Object Oriented Programming in Python

Terminology

I want to start with terminolgic words which uses in OOP in Python.

- Class
  - A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class Variable
  - A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data Member
  - A class variable or instance variable that holds data associated with a class and its objects.
- Function Overloading
  - The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.
- Instance Variable
  - A variable that is defined inside a method and belongs only to the current instance of a class.
- Inheritence
  - The transfer of the characteristics of a class to other classes that are derived from it.
  - Diamond Problem
- Encapsualtion
  - selam falan
- Instance
  - An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- Instantiation
  - The creation of an instance of a class.
- Method
  - A special kind of function that is defined in a class definition.
- Object
  - A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- Operator Overloading
  - The assignment of more than one function to a particular operator.
- Constructors
  - Constructors are generally used for instantiating an object.The task of constructors is to initialize(assign values) to the

data members of the class when an object of class is created.In Python the *init*() method is called the constructor and is always called when an object is created.

- Context Manager
    - With, yield usage falan
- Generator Functions
- Garbage Collector
    - Selam falan

Classes

Define A Class

```
class Person():

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


first_person = Person("Selami", 77, "Male")

print(first_person.name)
print(first_person.age)
print(first_person.sex)
```

Output

```
Selami
77
Male
```

- The first method *init*() is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.

Define A Class with Default Paremeters

```
class Person():

    def __init__(self, name="Regen KID", age=23, sex=None):
        self.name = name
        self.age = age
        self.sex = sex


person_one = Person()
person_two = Person(name="Debbie Harry", age=75, sex="Female")

print(f"person_one's Name: {person_one.name}, Age: {person_one.age}, Sex: {person_one.sex}")
print(f"person_two's Name: {person_two.name}, Age: {person_two.age}, Sex: {person_two.sex}")
```

Output

```
person_one's Name: Regen KID, Age: 23, Sex: None
person_two's Name: Debbie Harry, Age: 75, Sex: Female
```

- Classes can own functions (a.k.a 'METHOD').

Class Methods

```
import random

class Person():

    def __init__(self, name="Regen KID", age=23, sex=None):
        self.name = name
        self.age = age
        self.sex = sex
        self.location = "Turkey"

    def increase_age(self):
        self.age = self.age + 1

    def create_bank_account(self):
        self.account_id = random.randint(10000000, 99999999)

    def get_account_id(self):
        return self.account_id

    def move_to_another_country(self, new_country:str):
        self.location = new_country

person_two = Person(name="Debbie Harry", age=75)

print(f"person_two's Name: {person_two.name}, Age: {person_two.age}, Location: {person_two.location}")
person_two.create_bank_account()
print(f"{person_two.name}'s bank acoount id is: {person_two.get_account_id()}")
```

Output

```
person_two's Name: Debbie Harry, Age: 75, Location: Turkey
Debbie Harry's bank acoount id is: 37652487
```

Encapsulation

- We can reach and change the objects variables. Sometetimes We may prohibit the access from out of scope (for security etc.). So If We will use encapsulation, we can prevent undesired usages.

Encapsulation of Object's variables

```
class Person():

    def __init__(self, name, age=23, sex=None):
        self.__name = name
        self.age = age
        self.sex = sex

    def get_name(self):
        return self.__name


person_two = Person(name="Debbie Harry", age=75, sex="Female")
print(f"person_two's Name: {person_two.name()}, Age: {person_two.age}, Sex: {person_two.sex}")
print(f"person_two's Name: {person_two.get_name()}, Age: {person_two.age}, Sex: {person_two.sex}")
```

Output

```
AttributeError: 'Person' object has no attribute 'name'
```

- We couldn't get name value from out of class scope, so we wrote a getter function to reach this variable. With Encapsulation, we can control attributes.

Get and Set Encapsulated Object's variables

```
class Person():

    def __init__(self, name, age=23, sex=None):
        self.__name = name
        self.age = age
        self.sex = sex

    def get_name(self):
        return self.__name

    def set_name(self, new_name):
        self.__name = new_name


person_two = Person(name="Debbie Harry", age=75, sex="Female")
print(f"person_two's Name: {person_two.get_name()}, Age: {person_two.age}, Sex: {person_two.sex}")
person_two.set_name("New Debbie")
print(f"person_two's Name: {person_two.get_name()}")
```

Output

```
person_two's Name: Debbie Harry, Age: 75, Sex: Female
person_two's Name: New Debbie
```

Inheritence

- Inheritance models what is called an is a relationship. This means that when you have a Derived class that inherits from a Base class, you created a relationship where Derived is a specialized version of Base.
    - Classes that inherit from another are called derived classes, subclasses, or subtypes.
    - Classes from which other classes are derived are called base classes or super classes.
    - A derived class is said to derive, inherit, or extend a base class.