Object Oriented Programming (Python)

In [ ]:
```
'''

Hridoy Ahmed
Daffodil International University
Dept. of Computer Science & Engineering
Web: hridoyahmed.pythonanywhere.com
Email: hridoy15-7981@diu.edu.bd
Facebook Profile: https://web.facebook.com/HridoyAhmedCSE

'''
```

In [ ]:
```

```

In [ ]:
```
'''
Object Oriented Programming allows programmers to create their own objec
In general, OOP allows us to create code that is repeatable and organize
Methods act as fuctions that use information about the Object
Method Systax: .method_name()  like, list_item.append()
'''
```

In [ ]:
```

```

In [1]:
```
# class NameOfClass():
#     def __init__(self, param1, param2):
#         self.param1 = param1
#         self.param2 = param2

#     def some_method(self):
#         #Perform some action
#         print(self.param1)
```

In [2]:
```
class Dog():

    # Class object attribute
    # Same for any instance of a class

    species = 'mammal'

    # Dunder Method __init__ for initializing a class.
    def __init__(self, breed, name, spots):

        # Attributes
        # We take in the argument
        # Assign it using self.attribute_name

        self.breed = breed
        self.name = name
        self.spots = spots

    #OPERATIONS/Actions -----> Methods
    def bark(self, number):
        print(f'Gheu Gheu!!! My Name is {self.name} and the number is {n
```

In [3]:
```
my_dog = Dog(breed='Huskie', name='Rock', spots="No Spots")
```

In [4]:
```
my_dog.name
```

Out[4]:
```
'Rock'
```

In [5]:
```
my_dog.breed
```

Out[5]:
```
'Huskie'
```

In [6]:
```
my_dog.spots
```

Out[6]:
```
'No Spots'
```

In [7]:
```
my_dog.species
```

Out[7]:
```
'mammal'
```

In [8]:
```
my_dog.bark
```

Out[8]:
```
<bound method Dog.bark of <__main__.Dog object at 0x7f8a58138970>>
```

In [9]:
```
my_dog.bark(10)
```

```
Gheu Gheu!!! My Name is Rock and the number is 10
```

In [ ]:

In [ ]:

In [ ]:

In [10]:
```
class Circle():

    # CLASS OBJECT ATTRIBUTE
    pi = 3.1416

    def __init__(self, radius=1):

        self.radius = radius
        self.area = radius * radius * Circle.pi
        '''
        radius hocche parameter r pi hocche attribute.
        attribute use korte hole classname.atrributeName
        dite hoy.
        '''

    #METHOD
    def get_circumference(self):

        #Poridhi = 2 * pi * r
        return self.radius * Circle.pi * 2
```

In [11]:
```
my_circle = Circle(20)
```

In [12]:
```
my_circle.get_circumference()
```

Out[12]: 125.664

In [13]:
```
my_circle.area
```

Out[13]: 1256.6399999999999

In [ ]:

In [ ]:

In [ ]:

In [14]:
```
# Inheritance
```

In [ ]:
```
'''
It's a way to form new classes using classes that have already been defi
mane hocche ekta class er details onno ekta class a use kora.
Inheritance er 2 ta class thake ekta base onnota derived.
Base hocche hocche j class er details amra onno class a use korbo.
R derived class hocche j class a amra base class use korbo.

Benefits:
Ability to reuse code someone have already worked on.
To reduce the complexity of a program.
'''
```

In [ ]:
```
```

In [15]:
```
# Base class
class Animal():

    def __init__(self):
        print('Animal Created')

    def who_am_i(self):
        print('I am an animal')

    def eat(self):
        print('I am eating')
```

In [16]:
```
# derived class
'''
derived class er bracket er vitor base class er nam likhte hoy.
BaseClassName.__init__(self) diye Bas class er sob self attribute
derived class a niye asa hoy.
'''
class Cat(Animal):

    def __init__(self): # Instance of Cat Class
        Animal.__init__(self) #Instance of Animal Class
        print('Cat Created')

    def who_am_i(self):
        print('I am a cat')

    def fav_dish(self):
        print('Milk')
```

In [17]:
```
my_cat = Cat()
```

```
Animal Created
Cat Created
```

In [18]:
```
'''
method er por ovossoie () dite hobe ta chara shuru method
er address location dekhabe output show korbe na.
'''
my_cat.who_am_i
```

Out[18]:
```
<bound method Cat.who_am_i of <__main__.Cat object at 0x7f8a58138580>>
```

In [19]:
```
my_cat.who_am_i()
```

```
I am a cat
```

In [20]:
```
my_cat.fav_dish()
```

```
Milk
```

In [ ]:

In [ ]:

In [21]:
```
# Polymorphism
```

In [ ]:
```
'''
Polymorphism refers to the way in which different object can share the s
And then those methods can be called from the same place
even though a variety of different objects might be passed in.

Suppose, ami 5 ta class a inheritance apply korchi
othoba 5 ta class ache jekhane prottek class er method name
same kintu ami chacchi method name same thakleo alada alada
data hold korbe sekhetre polymorphism er dorkar pore.
'''
```

In [ ]:

In [22]:
```
# Polymorphism in diffrent classes
```

In [23]:
```
class Dog():

    def __init__(self, name):
        self.name  = name

    def speak(self):
        return self.name + " says woof!"
```

In [24]:
```
class Cat():

    def __init__(self, name):
        self.name  = name

    def speak(self):
        return self.name + " says meow!"
```

In [25]:
```
niko = Dog('Niko')
felix = Cat('Felix')
```

In [26]:
```
print(niko.speak())
```

```
Niko says woof!
```

In [27]:
```
print(felix.speak())
```

```
Felix says meow!
```

In [28]:
```
for pet_class in [niko, felix]:
    print(type(pet_class))
    print(pet_class.speak())
```

```
<class '__main__.Dog'>
Niko says woof!
<class '__main__.Cat'>
Felix says meow!
```

In [29]:
```
def pet_speak(pet):
    print(pet.speak())
```

In [30]:
```
pet_speak(niko)
```

```
Niko says woof!
```

In [31]:
```
pet_speak(felix)
```

```
Felix says meow!
```

In [ ]:
```

```

In [32]:
```
# Polymorphism with Inheritance:
```

In [33]:
```
class Bird():
  def intro(self):
    print("There are many types of birds.")

  def flight(self):
    print("Most of the birds can fly but some cannot.")
```

In [34]:
```
class sparrow(Bird):
  def flight(self):
    print("Sparrows can fly.")
```

In [35]:
```
class ostrich(Bird):
  def flight(self):
    print("Ostriches can not fly.")
```

In [36]:
```
obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()
```

In [ ]:
```

```

In [37]:
```
obj_bird.intro()
obj_bird.flight()
```

```
There are many types of birds.
Most of the birds can fly but some cannot.
```

In [38]:
```
obj_spr.intro()
obj_spr.flight()
```

```
There are many types of birds.
Sparrows can fly.
```

In [39]:
```
obj_ost.intro()
obj_ost.flight()
```

```
There are many types of birds.
Ostriches can not fly.
```

In [ ]:
```

```

In [40]:
```
# Abstract Classes
```

In [ ]:
```
'''
Never expects to be instantiated.
You never expect to create an instance of this class.
Instead it's just designed to basically only serve as a base class.

Orthat ami amar base class a amon ekta method chacchi jake call kora
jabe na kintu onno derived class a oi method take ovossoie
update korte hobe sekhetre oi method wala class ke abstract class bole.
'''
```

In [41]:
```
'''
raise diye Customize error dekhano jay.
'''
class Animal():

    def __init__(self, name):
        self.name = name

    def speak(self):
        raise NotImplementedError('Subclass must implement this abstract
```

In [42]:
```
class Dog(Animal):

    def speak(self):
        return self.name + ' says woof!'
```

In [43]:
```
class Cat(Animal):

    def speak(self):
        return self.name + ' says meow!'
```

In [44]:
```
fido = Dog('Fido')
```

In [45]:
```
isis = Cat('Isis')
```

In [46]:
```
print(fido.speak())
```

```
Fido says woof!
```

In [47]:
```
print(isis.speak())
```

```
Isis says meow!
```

In [ ]:
```

```

In [48]:
```
# Recursive Function

def factor(y):
    if y == 0:
        return 1

    return y * factor(y - 1)

print(factor(4))
```

```
24
```

In [49]:
```
4 * 3 * 2 * 1 * 1
```

Out[49]:
```
24
```

In [ ]:
```

```

In [50]:
```
# Base Class

class Human():

    # CLASS OBJECT ATTRIBUTES

    legs = 2
    hands = 2
    head = 1
    ears = 2
    eyes = 2
    nose = 1
    mouth = 1

    def __init__(self):

        print('Human Class Created')

    # METHODS
    def eat(self):
        print('Eat to survive')

    # Abstract Class
    def work_place(self):
        raise NotImplementedError('Subclass must implement this abstract
```

In [ ]:
```
```

In [51]:
```
human = Human()
```

```
Human Class Created
```

In [52]:
```
human.work_place()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call
 last)
<ipython-input-52-39010dcc7098> in <module>
----> 1 human.work_place()

<ipython-input-50-f0cc268fc1d8> in work_place(self)
     23     # Abstract Class
     24     def work_place(self):
---> 25         raise NotImplementedError('Subclass must implement thi
s abstract method')

NotImplementedError: Subclass must implement this abstract method
```

In [ ]:
```
```

In [53]:
```
# Derived Class

class Female(Human):

    def __init__(self, chromosome): # Instance of Female Class

        Human.__init__(self) # Instance of Human Class

        print('Female Class Created')

        self.chromosome = chromosome

    def work_place(self):
        print('A large no of women work at home')
```

In [ ]:

In [54]:
```
# Derived Class

class Male(Human):

    def __init__(self, chromosome, beard):
        Human.__init__(self)

        print('Male Class Created')
        # Attributes
        self.chromosome = chromosome
        self.beard = beard

    def work_place(self):
        print('A large no of men work at office')
```

In [ ]:

In [55]:
```
male1 = Male('XY', 'Yes')
```

```
Human Class Created
Male Class Created
```

In [ ]:

In [56]:
```
female1 = Female('XX')
```

```
Human Class Created
Female Class Created
```

In [ ]:

In [57]:
```
male1.ears
```

Out[57]:
```
2
```

In [58]:
```
female1.eyes
```

Out[58]: 2

In [59]:
```
male1.eat
```

Out[59]: <bound method Human.eat of <__main__.Male object at 0x7f8a58112850>>

In [60]:
```
male1.eat()
```

```
Eat to survive
```

In [61]:
```
female1.eat()
```

```
Eat to survive
```

In [62]:
```
male1.work_place()
```

```
A large no of men work at office
```

In [63]:
```
female1.work_place()
```

```
A large no of women work at home
```

In [64]:
```
male1.chromosome
```

Out[64]: 'XY'

In [65]:
```
female1.chromosome
```

Out[65]: 'XX'

In [ ]:
```

```

In [66]:
```
# Easiest Example of Abstract Class
```

In [67]:
```
# Base Class

class User():

    def __init__(self, username, password):

        self.username = username
        self.password = password

        print('User Registration Successfull')

    def login(self):
        raise NotImplementedError ('User must login with Credentials')
```

In [68]:
```
user1 = User('Hridoy', '1234')
```

```
User Registration Successfull
```

In [69]:
```
user1.username
```

Out[69]:
```
'Hridoy'
```

In [70]:
```
user1.password
```

Out[70]:
```
'1234'
```

In [71]:
```
user1.login()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-71-ed78c0a45454> in <module>
----> 1 user1.login()

<ipython-input-67-b8d6f17e5004> in login(self)
     11
     12     def login(self):
---> 13         raise NotImplementedError ('User must login with Credentials')

NotImplementedError: User must login with Credentials
```

In [72]:
```
# Derieved class

class RegisteredUser(User):

    def __init__ (self):
        #User.__init__(self)
        pass

    def login(self):
        print('Login Successful')
```

In [73]:
```
reguser1 = RegisteredUser()
```

In [74]:
```
reguser1.login()
```

```
Login Successful
```

In [ ]:

In [ ]:

In [75]:
```
# Special (MagicDunder) Methods
```

In [76]:
```
# Example:

my_list = [1, 2, 3, 9]

print(len(my_list))

print(type(my_list))
```

```
4
<class 'list'>
```

In [ ]:
```

```

In [77]:
```
class Sample():
    pass
```

In [78]:
```
mysample = Sample()
```

In [79]:
```
print(mysample)
```

```
<__main__.Sample object at 0x7f8a58112340>
```

In [80]:
```
print(type(mysample))
```

```
<class '__main__.Sample'>
```

In [ ]:
```

```

In [81]:
```
class Books():

    def __init__(self, title, author, pages):

        self.title = title
        self.author = author
        self.pages = pages
```

In [82]:
```
book1 = Books('psychoCoders', 'Hridoy', 500)
```

In [83]:
```
print(book1)
```

```
<__main__.Books object at 0x7f8a58112370>
```

In [84]:
```
str(book1)
```

Out[84]:
```
'<__main__.Books object at 0x7f8a58112370>'
```

In [85]:
```
len(book1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-85-29366f8bf81e> in <module>
----> 1 len(book1)

TypeError: object of type 'Books' has no len()
```

In [87]:
```
del book1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-87-3c1a9d1f5138> in <module>
----> 1 del book1

NameError: name 'book1' is not defined
```

In [88]:
```
book1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-88-505b52038518> in <module>
----> 1 book1

NameError: name 'book1' is not defined
```

In [ ]:
```

```

In [89]:
```
'''
Uporer error handle korar jonno amra magicDunder mathod
like __str__, __len__, __del__ etc. use kori.
'''
class Books():

    def __init__(self, title, author, pages):

        self.title = title
        self.author = author
        self.pages = pages

    def __str__(self):
        return f'{self.title} by {self.author}'

    def __len__(self):
        return self.pages

    def __del__(self):
        print('A book object has been deleted, create another one.')
```

In [90]:
```
book1 = Books('psychoCoders', 'Hridoy', 500)
```

In [91]:
```
print(book1)
```

```
psychoCoders by Hridoy
```

In [92]:
```
str(book1)
```

Out[92]:
```
'psychoCoders by Hridoy'
```

In [93]:
```
len(book1)
```

Out[93]:
```
500
```

In [94]:
```
del book1
```

```
A book object has been deleted, create another one.
```

In [95]:
```
book1
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-95-505b52038518> in <module>
----> 1 book1

NameError: name 'book1' is not defined
```

In [96]:
```
book1 = Books('Pyhton-Noob2Pro', 'Hridoy Ahmed', 200)
```

In [97]:
```
book1.pages
```

Out[97]: 200

In [98]:
```
print(book1)
```

```
Pyhton-Noob2Pro by Hridoy Ahmed
```

In [ ]:
```
================================ Done ================================
```