

Object Oriented Programming (OOP)



Hello!

I am Eslam Ahmed

I am a software engineer.

You can find me at jeksogsa@gmail.com



Why OOP ?!

Let's see Why ^_~



Calculate Sum 10000 times ?!

```
1 x = 10
2 y = 20
3 print(x + y)
4
5 x = 50
6 y = 70
7 print(x + y)
8
9 x = 90
10 y = 200
11 print(x + y)
```



Calculate Sum 10000 times ?!

What if I want to modify code to multiply ?



Calculate Sum 10000 times ?!

What if I w



```
1 x = 10
2 y = 20
3 print(x + y)
4
5 x = 50
6 y = 70
7 print(x + y)
8
9 x = 90
10 y = 200
11 print(x + y)
```

Albert Einstein: Insanity is doing the same thing over and over and expecting different results



Calculate Sum 10000 times ?!

What if I want to modify code to multiply ?

```
1 def calculate(x, y):  
2     return x + y # to multiply just make '+' -> '*'  
3  
4 calculate(1, 2) # 3  
5 calculate(10, 20) # 30  
6 calculate(50, 40) # 90  
7 ...
```

Much Better ^_^



Make Full Calculator ?

```
1 def summ(x, y):  
2     return x + y  
3  
4 def multiply(x, y):  
5     return x * y  
6  
7 def subtract(x, y):  
8     return x - y  
9  
10 def divide(x, y):  
11     return x / y  
12  
13 summ(1, 2) # 3  
14 multiply(10, 20) # 200  
15 subtract(50, 40) # 10  
16 divide(50, 2) # 25  
17 ...
```



Make Lots of Features ?

```
● ● ●  
1 def summ(x, y):  
2     pass  
3  
4 def subtract(x, y):  
5     pass  
6  
7 def read_file(path):  
8     pass  
9  
10 def write_data_in_file(data, path):  
11    pass  
12  
13 def get_data_from_internet(url):  
14    pass  
15  
16 def delete_data_from_database(data):  
17    pass  
18  
19 ...
```



Make Lots of Features ?

```
1 def summ(x, y):  
2     pass  
3  
4 def subtract(x, y):  
5     pass  
6  
7 def read_file(path):  
8     pass  
9  
10 def write_data_in_file(da  
11     pass  
12  
13 def get_data_from_interne  
14     pass  
15  
16 def delete_data_from_data  
17     pass  
18  
19 ...
```



What is OOP ?!

Everything is an
Object



Dog

Properties

- Size
- Color
- Breed
- Age
- ...

Methods

- Run()
- Park()
- Eat()
- Sleep()
- ...



Car

Properties

- Model
- Color
- Plate Number
- Speed
- ...

Methods

- Move_Forward()
- Move_Backward()
- Break()
- Throttle()
- ...



Person

Properties

- Name
- Age
- Address
- Gender
- ...

Methods

- Eat()
- Sleep()
- Walk()
- Pray()
- ...



Camera

Properties

- Lens_width
- Has_Flash
- Depth
- ...

Methods

- Take_Photo()
- Take_Video()
- Toggle_Flash()
- Zoom()
- ...



Machine Learning Model

Properties

- Name
- Learning_rate
- Num_of_iterations
- Weights
- ...

Methods

- fit()
- predict()
- ...



Classes & Objects



Person

Properties

- Name
- Age
- Address
- Gender
- ...

Methods

- Eat()
- Sleep()
- Walk()
- Pray()
- ...



Class: Person

Object 1

- Name: Eslam
- Age: 26
- Gender: Male
- Eat Meat
- Sleep at 9 pm

Object 2

- Name: Ahmed
- Age: 15
- Gender: Male
- Eat Chicken
- Sleep at 12 am

Object 3

- Name: Sara
- Age: 30
- Gender: Female
- Eat Fish
- Sleep at 6 am

Machine Learning Model

Properties

- Name
- Learning_rate
- Num_of_iterations
- Weights
- ...

Methods

- fit()
- predict()
- ...



Class: Machine Learning Model

Object 1

- Name: Linear Regression
- ...
- Fit()
- Predict()

Object 2

- Name: SVM
- ...
- Fit()
- Predict()



Example of Classes & Objects in Python ?



```
1 from sklearn.preprocessing import StandardScaler
2 from sklearn.svm import SVC
3
4 scaler = StandardScaler()
5 x = scaler.fit_transform(x)
6
7 model = SVC()
8 model.fit(x_train, y_train)
9 y_pred = model.predict(x_test)
10
11
```





```
1 class Circle:  
2     pi = 3.14  
3  
4     # Circle gets instantiated with a radius (default is 1)  
5     def __init__(self, radius=1):  
6         self.radius = radius  
7  
8  
9     # Method for getting Circumference  
10    def getcircumference(self):  
11        return self.radius * self.pi * 2  
12  
13  
14    # Method for getting Area  
15    def getArea(self):  
16        return self.radius * self.radius * self.pi  
17  
18  
19 c = Circle()  
20  
21 print('Radius is: ',c.radius)  
22 print('Circumference is: ',c.getcircumference())  
23 print('Area is: ',c.getArea())  
24
```

OOP in Python ?



Circle

```
1 class Person:
2
3     def __init__(self, name, age, gender):
4         self.name = name
5         self.age = age
6         self.gender = gender
7
8
9     def greet(self):
10        if self.gender == 'male':
11            print('Hello, Mr. ' + self.name)
12        elif self.gender == 'female':
13            print('Hello, Mrs. ' + self.name)
14
15
16    def is_old(self):
17        return (self.age >= 60)
18
19
20 ahmed = Person('ahmed', 20, 'male')
21 mohammed = Person('mohammed', 67, 'male')
22 sara = Person('sara', 30, 'female')
23
24 ahmed.is_old() # false
25 mohammed.is_old() # true
26
27 ahmed.greet() # Hello, Mr. ahmed
28 sara.greet() # Hello, Mrs. sara
29
```

OOP in Python ?



Person

OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- Polymorphism
- Special Functions and Operators Overloading

Data Hiding (Encapsulation)



Data Hiding (Encapsulation)

An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then are not be directly visible to outsiders.

Data Hiding (Encapsulation)

```
1 class Circle:  
2     pi = 3.14  
3  
4  
5     # Circle gets instantiated with a radius (default is 1)  
6     def __init__(self, radius=1):  
7         self.__radius = radius  
8  
9     # Method for resetting Radius  
10    def getArea(self):  
11        return self.__radius * self.__radius * self.pi  
12  
13    # Method for getting Circumference  
14    def getCircumference(self):  
15        return self.__radius * self.pi * 2  
16  
17  
18  
19 c1 = Circle(10)  
20  
21 print('Radius is: ',c1.__radius)  
22  
23 """  
24 AttributeError: 'Circle' object has no attribute '__radius'  
25 """
```



OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- Polymorphism
- Special Functions and Operators Overloading

Inheritance



Inheritance

It is a way to form new classes using classes that have already been defined. The newly formed classes are called derived classes, the classes that we derive from are called base classes.

```
1 class Vehicle():
2
3     def __init__(self, speed, fuel_type):
4         self.fuel_type = fuel_type
5         self.speed = speed
6
7     def drive(self):
8         print('Driving with speed: ' + str(self.speed))
9
10    def stop(self):
11        print('stopping...')
12
13 # -----
14
15 class Car(Vehicle):
16
17     def drive(self):
18         print('start with ur key and power me on')
19
20 class Plane(Vehicle):
21
22     def drive(self):
23         print('Im Flying')
24
25 # -----
26
27 mitshpishi = Car(100, 'gas')
28 mitshpishi.drive() # start with ur key and power me on
29 mitshpishi.stop() # stopping...
30
31 planel = Plane(700, 'diesel')
32 planel.drive() # Im Flying
33 planel.stop() # stopping...
34
```

Inheritance



OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- Polymorphism
- Special Functions and Operators Overloading

Polymorphism



Polymorphism

polymorphism refers to the way in which different object classes can share the same method name but each acts differently.

Polymorphism

```
1 class Dog:  
2     def __init__(self, name):  
3         self.name = name  
4  
5     def speak(self):  
6         return self.name+' says Woof!'  
7  
8 class Cat:  
9     def __init__(self, name):  
10        self.name = name  
11  
12    def speak(self):  
13        return self.name+' says Meow!'  
14  
15  
16 jack = Dog('Jack')  
17 meshmesh = Cat('Meshmesh')  
18  
19  
20 for pet in [jack,meshmesh]:  
21     print(pet.speak())
```



OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- Polymorphism
- Special Functions and Operators Overloading

Special Functions and Operators Overloading

Math Operator

- __add__
- __sub__
- __mul__
- __truediv__
- __floordiv__
- __mod__
- __pow__

Comparison Operator

- __eq__
- __nq__
- __lt__
- __le__
- __gt__
- __ge__

Special Functions and Operators Overloading

Other Operators

- hexagon icon `__init__`
- hexagon icon `__str__`
- hexagon icon `...`

Other Concepts

- ▷ Exceptions
- ▷ Decorators
- ▷ Iterators and Generators



Exceptions

```
1 try:  
2     val = int(input("Please enter an integer: "))  
3 except:  
4     print("Looks like you did not enter an integer!")  
5 finally:  
6     print("Finally, I executed!")  
7
```

<https://www.tutorialsteacher.com/python/error-types-in-python>

Other Concepts

- ◊ Exceptions
- ◊ Decorators
- ◊ Iterators and Generators



Decorators

```
1 def new_decorator(func):
2
3     def wrap_func():
4         print("Code would be here, before executing the func")
5         func()
6         print("Code here will execute after the func()")
7
8     return wrap_func
9
10 def func_needs_decorator():
11     print("This function is in need of a Decorator")
12
13
14 func_needs_decorator()
15 """
16 This function is in need of a Decorator
17 """
18
19 func_needs_decorator = new_decorator(func_needs_decorator)
20
21 func_needs_decorator()
22 """
23 Code would be here, before executing the func
24 This function is in need of a Decorator
25 Code here will execute after the func()
26 """
```

```
1 def new_decorator(func):
2
3     def wrap_func():
4         print("Code would be here, before executing the func")
5
6         func()
7
8         print("Code here will execute after the func()")
9
10    return wrap_func
11
12
13 @new_decorator
14 def func_needs_decorator():
15     print("This function is in need of a Decorator")
16
17
18 func_needs_decorator()
19 """
20 Code would be here, before executing the func
21 This function is in need of a Decorator
22 Code here will execute after the func()
23 """
```

Other Concepts

- ◊ Exceptions
- ◊ Decorators
- ◊ Iterators and Generators



Iterators

```
● ● ●  
1 # define a list  
2 my_list = [4, 7, 0, 3]  
3  
4 # get an iterator using iter()  
5 my_iter = iter(my_list)  
6  
7 ## iterate through it using next()  
8  
9 # prints 4  
10 print(next(my_iter))  
11  
12 # prints 7  
13 print(next(my_iter))  
14  
15 ## next(obj) is same as obj.__next__()  
16  
17 # prints 0  
18 print(my_iter.__next__())  
19  
20 # prints 3  
21 print(my_iter.__next__())  
22  
23 ## This will raise StopIteration error, no items left  
24 next(my_iter)
```



Custom Iterators

```
 1 class PowTwo:
 2
 3     def __init__(self, max = 0):
 4         self.max = max
 5
 6     def __iter__(self):
 7         self.n = 0
 8         return self
 9
10    def __next__(self):
11        if self.n <= self.max:
12            result = 2 ** self.n
13            self.n += 1
14            return result
15        else:
16            raise StopIteration
17
18
19 a = iter(PowTwo(4))
20 a.__next__() # 1
21 a.__next__() # 4
22 a.__next__() # 8
23 a.__next__() # 16
24 a.__next__() # 32
25 a.__next__() # Raise StopIteration Exception
```



Generator

Much Better

no need to implement `__iter__()` and `__next__()`

```
1 def my_gen():
2     n = 1
3     print('This is printed first')
4     yield n
5
6     n += 1
7     print('This is printed second')
8     yield n
9
10    n += 1
11    print('This is printed at last')
12    yield n
13
14 a = my_gen()
15 a.__next__() # This is printed first 1
16 a.__next__() # This is printed second 2
17 a.__next__() # This is printed at last 3
18 a.__next__() # Raise StopIteration Exception
19
20 #####
21
22 for item in my_gen():
23     print(item)
24
25 """
26 This is printed first
27 1
28 This is printed second
29 2
30 This is printed at last
31 3
32 """
```

Questions ?!



Thanks!

>_ Live long and prosper

