
    return (len(nums)-len(e))*len(e)
nums = [5, 4, 6]
ob = Solution()
print(ob.solve(nums))
```

14. Output

15. Predict the output:

```
class Solution:
   def solve(self, nums):
      s = sorted(nums)
      count = 0
      for i in range(len(nums)):
         if s[i] == nums[i]:
            count += 1
      return count
ob = Solution()
print(ob.solve([2, 8, 4, 5, 11]))
```

15. Output

16. Predict the output:

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('Hello, my name is', self.name)
p = Person('Shwetanshu')
p.say_hi()


class Test:
    def fun(self):
        print("Hello")
obj = Test()
obj.fun()
```

16. Output:

17: Predict the output:(*Python program to demonstrate protected members)*

```
class Base:
    def __init__(self):

        # Protected member
        self._a = 2

class Derived(Base):
    def __init__(self):

        Base.__init__(self)
        print("Calling protected member of base class: ")
        print(self._a)
obj1 = Derived()
obj2 = Base()
print(obj2.a)
```

18. Predict the output:

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

    def language(self):
        print("Hindi is 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()
```

18. Output:

19. Predict the output:

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

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

    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)
```

19. Output:

20. Write a program for following:

Write a Python class to find validity of a string of parentheses, '(', ')', '{', '}', '[' and ']. These brackets must be close in the correct order, for example "()" and "()[]{}" are valid, but "[)", "({[)]" and "{{{" are invalid.

20. Solution: