python oop

Encapsulation, Inheritance, polymorphism:

How to Design class?

```
class Student:
    college_name = "Hof University of Applied Science"
```

Parameterized constructor: — created new object

```
def __init__ (self, fullname, roll, age):
    print ("add new Student")
    self.name = fullname
    self.roll = roll
    self.age = age
```

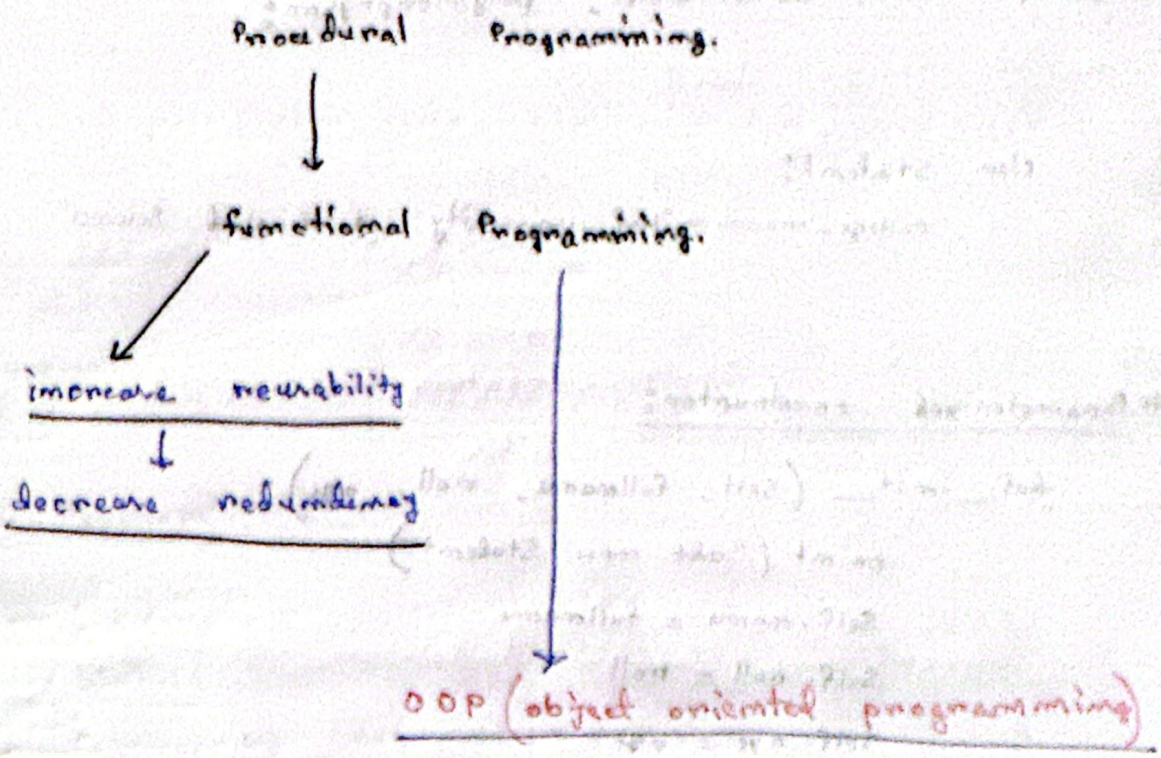
```
#   def welcome (self):
        Print ("welcome Student", self.name)
```

All classes have a function called init function. its executed when a function is called. object is being initiated.

```
S1 = Student ("Marud", 128, "29")
S2 = Student ("Rahel", 955, 30)
Print (S1.name, S1.Roll, S1.age)
Print (S1.name, S2.Roll, S2.age)

S2.welcome ( )
S2. Print (S2.collegeName)
```

OOP: In python:




```
class Student:
    name = "Nazmul Hasan"

S1 = Student()
Print (S1. name);
```

class is a blueprint of creating objects.

Methods:

Class is a collection of two things, attribute method.

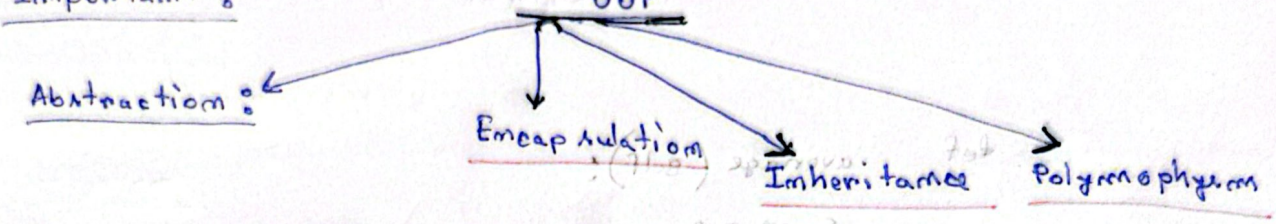
methods are functions that belongs to objects.

Static methods:

Static methods allow to write a method inside class without self parameter.

all the objects can use static Method. used for utility.

Important :



Abstraction : Hiding the implementation details.

Encapsulation : Wrapping data and functions into a Single unit.

Del keyword :

used to delete [object or] object properties.

```
del s1.name
del s1
```

Private Attribute : Self. __ pass

~~Self. __ pass = pass~~

Private methods : ~~def __hello( ):~~

```
def __hello( ):
```

Inheritence in python:

→ When one class derived the properties of other class.

class car:

Single level of inheritance.

multi level of inheritance.

multiple inheritance.

class

class Toyota (car):

class porsche (Toyota):

def __init__ (self, type):

self.type = type

s1 = porsche ("Diesel")

s1.start()

s2.stop()

ltiple inheritance:

Method:

class Method:

Static Method.

Instance Method.

#

Property:

attribute value ⟶ function

Property.

decorator

↓

@ Property

```
def percentage (self):
    return str ((self.phy + self.che + self.Math)/3) + "%"
```

Polymorphism: Operator overloading:

```
def ShowNumber (self):
    Print( self.img , f"i + " , self.real , " j")
```

dunder function:

~~a + b~~ python special methods that will allow operator overloading and custom object behavior.

object overloading:

when you can use the same operator for different meaning.