OOP

In [1]:
```
#class and Object
class Student:
    roll=""
    gpa=""

rahim=Student()
#Print isinstance(rahim,Student)) #testing object

rahim.roll=101
rahim.gpa=3.71
print(f" Roll: {rahim.roll} and GPA: {rahim.gpa}")

karim=Student()
karim.roll=109
karim.gpa=3.21
print(f" Roll: {karim.roll} and GPA: {karim.gpa}")
```

```
 Roll: 101 and GPA: 3.71
 Roll: 109 and GPA: 3.21
```

In [2]:
```
#class and Object
class Student:
    roll=""
    gpa=""
    #self look like this
    def setValue(self, roll, gpa):
        self.roll=roll
        self.gpa=gpa
    def display(self):
        print(f" Roll: {self.roll} and GPA: {self.gpa}")

rahim=Student()
rahim.setValue(101,3.75)
rahim.display()

karim=Student()
karim.setValue(109,3.21)
karim.display()
```

```
 Roll: 101 and GPA: 3.75
 Roll: 109 and GPA: 3.21
```

In [3]:
```python
#Constructor মেথড এর ব্যবহার কমায় ।
class Student:
    roll=""
    gpa=""
    def __init__(self, roll, gpa):
        self.roll=roll
        self.gpa=gpa

    def display(self):
        print(f" Roll: {self.roll} and GPA: {self.gpa}")

rahim=Student(101,3.75)
rahim.display()

karim=Student(109,3.21)
karim.display()
```

```
 Roll: 101 and GPA: 3.75
 Roll: 109 and GPA: 3.21
```

In [4]:
```python
class Triangle:
    def __init__(self, base,height):
        self.base=base
        self.height=height
    def printArea(self):
        area=0.5*self.base*self.height
        print("Area is",area)

t1=Triangle(10,20)
t1.printArea()

t2=Triangle(20,16)
t2.printArea()
```

```
Area is 100.0
Area is 160.0
```

In [5]:
```
#Inheritance
class Phone:
    #Parent/Super/Base class
    def call(self):
        print("Call")
    def message(self):
        print("Message")
class Asus(Phone):
    ##Asus class extends Phone
    #Child/Sub/Derived class
    def camera(self):
        print("Camera")
    def song(self):
        print("MP3")
a=Asus()
#asus class has two class property
a.call()
a.message()
a.camera()
a.song()
print(issubclass(Asus, Phone))
print("\n")
p=Phone()
p.call()
p.message()
print(issubclass(Phone, Asus))
```

```
Call
Message
Camera
MP3
True


Call
Message
False
```

In [6]:
```
#Method Over-riding
class Phone:
    def __init__(self):
        print("Phone Constructor")
class Asus(Phone):
    def __init__(self):
        super().__init__()
        print("ASUS Constructor")
        super().__init__()

a=Asus()
```

```
Phone Constructor
ASUS Constructor
Phone Constructor
```

In [7]:
```
#inheritance example
class Shape:
    def __init__(self, b, h):
        self.b=b
        self.h=h
    def area(self):
        print("AREA METHOD")
class Triangle(Shape):
    def area(self):
        area=0.5*self.b*self.h
        print("AREA Tri",area)
class Rectangle(Shape):
    def area(self):
        area=self.b*self.h
        print("AREA Rec",area)
r=Rectangle(10,20)
r.area()
t=Triangle(2,3)
t.area()
```

```
AREA Rec 200
AREA Tri 3.0
```

In [8]:
```
#different tye of inheritance
#MULTI-LEVEL
class A:
    def displey(self):
        print("A class")
class B(A):
    def displey2(self):
        print("B class")
class C(B):
    def displey3(self):
        print("C class")
        super().displey2()

cc=C()
cc.displey()
cc.displey2()
cc.displey3()
```

```
A class
B class
C class
B class
```

In [9]:
```
#MULTIPLE
#different tye of inheritance
#MULTI-LEVEL
class A:
    def displey(self):
        print("A class")
class B:
    def displey2(self):
        print("B class")
class C(A,B):
    def displey3(self):
        print("C class")
        super().displey2()

cc=C()
cc.displey()
cc.displey2()
cc.displey3()
```

```
A class
B class
C class
B class
```

In [10]:
```
#Abstruction
from abc import ABC, abstractmethod
class Shape(ABC):
    def __init__(self, b, h):
        self.b=b
        self.h=h
    @abstractmethod
    def area(self):
        pass
class Triangle(Shape):
    def area(self):
        area=0.5*self.b*self.h
        print("AREA Tri",area)
class Rectangle(Shape):
    def area(self):
        area=self.b*self.h
        print("AREA Rec",area)
s=Shape(10,12)
s.area()

r=Rectangle(10,20)
r.area()
t=Triangle(2,3)
t.area()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-55b106816b5c> in <module>
     16         area=self.b*self.h
     17         print("AREA Rec",area)
---> 18 s=Shape(10,12)
     19 s.area()
     20

TypeError: Can't instantiate abstract class Shape with abstract methods area
```

In [11]:
```
#Polimorphism
#Built in poymorphism
print(len("Minhazul Kabir"))
print(len([1,2,3]))
#User Define
def add(x,y,z=0):
    return x+y+z

print(add(7,17))
print(add(7,17,8))
```

```
14
3
24
32
```

In [12]:
```
#Magic Method
class Bike:
    def __init__(self, name, color):
        self.name=name
        self.color=color
    #check equality of other object SO THAT print(b1==b2) is true
    def __eq__(self, other):
        return self.name==other.name and self.color ==other.color

    def __str__(self):
        return (f"Name={self.name} Color={self.color}")
    def display(self):
        print(f"Name={self.name} Color={self.color}")
b1=Bike("Honda","Blue")
print(b1)
b1.display()
b2=Bike("Honda","Blue")
print(b1==b2)
```

```
Name=Honda Color=Blue
Name=Honda Color=Blue
True
```

In [13]:
```
#Math Module
from math import pow,sqrt
# from math import *
print(pow(2,3))
```

```
8.0
```

In [14]:
```
from moduleF import tri,rec
tri(10,20)
rec(10,20)
```

```
Area of Triangle = 100.0
Area of Rectangle = 200
```

In [15]:
```
#Regular Expression
import re
pattern=r"colour"
if re.match(pattern,"Red is a colour"):
    print("Match")
else:
    print("Not Match")

pp=r"colour"
if re.search(pp,"Red is a colour"):
    print("Match")
else:
    print("Not Match")
ppp=r"a"
print(re.findall(ppp,"Minhazul A Kabir"))
```

```
Not Match
Match
['a', 'a']
```

In [16]:
```
import re
patt=r"color"
text="My favourite color is White"
match=re.search(patt,text)
print(match)
if match:
    print(match.start())
    print(match.end())
    print(match.span())
```

```
<re.Match object; span=(13, 18), match='color'>
13
18
(13, 18)
```

In [17]:
```
#Search and Replace
import re
pat=r"color"
text="College color is blue and white. Flag color is green"
newT=re.sub(pat,"RONG",text)
print(newT)
newT=re.sub(pat,"COLOR",text, count=1)
print(newT)
```

```
College RONG is blue and white. Flag RONG is green
College COLOR is blue and white. Flag color is green
```

In [18]:

```
#Meta Character
import re
pattern=r"colo.r"
if re.match(pattern, "coloJra"):
    print("Match 1")
pattern=r"colo..r"
if re.match(pattern, "coloJAra"):
    print("Match 2")
pattern=r"^colo..r$"
if re.match(pattern, "coloJAr"):
    print("Match 3")
pattern=r"a*"
if re.match(pattern, "coloJr"):
    print("Match 4")
pattern=r"a+"
if re.match(pattern, "acoloJr"):
    print("Match a+")

pattern=r"a{1,3}$"
if re.match(pattern, "aaaa"):
    print("Match a aa aaa ")
```

```
Match 1
Match 2
Match 3
Match 4
Match a+
```