A method which modifies values of object is called setter method. A method which does not modify values of object is called getter method.

Example:

```
class Time: # Data type
    def __init__(self):
        self.hh=0
        self.mm=0
        self.ss=0
    def getHH(self):
        return self.hh
    def getMM(self):
        return self.mm
    def getSS(self):
        return self.ss
    def setHH(self,h):
        self.hh=h
    def setMM(self,m):
        self.mm=m
    def setSS(self,s):
        self.ss=s



time_in=Time()
print(time_in.getHH(),time_in.getMM(),time_in.getSS())
time_in.setHH(7)
time_in.setMM(50)
time_in.setSS(40)
print(time_in.getHH(),time_in.getMM(),time_in.getSS())
```

Output

```
0 0 0
7 50 40
```

Advantage of encapsulation is data hiding, preventing data access from unrelated operations or outside functions.

Data hiding is achieved by defining members of the class as private.

Access modifiers
Access modifiers define the accessibility of the members of the class.

1. Private
2. Protected
3. Public

Default members (variables/methods) of the class are public.

Example:

```
class A:
    def m1(self): # public
        print("public m1 of A class")
    def __init__(self):
        self.x=100 # public
    def m2(self):
        print(self.x)



obj1=A()
obj1.m1()
print(obj1.x)
obj1.m2()
obj1.x=400
obj1.m2()
```

Output

```
public m1 of A class
100
100
400
```

Private members are accessible within class or members of the same class but cannot accessible outside the class or non members.
Private members of class are prefix with __ (double score).

Example:

```
class A:
    def __m1(self): # private method
        print("private method")
    def __int__(self):
        self.__x=100 # private I.V


obj1=A()
#obj1.__m1()
print(obj1.__x)
```

Output

```
Traceback (most recent call last):
  File "C:\Users\nit\PycharmProjects\project1\test15.py", line 10, in
<module>
    print(obj1.__x)
          ^^^^^^^^
AttributeError: 'A' object has no attribute '__x'
```

Hidden data is accessible using public methods.

Example:

```
class Complex: # Datatype
    def __init__(self,r=0.0,i=0.0):
        self.__real=r
        self.__imag=i
    def getReal(self):
        return self.__real
    def getImag(self):
        return self.__imag
    def setImag(self,i):
        self.__imag=i
    def setReal(self,r):
        self.__real=r


comp1=Complex()
# print(comp1.__real)
print(comp1.getReal(),comp1.getImag())
comp1.setReal(1.5)
comp1.setImag(2.5)
print(comp1.getReal(),comp1.getImag())
```

Output

```
0.0 0.0
1.5 2.5
```

Example:

```
class Account:  # Datatype
    def __init__(self,a,c,b):
        self.__accno=a
        self.__cname=c
        self.__balance=b
    def deposit(self,t):
        self.__balance=self.__balance+t
    def withdraw(self,t):
        if self.__balance<t:
```

```
            print("insuff balance")
        else:
            self.__balance=self.__balance-t
    def print_account(self):
        print(f'AccountNo {self.__accno}\nCustomerName
{self.__cname}\nBalance {self.__balance}')



def atm():
    acc1=Account(101,"naresh",5000)
    acc1.print_account()
    acc1.deposit(4000)
    acc1.print_account()
    acc1.withdraw(2000)
    acc1.print_account()
    acc1.withdraw(9000)

atm()
```

Output

```
AccountNo 101
CustomerName naresh
Balance 5000
AccountNo 101
CustomerName naresh
Balance 9000
AccountNo 101
CustomerName naresh
Balance 7000
insuff balance
```

Example:

```
class Stack: # Datatype
    def __init__(self):
```

```
        self.__s=[]
    def push(self,ele):
        self.__s.append(ele)
    def pop(self):
        if len(self.__s)==0:
            return "stack is empty"
        else:
            value=self.__s[-1]
            del self.__s[-1]
            return value


stack1=Stack()
stack1.push(10)
stack1.push(20)
print(stack1.pop())
print(stack1.pop())
print(stack1.pop())
```

Output

20

10

stack is empty

Class Level Variables