Object Oriented Programming using Python (I)

Lecture(10)

Data hiding

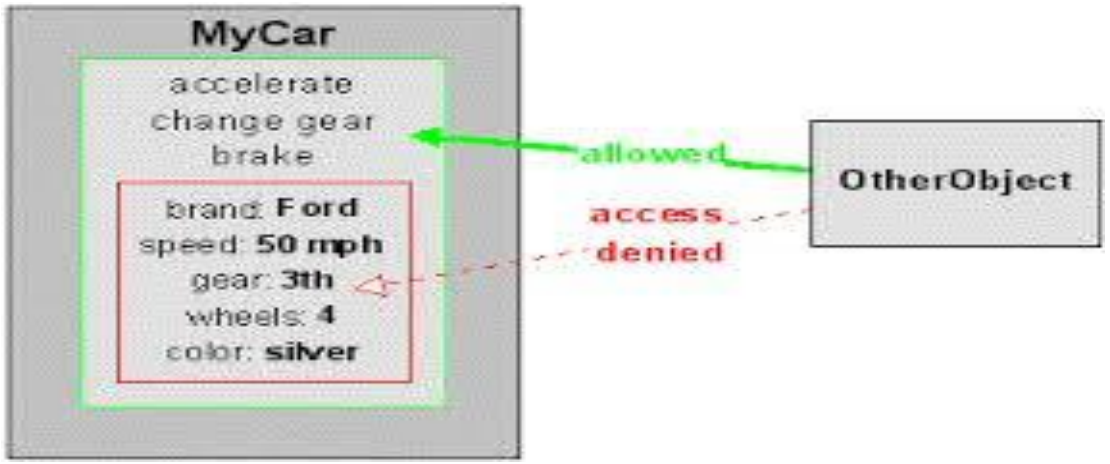
Prepared by. Ahmed Eskander Mezher

University of Information Technology and Communications

College of Business Informatics

Data Hiding

- **Data hiding** is a technique specifically used in object-oriented programming (OOP) to **hide** internal object details (**data** members). **Data hiding** ensures exclusive **data** access to class members and protects object by preventing unintended or intended changes.
- Data hiding isolates the end users from knowledge the internal design of an object.



Python Data Hiding

Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type.

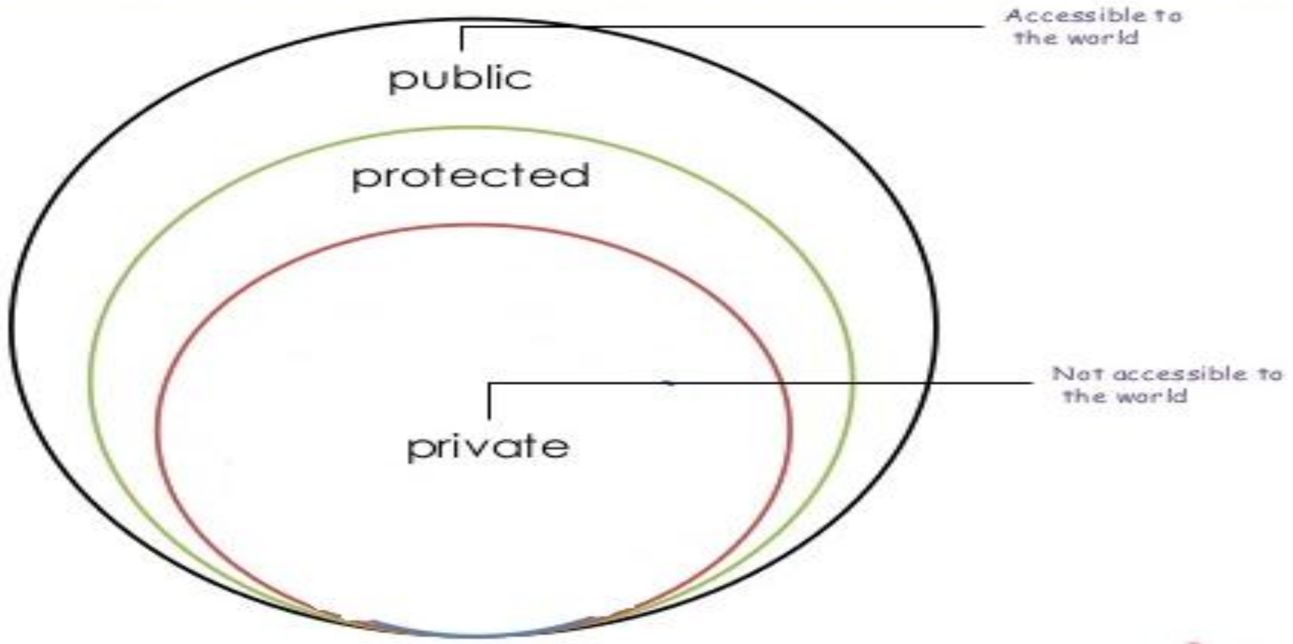
By default all members of a class can be accessed outside of class.

You can prevent this by making class members **private** or **protected**.

In Python, we use double underscore (__) before the attributes name to make those attributes **private**.

We can use single underscore (_) before the attributes name to make those attributes **protected**.

Position of Access Rights

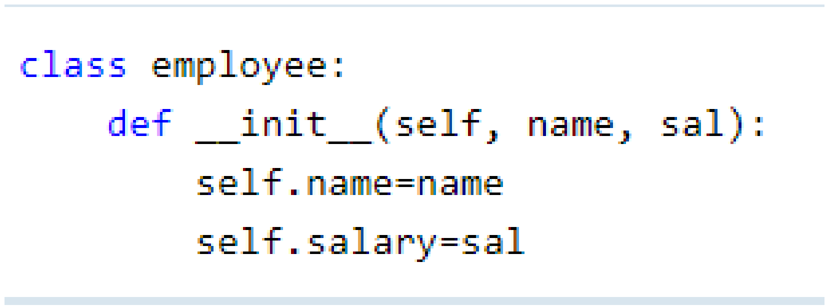


The following table shows the different behavior Public, Protected and Private Data

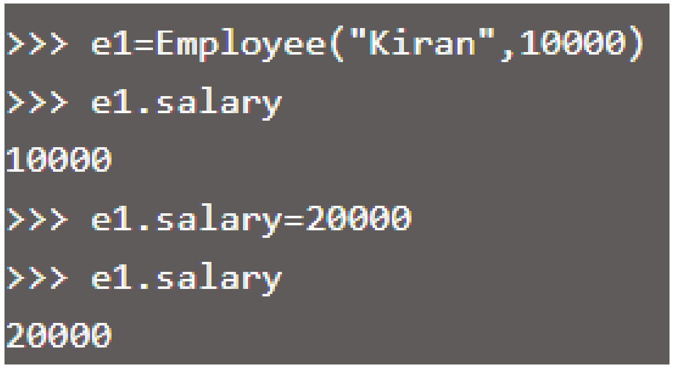
Name	Notation	Behavior
name	Public	Can be accessed from inside and outside
_name	Protected	Like a public member, but they shouldn't be directly accessed from outside
__name	Private	Can't be seen and accessed from outside

Python - **public**, protected and private

All members in a Python class are **public** by default. Any member can be accessed from outside the class environment.



```
class employee:
    def __init__(self, name, sal):
        self.name=name
        self.salary=sal
```



```
>>> e1=Employee("Kiran",10000)
>>> e1.salary
10000
>>> e1.salary=20000
>>> e1.salary
20000
```

Python - public, **protected** and private

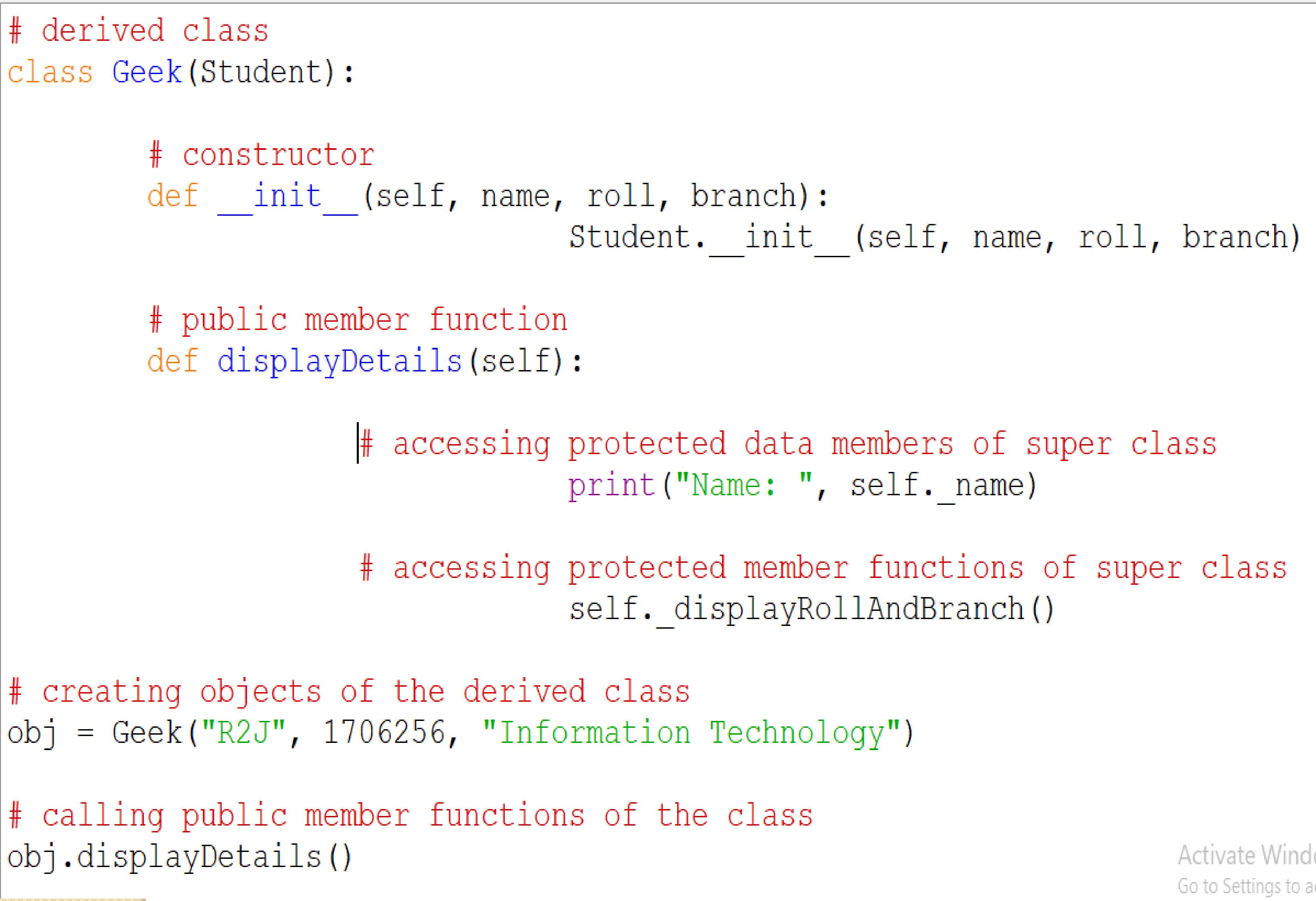
Python's principle to make an instance variable **protected** is to add a prefix _ (single underscore) to it. This effectively prevents it to be accessed, unless it is from within a sub-class.

```
# program to illustrate protected access modifier in a class
# super class
class Student:
        # protected data members
        _name = None
        _roll = None
        _branch = None

        # constructor
        def __init__(self, name, roll, branch):
                self._name = name
                self._roll = roll
                self._branch = branch

        # protected member function
        def _displayRollAndBranch(self):

                # accessing protected data members
                print("Roll: ", self._roll)
                print("Branch: ", self._branch)
```



```
# derived class
class Geek(Student):

        # constructor
        def __init__(self, name, roll, branch):
                                Student.__init__(self, name, roll, branch)

        # public member function
        def displayDetails(self):

                    # accessing protected data members of super class
                                print("Name: ", self._name)

                    # accessing protected member functions of super class
                                self._displayRollAndBranch()

# creating objects of the derived class
obj = Geek("R2J", 1706256, "Information Technology")

# calling public member functions of the class
obj.displayDetails()
```

Python - public, protected and **private**

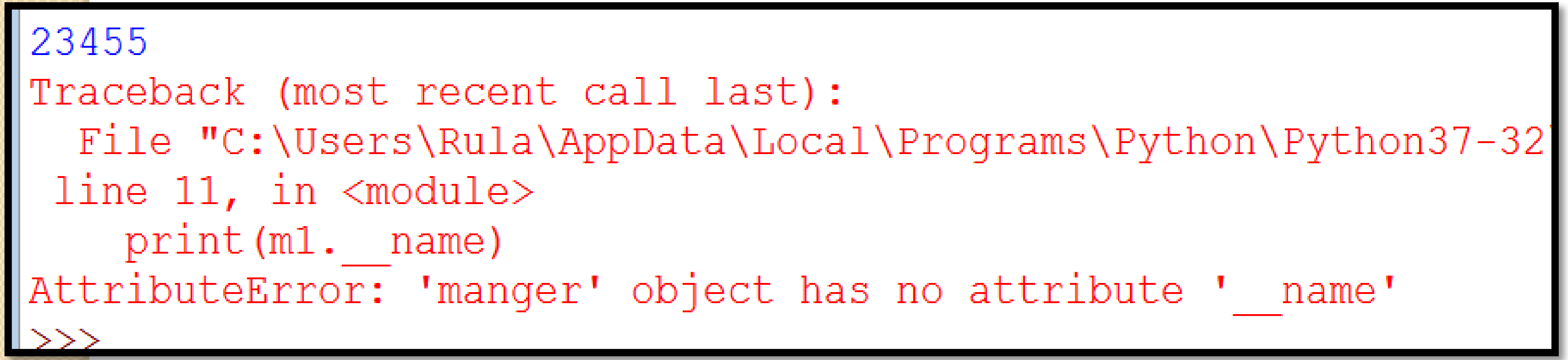
Similarly, a double underscore __ prefixed to a variable makes it **private**. It gives a strong suggestion not to touch it from outside the class. Any attempt to do so will result in an AttributeError:

```
class employee:
    def __init__(self, name, sal):
        self.__name=name  # private attribute
        self.__salary=sal # private attribute
```

```
>>> e1=employee("Bill",10000)
>>> e1.__salary
AttributeError: 'employee' object has no attribute '__salary'
```

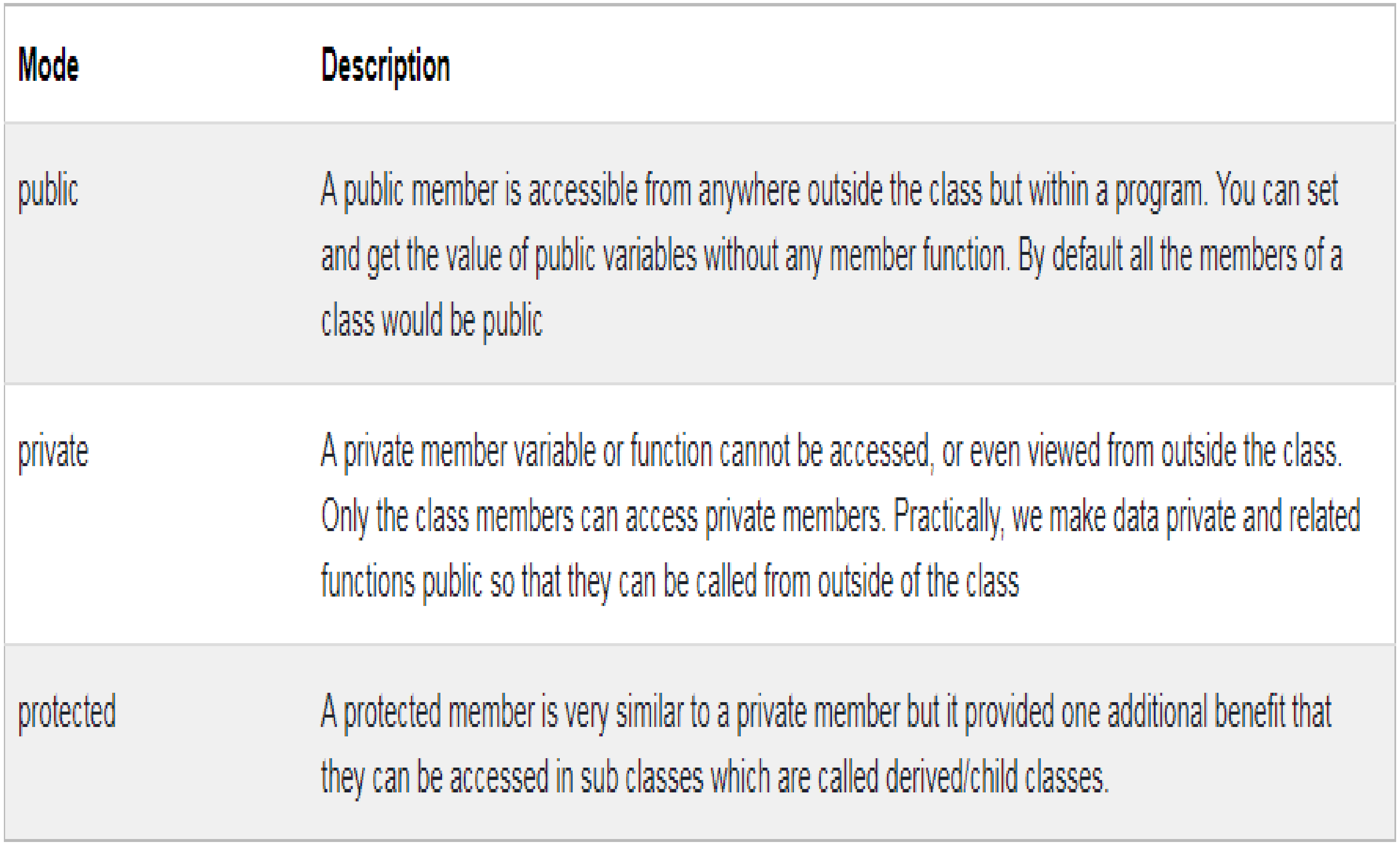
```
class employee:
    def __init__(self, name, sal):
        self.__name=name   # Private attributes
        self._salary=sal # protected attribute
class manger(employee):
     def __init__(self, name, sal,pos):
      employee.__init__(self, name, sal)

m1=manger("jack",23455,'chief manager')
print(m1._salary)
print(m1.__name)
```



```
23455
Traceback (most recent call last):
  File "C:\Users\Rula\AppData\Local\Programs\Python\Python37-32
 line 11, in <module>
    print(m1.__name)
AttributeError: 'manger' object has no attribute '__name'
>>>
```

Difference between public, private and protected



Mode	Description
public	A public member is accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function. By default all the members of a class would be public
private	A private member variable or function cannot be accessed, or even viewed from outside the class. Only the class members can access private members. Practically, we make data private and related functions public so that they can be called from outside of the class
protected	A protected member is very similar to a private member but it provided one additional benefit that they can be accessed in sub classes which are called derived/child classes.