

APP WEEK-3 LAB

COMPULSARY QUESTION:

Develop a set of classes for a college to use in various student service and personnel applications. Classes you need to design include the following:

- **Person**—A Person contains a first name, last name, street address, zip code, and phone number. The class also includes a method that sets each data field, using a series of dialog boxes and a display method that displays all of a Person's information on a single line at the

command line on the screen.

- **CollegeEmployee**—CollegeEmployee descends from Person. A CollegeEmployee also includes a Social Security number, an annual salary, and a department name, as well as methods that override the Person methods to accept and display all CollegeEmployee data.

- **Faculty**—Faculty descends from CollegeEmployee. This class also includes a Boolean field

that indicates whether the faculty member is tenured, as well as methods that override the CollegeEmployee methods to accept and display this additional piece of information.

- **Student**—Student descends from Person. In addition to the fields available in Person, a Student contains a major field of study and a grade point average as well as methods that override the Person methods to accept and display these additional facts.

Write an application named CollegeList that declares an array of four "regular"

CollegeEmployees, three Faculty, and seven Students. Prompt the user to specify which type

of person's data will be entered (C, F, or S), or allow the user to quit (Q). While the user chooses to continue (that is, does not quit), accept data entry for the appropriate type of Person. If the user attempts to enter data for more than four CollegeEmployees, three Faculty,

or seven Students, display an error message. When the user quits, display a report on the screen listing each group of Persons under the appropriate heading of "College Employees,"

"Faculty," or "Students." If the user has not entered data for one or more types of Persons

during a session, display an appropriate message under the appropriate heading

Code:

class Person:

```
def __init__(self, first_name, last_name, street_address, zip_code, phone_number):  
    self.first_name = first_name  
    self.last_name = last_name  
    self.street_address = street_address  
    self.zip_code = zip_code  
    self.phone_number = phone_number
```

```
def set_data(self):
```

```
    self.first_name = input("Enter first name: ")  
    self.last_name = input("Enter last name: ")  
    self.street_address = input("Enter street address: ")  
    self.zip_code = input("Enter zip code: ")  
    self.phone_number = input("Enter phone number: ")
```

```

def display(self):
    print("Name: ", self.first_name, self.last_name)
    print("Address: ", self.street_address)
    print("Zip Code: ", self.zip_code)
    print("Phone Number: ", self.phone_number)

class CollegeEmployee(Person):
    def __init__(self, social_security_number, annual_salary, department_name, first_name, last_name, street_address, zip_code, phone_number):
        Person.__init__(self, first_name, last_name, street_address, zip_code, phone_number)
        self.social_security_number = social_security_number
        self.annual_salary = annual_salary
        self.department_name = department_name

    def set_data(self):
        Person.set_data(self)
        self.social_security_number = input("Enter social security number: ")
        self.annual_salary = input("Enter annual salary: ")
        self.department_name = input("Enter department name: ")

    def display(self):
        Person.display(self)
        print("Social Security Number: ", self.social_security_number)
        print("Annual Salary: ", self.annual_salary)
        print("Department Name: ", self.department_name)

class Faculty(CollegeEmployee):
    def __init__(self, tenured, social_security_number, annual_salary, department_name, first_name, last_name, street_address, zip_code, phone_number):
        CollegeEmployee.__init__(self, social_security_number, annual_salary, department_name, first_name, last_name, street_address, zip_code, phone_number)
        self.tenured = tenured

    def set_data(self):
        CollegeEmployee.set_data(self)
        self.tenured = input("Is the faculty member tenured (True or False): ")

    def display(self):
        CollegeEmployee.display(self)
        print("Tenured: ", self.tenured)

class Student(Person):
    def __init__(self, major, gpa, first_name, last_name, street_address, zip_code, phone_number):
        Person.__init__(self, first_name, last_name, street_address, zip_code, phone_number)
        self.major = major
        self.gpa = gpa

    def set_data(self):
        Person.set_data(self)
        self.major = input("Enter major: ")
        self.gpa = input("Enter GPA: ")

    def display(self):
        Person.display(self)
        print("Major: ", self.major)

```

```

print("GPA: ", self.gpa)

class CollegeList:
    def __init__(self):
        self.college_employees = []
        self.faculty = []
        self.students = []

    def add_person(self, person):
        if isinstance(person, CollegeEmployee):
            if len(self.college_employees) < 4:
                self.college_employees.append(person)
            else:
                print("Error: Cannot add more than 4 College Employees.")
        elif isinstance(person, Faculty):
            if len(self.faculty) < 3:
                self.faculty.append(person)
            else:
                print("Error: Cannot add more than 3 Faculty members.")
        elif isinstance(person, Student):
            if len(self.students) < 7:
                self.students.append(person)
            else:
                print("Error: Cannot add more than 7 Students.")
        else:
            print("Error: Invalid person type.")

    def display_report(self):
        if self.college_employees:
            print("\nCollege Employees:")
            for ce in self.college_employees:
                ce.display()
        else:
            print("\nNo College Employees entered.")

        if self.faculty:
            print("\nFaculty:")
            for f in self.faculty:
                f.display()
        else:
            print("\nNo Faculty members entered.")

        if self.students:
            print("\nStudents:")
            for s in self.students:
                s.display()
        else:
            print("\nNo Students entered.")

if __name__ == '__main__':
    college_list = CollegeList()
    choice = "

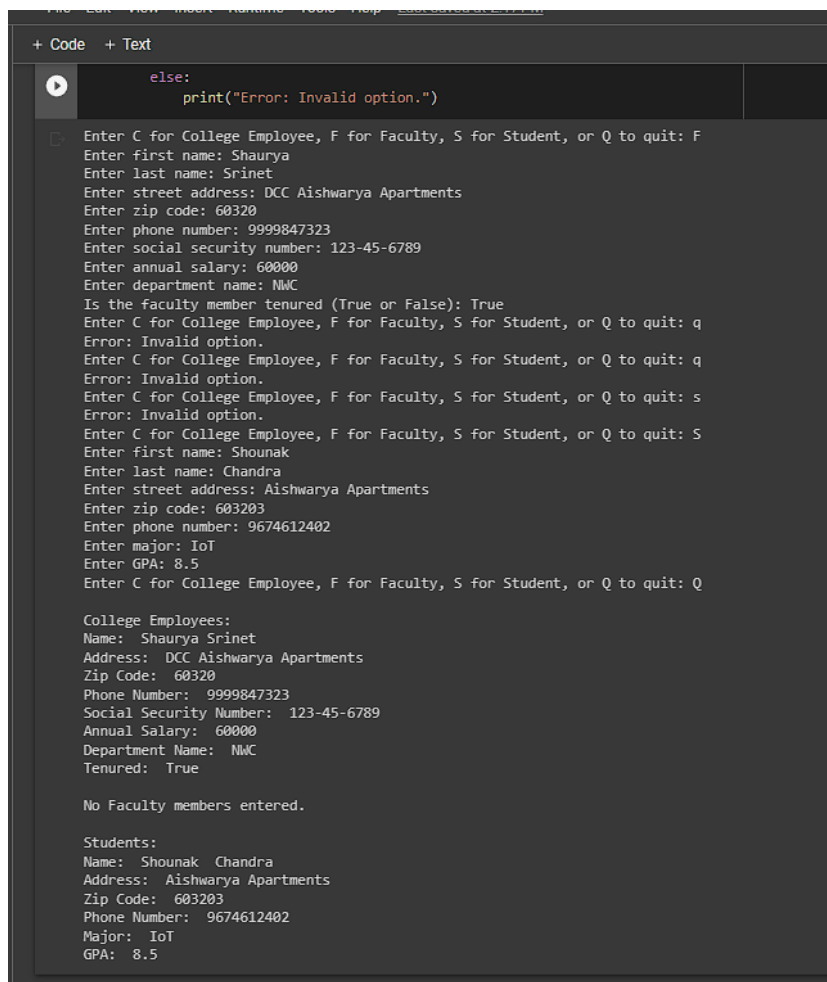
```

```

while choice != 'Q':
    choice = input("Enter C for College Employee, F for Faculty, S for Student, or Q to quit: ")
)
    if choice == 'C':
        ce = CollegeEmployee(social_security_number='123-45-6789', annual_salary=60000, department_name='Computer Science', first_name='Jane', last_name='Doe', street_address='456 Elm St', zip_code='67890', phone_number='555-555-1212')
        ce.set_data()
        college_list.add_person(ce)
    elif choice == 'F':
        f = Faculty(tenured=True, social_security_number='123-45-6789', annual_salary=60000, department_name='Computer Science', first_name='Jane', last_name='Doe', street_address='456 Elm St', zip_code='67890', phone_number='555-555-1212')
        f.set_data()
        college_list.add_person(f)
    elif choice == 'S':
        s = Student(major='Computer Science', gpa=3.5, first_name='John', last_name='Doe', street_address='123 Main St', zip_code='12345', phone_number='555-555-1212')
        s.set_data()
        college_list.add_person(s)
    elif choice == 'Q':
        college_list.display_report()
    else:
        print("Error: Invalid option.")

```

SnapShot:



```

+ Code + Text
else:
    print("Error: Invalid option.")

Enter C for College Employee, F for Faculty, S for Student, or Q to quit: F
Enter first name: Shaurya
Enter last name: Srinet
Enter street address: DCC Aishwarya Apartments
Enter zip code: 60320
Enter phone number: 9999847323
Enter social security number: 123-45-6789
Enter annual salary: 60000
Enter department name: NWC
Is the faculty member tenured (True or False): True
Enter C for College Employee, F for Faculty, S for Student, or Q to quit: q
Error: Invalid option.
Enter C for College Employee, F for Faculty, S for Student, or Q to quit: q
Error: Invalid option.
Enter C for College Employee, F for Faculty, S for Student, or Q to quit: s
Error: Invalid option.
Enter C for College Employee, F for Faculty, S for Student, or Q to quit: S
Enter first name: Shounak
Enter last name: Chandra
Enter street address: Aishwarya Apartments
Enter zip code: 603203
Enter phone number: 9674612402
Enter major: IoT
Enter GPA: 8.5
Enter C for College Employee, F for Faculty, S for Student, or Q to quit: Q

College Employees:
Name: Shaurya Srinet
Address: DCC Aishwarya Apartments
Zip Code: 60320
Phone Number: 9999847323
Social Security Number: 123-45-6789
Annual Salary: 60000
Department Name: NWC
Tenured: True

No Faculty members entered.

Students:
Name: Shounak Chandra
Address: Aishwarya Apartments
Zip Code: 603203
Phone Number: 9674612402
Major: IoT
GPA: 8.5

```

Q1.

Develop an Python code to display the following output using class and object (only one class and two objects)

Blu is a bird

Woo is also a bird

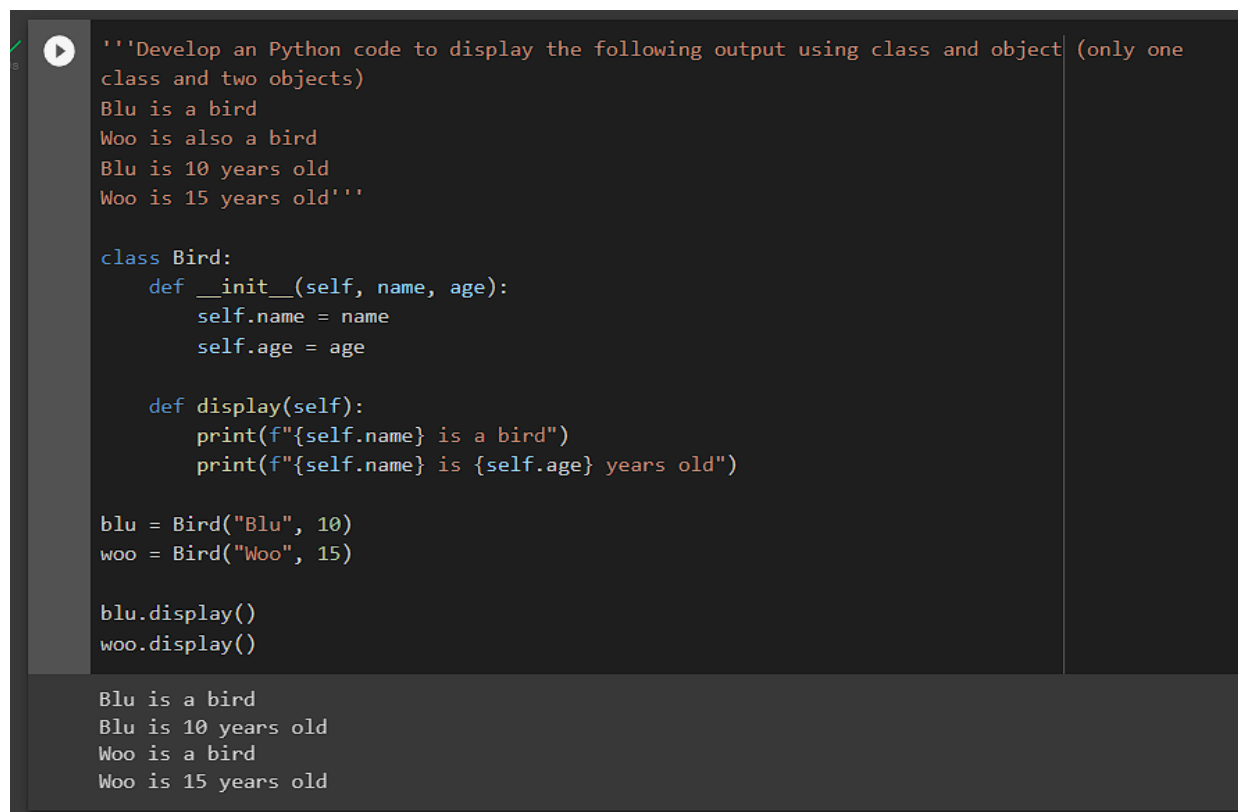
Blu is 10 years old

Woo is 15 years old

Code:

```
class Bird:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print(f"{self.name} is a bird")
        print(f"{self.name} is {self.age} years old")
blu = Bird("Blu", 10)
woo = Bird("Woo", 15)
blu.display()
woo.display()
```

SnapShot:

A screenshot of a code editor with a dark background. The editor shows a Python class named 'Bird' with two methods: '__init__' and 'display'. The '__init__' method takes 'name' and 'age' as arguments and assigns them to 'self.name' and 'self.age'. The 'display' method prints two lines: the first line says '{self.name} is a bird' and the second line says '{self.name} is {self.age} years old'. Below the class definition, two objects are created: 'blu' with name 'Blu' and age 10, and 'woo' with name 'Woo' and age 15. Then, the 'display' method is called on both objects. The output of the code is shown at the bottom of the editor, displaying the expected output: 'Blu is a bird', 'Blu is 10 years old', 'Woo is a bird', and 'Woo is 15 years old'.

```
'''Develop an Python code to display the following output using class and object (only one
class and two objects)
Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old'''

class Bird:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print(f"{self.name} is a bird")
        print(f"{self.name} is {self.age} years old")

blu = Bird("Blu", 10)
woo = Bird("Woo", 15)

blu.display()
woo.display()
```

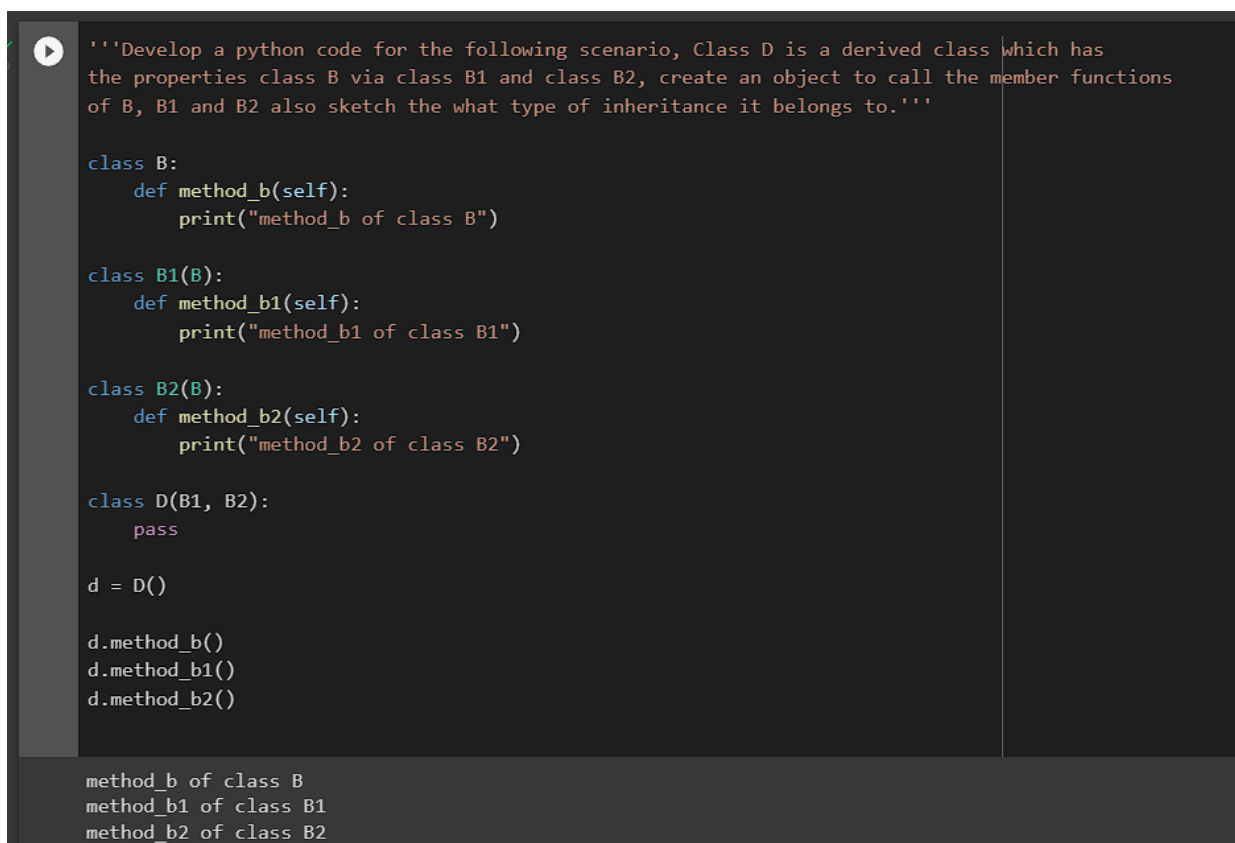
Blu is a bird
Blu is 10 years old
Woo is a bird
Woo is 15 years old

Q2. Develop a python code for the following scenario, Class D is a derived class which has the properties class B via class B1 and class B2, create an object to call the member functions of B, B1 and B2 also sketch the what type of inheritance it belongs to.

Code:

```
class B:
    def method_b(self):
        print("method_b of class B")
class B1(B):
    def method_b1(self):
        print("method_b1 of class B1")
class B2(B):
    def method_b2(self):
        print("method_b2 of class B2")
class D(B1, B2):
    pass
d = D()
d.method_b()
d.method_b1()
d.method_b2()
```

SnapShot:



The screenshot shows a Python IDE with a dark theme. On the left, there is a play button icon. The main editor area contains the following Python code:

```
'''Develop a python code for the following scenario, Class D is a derived class which has
the properties class B via class B1 and class B2, create an object to call the member functions
of B, B1 and B2 also sketch the what type of inheritance it belongs to.'''

class B:
    def method_b(self):
        print("method_b of class B")

class B1(B):
    def method_b1(self):
        print("method_b1 of class B1")

class B2(B):
    def method_b2(self):
        print("method_b2 of class B2")

class D(B1, B2):
    pass

d = D()

d.method_b()
d.method_b1()
d.method_b2()
```

Below the code, the output of the program is displayed in a separate window:

```
method_b of class B
method_b1 of class B1
method_b2 of class B2
```

Q3.

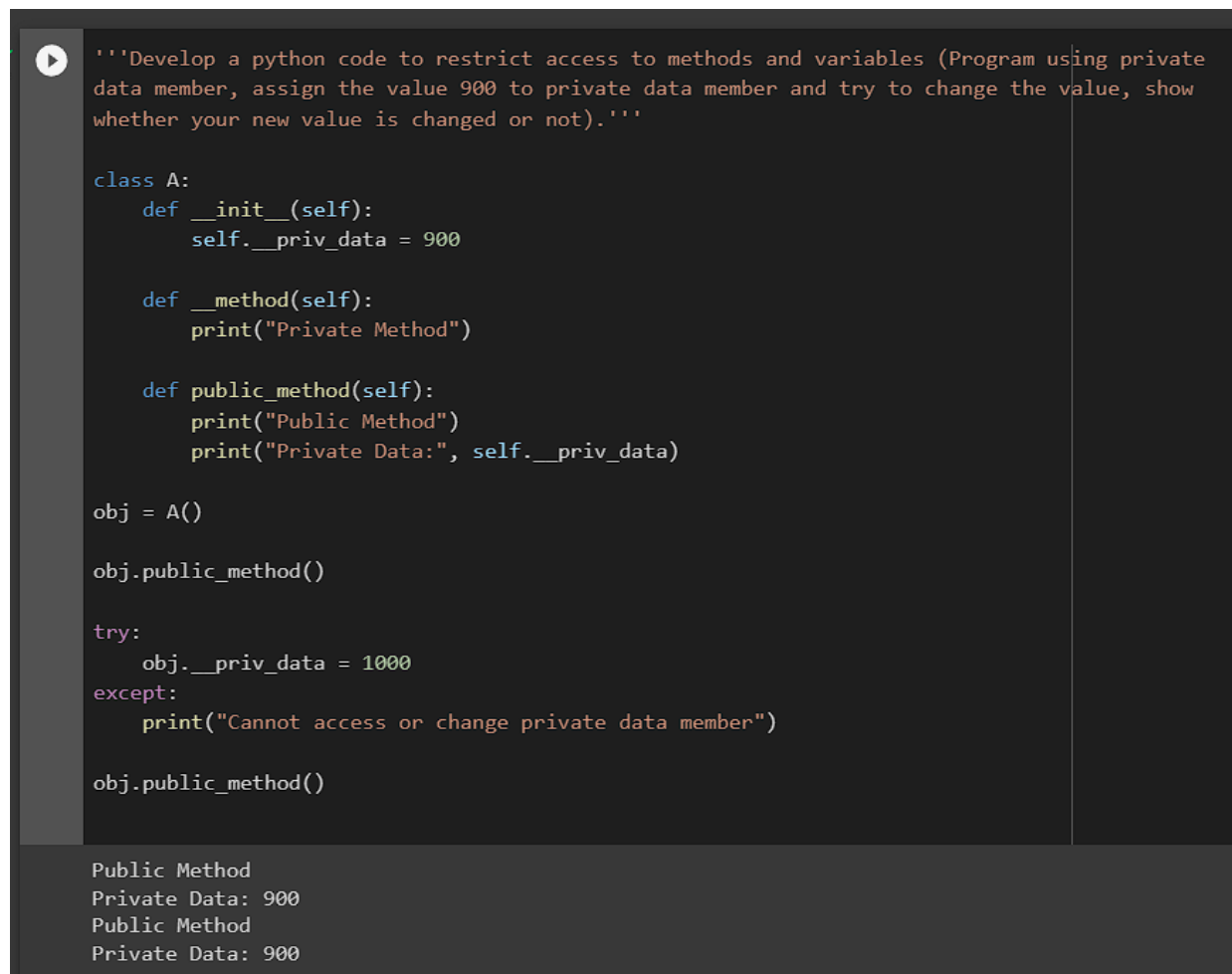
Develop a python code to restrict access to methods and variables (Program using private data member, assign the value 900 to private data member and try to change the value, show

whether your new value is changed or not).

Code:

```
class A:
    def __init__(self):
        self.__priv_data = 900
    def __method(self):
        print("Private Method")
    def public_method(self):
        print("Public Method")
        print("Private Data:", self.__priv_data)
obj = A()
obj.public_method()
try:
    obj.__priv_data = 1000
except:
    print("Cannot access or change private data member")
obj.public_method()
```

SnapShot:



The screenshot shows a Python IDE with a dark theme. The editor window contains the same Python code as in the previous block. Below the editor, the output console displays the results of the program execution. The output shows the public method being called twice, each time printing the public method name and the private data value (900). The attempt to change the private data value is caught by an exception, and the message 'Cannot access or change private data member' is printed.

```
'''Develop a python code to restrict access to methods and variables (Program using private
data member, assign the value 900 to private data member and try to change the value, show
whether your new value is changed or not).'''

class A:
    def __init__(self):
        self.__priv_data = 900

    def __method(self):
        print("Private Method")

    def public_method(self):
        print("Public Method")
        print("Private Data:", self.__priv_data)

obj = A()

obj.public_method()

try:
    obj.__priv_data = 1000
except:
    print("Cannot access or change private data member")

obj.public_method()
```

Public Method
Private Data: 900
Public Method
Private Data: 900

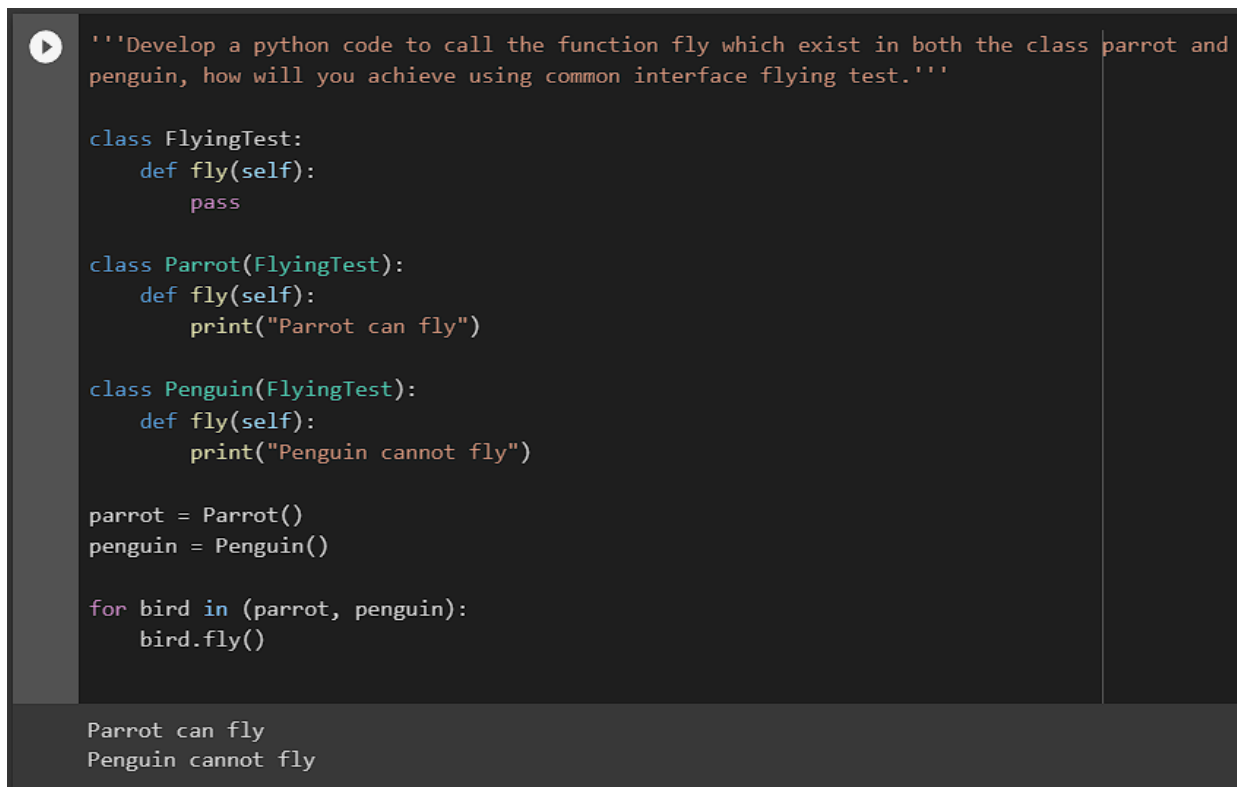
Q4.

Develop a python code to call the function fly which exist in both the class parrot and penguin, how will you achieve using common interface flying test.

Code:

```
class FlyingTest:
    def fly(self):
        pass
class Parrot(FlyingTest):
    def fly(self):
        print("Parrot can fly")
class Penguin(FlyingTest):
    def fly(self):
        print("Penguin cannot fly")
parrot = Parrot()
penguin = Penguin()
for bird in (parrot, penguin):
    bird.fly()
```

SnapShot:



The screenshot shows a code editor with a dark background. On the left, there is a play button icon. The code is written in a light-colored font. The code defines a base class `FlyingTest` with a `fly` method that does nothing. It then defines two subclasses, `Parrot` and `Penguin`, both inheriting from `FlyingTest`. `Parrot` overrides the `fly` method to print "Parrot can fly", and `Penguin` overrides it to print "Penguin cannot fly". The code then creates instances of both classes and iterates over them, calling the `fly` method on each. The output of the code is shown at the bottom of the editor, displaying "Parrot can fly" and "Penguin cannot fly" on separate lines.

```
'''Develop a python code to call the function fly which exist in both the class parrot and penguin, how will you achieve using common interface flying test.'''

class FlyingTest:
    def fly(self):
        pass

class Parrot(FlyingTest):
    def fly(self):
        print("Parrot can fly")

class Penguin(FlyingTest):
    def fly(self):
        print("Penguin cannot fly")

parrot = Parrot()
penguin = Penguin()

for bird in (parrot, penguin):
    bird.fly()
```

Parrot can fly
Penguin cannot fly

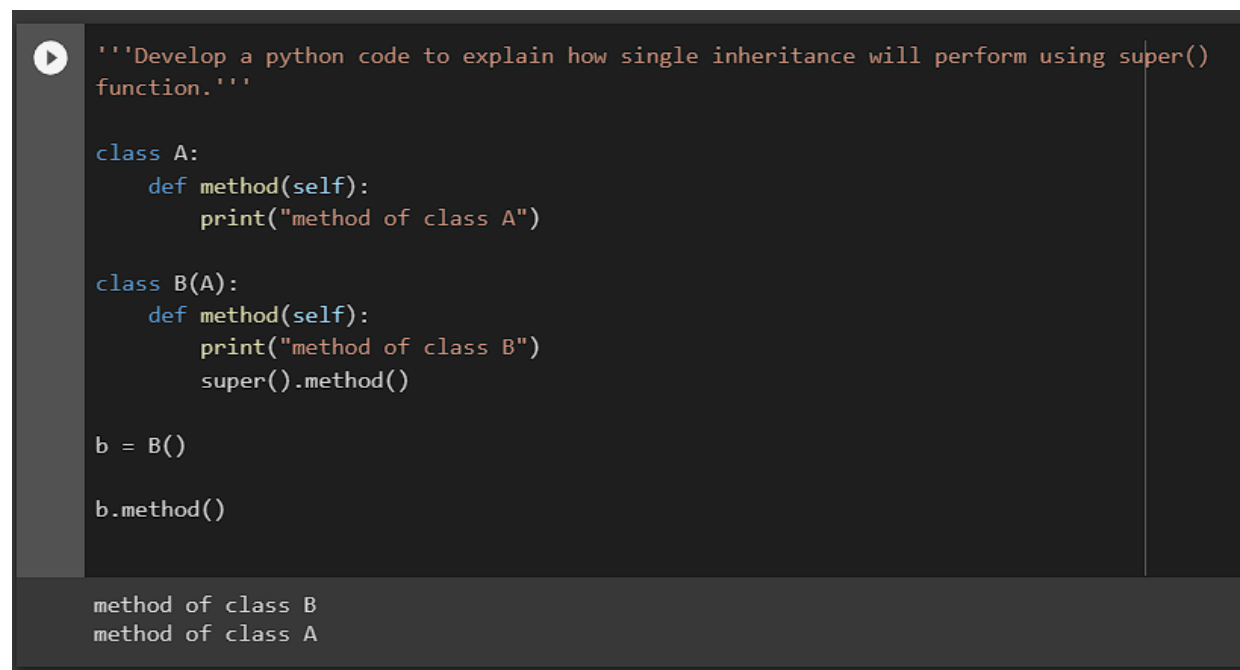
Q5.

Develop a python code to explain how single inheritance will perform using super() function.

Code:

```
class A:
    def method(self):
        print("method of class A")
class B(A):
    def method(self):
        print("method of class B")
        super().method()
b = B()
b.method()
```

SnapShot:



```
'''Develop a python code to explain how single inheritance will perform using super()
function.'''

class A:
    def method(self):
        print("method of class A")

class B(A):
    def method(self):
        print("method of class B")
        super().method()

b = B()

b.method()
```

method of class B
method of class A