MADE BY: YASMEN AHMED FADEL

ASS OF AI

TO ENG: AHMED KHALED

**Dunder method:-**

 dunder method its a method which has two underscores before it and two underscores after it.

These are commonly used for operator overloading. Few examples for magic methods are: \_\_init\_\_, \_\_add\_\_, \_\_len\_\_, \_\_repr\_\_ etc….

# declare our own string class

class String:

    # magic method to initiate object

    def \_\_init\_\_(self, string):

        self.string = string

    # print our string object

    def \_\_repr\_\_(self):

        return 'Object: {}'.format(self.string)

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

    # object creation

    string1 = String('you can get it')

    # print object location

    print(string1)

**Abstraction and Encapsulation :-**

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| It is the process of gaining information | It is a method that helps wrap up data into a single module |
| The problems in this technique are solved at the interface level | Problems in encapsulation are solved at the implementation level |
| It can hide unwanted information | The data is hidden using methods such as getters and setters |
| It can be implemented using abstract classes and interfaces | It helps hide data using a single entity, or using a unit with the help of method that helps protect the information |
| The complexities of the implementation are hidden using interface and abstract class | It can be implemented using access modifiers like public, private and protected |
| The abstraction can be performed using objects that are encapsulated within a single module | Objects in encapsulation don’t need to be in abstraction |
| from abc import ABC  class geometric(ABC):      def volume(self):  #abstract method       pass  class Rect(geometric):   length = 4   width = 6   height= 6   def volume(self):    return self.length \* self.width \*self.height  class Sphere(geometric):    radius = 8    def volume(self):     return 1.3 \* 3.14 \* self.radius \* self.radius \*self.radius  class Cube(geometric):   Edge = 5   def volume(self):    return self.Edge \* self.Edge \*self.Edge  class triangle\_3D:   length = 5   width = 4   def volume(self):    return 0.5 \* self.length \* self.width  rr = Rect()  ss = Sphere()  cc = Cube()  tt = triangle\_3D()  print("Volume of a rectangle:", rr.volume())  print("Volume of a circle:", ss.volume())  print("Volume of a square:", cc.volume())  print("Volume of a triangle:", tt.volume()) | class Base:      def \_\_init\_\_(self):          # Protected member          self.\_a = 2  # Creating a derived class  class Derived(Base):      def \_\_init\_\_(self):          # Calling constructor of          # Base class          Base.\_\_init\_\_(self)          print("Calling protected member of base class: ",self.\_a)          # Modify the protected variable:          self.\_a = 3          print("Calling modified protected member outside class: ",self.\_a)  obj1 = Derived()  obj2 = Base()  # Calling protected member  # Can be accessed but should not be done due to convention  print("Accessing protected member of obj1: ", obj1.\_a)  # Accessing the protected variable outside  print("Accessing protected member of obj2: ", obj2.\_a) |

**Difference between \*\*args and \*\*kwordargs:-**

|  |  |
| --- | --- |
| \*\*args | \*\*kwordargs |
| It is used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list. | is used to pass a keyworded, variable-length argument list |
| any number of extra arguments can be tacked on to our current formal parameters | A keyword argument is where ه provide a name to the variable as ه pass it into the function. |
| the variable that we associate with the \* becomes an iterable | I can name my arguments |
| creates tuple | creates dictionary |
| def Fun(\*argv):      for arg in argv:          print(arg)      Fun('Hello', 'yas', 'to', 'my assignment') | def Fun(\*\*kwargs):      for key, value in kwargs.items():          print("%s == %s" % (key, value))      # Driver code  Fun(first='yass', mid='made', last='assignments') |