**OOP Part 1 Intro**

**# =================**

**# == What Is OOP ==**

**# =================**

**# - python support Object oriented programming**

**# - OOP is a paradigm or coding style**

**# - paradigm: means structuring program so the methods [functions] and attributes [data]**

**# are bundled into object**

**# - methods: acts as functions that use the information of the object**

**# - python is multi-paradigm programming language [procedural, OOP, functional]**

**# - procedural: structure app like recipe, sets of steps to make the task**

**# - functional: built on the concept of mathematical functions**

**# =============**

**# == Why OOP ==**

**# =============**

**# - OOP allow you to organize your code and make it readable and reusable**

**# - everything in python is an object**

**# => if man is object**

**# - attributes: Name, Age, Address, Phone, Number, Info, ...**

**# - methods: Walking, Eating, Signing, Playing, ...**

**# =============**

**# == Class ==**

**# =============**

**# - is the template for creating objects [object constructor | blueprint]**

**# - class car can make many cars object**

**# - class user can create many users object**

OOP Part 2 Class Syntax and Info

**# ===========================**

**# == Class Syntax and Info ==**

**# ===========================**

**# - [01] class is the blueprint or constructor of the object**

**# - [02] class instantiate means create instance of a class**

**# - [03] instance: object created from class and have their methods and attributes**

**# - [04] class define with keyword "class"**

**# - [05] class name written with pascalCase [upperCamelCase] style**

**# - [06] class may contains methods and attributes**

**# - [07] when creating object python look for the built in \_\_init\_\_ method**

**# - [08] \_\_init\_\_ method called every time when you create object from class**

**# - [09] \_\_init\_\_ method is initialize to the data for the object**

**# - [10] any method with two underscore in the start and end called dunder or magic function**

**# - [11] self refers to the current instance that created from the class and must be first parameter**

**# - [12] self can be named anything**

**# - [13] in python you dont need to call new() keyword to create object**

**# - [14] when using Class any methods inside the class can be calculated together**

**# create class**

**class Member:**

**# self: must be first parameter at the \_\_init\_\_() dunder function**

**def \_\_init\_\_(self):**

**print("New Member Has Been Added")**

**class Student:**

**def \_\_init\_\_(self):**

**print("New Student Has Been Added")**

**Member() # New Member Has Been Added**

**Member() # New Member Has Been Added**

**# show all methods on the class using dir() method and attributes**

**print(dir(Member))**

**# assigm the class to variale**

**student\_one = Student()**

**member\_one = Member()**

**# determine the class of any object using \_\_class\_\_ dunder function**

**print(student\_one.\_\_class\_\_) # <class '\_\_main\_\_.Student'>**

**print(member\_one.\_\_class\_\_) # <class '\_\_main\_\_.Member'>**

OOP Part 3 Instance Attributes & Methods part 1

**# =====================================**

**# == instance attributes and methods ==**

**# =====================================**

**# - self: point to instance that created from class**

**# - instance attributes: that defined inside the constructor**

**# ---------------------------------------------------------------**

**# - instance methods take self parameter which point to instance created from class**

**# - instance method can have more than one parameter like any function**

**# - instance methods can freely access attributes and methods on the same object**

**# - instance methods can access the class itself**

**# ---------------------------------------------------------------**

**class Member:**

**# Define The Constructor**

**def \_\_init\_\_(self, fname, lname):**

**self.first\_name = fname**

**self.last\_name = lname**

**# assign class to a variable**

**member\_one = Member('osama', 'mohamed')**

**# access the class attributes**

**print(member\_one.first\_name) # osama**

**print(member\_one.last\_name) # Mohamed**

OOP Part 4 Instance Attributes & Methods part 2

**# ====================================================**

**# == instance attributes & method attributes part 3 ==**

**# ====================================================**

**class Member:**

**# define the constructor**

**def \_\_init\_\_(self, first\_name, last\_name, gender):**

**self.fname = first\_name**

**self.lname = last\_name**

**self.gender = gender**

**def full\_name(self):**

**return f"{self.fname} {self.lname}"**

**def name\_with\_title(self):**

**if self.gender == 'male' or self.gender == "Male":**

**return f"hello MR {self.fname}"**

**elif self.gender == "female" or self.gender == "Female":**

**return f"hello MISS {self.fname}"**

**else:**

**return f"sorry unkown {self.gender}"**

**def get\_all\_info(self):**

**return f"{self.name\_with\_title()}, your full name is: {self.full\_name()}"**

**member\_one = Member('osama', 'mohamed', 'male')**

**member\_two = Member('mona', 'mohamed', 'female')**

**print(member\_one.full\_name()) # osama mohamed**

**print(member\_one.name\_with\_title()) # hello MR osama**

**print(member\_two.name\_with\_title()) # hello MISS mona**

**print(member\_one.get\_all\_info()) # hello MR osama, your full name is: osama mohamed**

**OOP Part 5 Class Attributes**

**# ======================**

**# == class attributes ==**

**# ======================**

**# - the attributes of class is exist outside the constructor**

**# - to access the class attributes must write the class name then write**

**# the attributes like [Member.note\_allowed\_names]**

**# -**

**class Member:**

**# ======================**

**# == class attributes ==**

**# ======================**

**not\_allowed\_names = ['hell', 'baloot', 'elzero']**

**users\_count = 0**

**# define the constructor**

**def \_\_init\_\_(self, first\_name, last\_name, gender):**

**self.fname = first\_name**

**self.lname = last\_name**

**self.gender = gender**

**# while create new user the constructor will add it to the count**

**Member.users\_count += 1**

**def full\_name(self):**

**if self.fname in Member.not\_allowed\_names:**

**return f"{self.fname}: this name is not allowd to use"**

**else:**

**return f"{self.fname} {self.lname}"**

**def name\_with\_title(self):**

**if self.gender == 'male' or self.gender == 'Male':**

**return f"Hello MR {self.fname}"**

**elif self.gender == 'female' or self.gender == "Female":**

**return f"Hello MISS {self.fname}"**

**else:**

**return f"Hello {self.fname}"**

**def delete\_user(self):**

**Member.users\_count -= 1**

**member\_one = Member("elzero", 'mohamed', 'male') # this called instance**

**member\_two = Member('mona', 'ahmed', 'female') # this called instance**

**member\_three = Member('mona', 'ahmed', 'other') # this called instance**

**print(member\_one.full\_name()) # osama mohamed**

**print(member\_two.name\_with\_title()) # Hello MISS mona**

**print(member\_three.name\_with\_title()) # Hello mona**

**# =================================**

**# == access the class attributes ==**

**# =================================**

**# print the number of users that created**

**print(Member.users\_count) # 3**

**member\_one.delete\_user() # will delete member\_one from users**

**print(Member.users\_count) # 2**

**OOP Part 6 Class Methods & Static Methods**

**# ======================================**

**# == class methods and static methods ==**

**# ======================================**

**# --------------------**

**# --- class method ---**

**# --------------------**

**# - marked with @classmethod decorator to flag it as class method**

**# - it take Cls parameter not self to point to the class not instance**

**# - it does not require creation of class instance**

**# - used when you want to do something with the class itself**

**# -------------------**

**# -- static method --**

**# -------------------**

**# - it takes no parameters**

**# - its bound to the class not instance**

**# - used when doing somthing does not have access to the object or class but related to class**

**# ------------------------------------------------------------------------------------------------**

**class Member:**

**users\_count = 0**

**def \_\_init\_\_(self, first\_name, last\_name, gender):**

**self.fname = first\_name**

**self.lname = last\_name**

**self.gender = gender**

**# while create new instance will add new user**

**Member.users\_count += 1**

**def delete\_user(self):**

**Member.users\_count -= 1**

**def full\_name(self):**

**return f"{self.fname} {self.lname}"**

**# ------------------**

**# -- class method --**

**# ------------------**

**@classmethod**

**def show\_users\_count(cls):**

**print(f"we have {cls.users\_count} users in the system.")**

**# -------------------**

**# -- static method --**

**# -------------------**

**@staticmethod**

**def say\_hello():**

**return "hello from static method"**

**member\_one = Member("osama", 'mohamed', 'male')**

**member\_two = Member("mona", 'mohamed', 'female')**

**member\_three = Member("osama", 'mohamed', 'male')**

**member\_four = Member("osama", 'mohamed', 'male')**

**print(Member.users\_count) # 4**

**member\_one.delete\_user()**

**print(Member.users\_count) # 3**

**Member.show\_users\_count() # we have 3 users in the system.**

**print(Member.say\_hello()) # hello from static method**

OOP Part 7 Magic Method

**# ===================**

**# == magic methods ==**

**# ===================**

**# - everything in python is an object**

**# - \_\_init\_\_ called automatically when instantiating class**

**# - self.\_\_class\_\_ the class to which a class instance belongs**

**# - \_\_str\_\_ gives a human-readable output of the object**

**# - \_\_len\_\_ returns the length of the container**

**# calld when we use the built-in len() function on the object**

**# ----------------------------------------------------------------**

**class Skill:**

**# define the constructor**

**def \_\_init\_\_(self):**

**self.skills = ['html', 'css', 'js']**

**# get human readable about the object**

**def \_\_str\_\_(self):**

**return f"this is my skills {self.skills}"**

**# get the length of the skills that exist**

**def \_\_len\_\_(self):**

**return f"the number of the skills are: {len(self.skills)}"**

**profile = Skill()**

**print(profile.\_\_class\_\_) # <class '\_\_main\_\_.Skill'>**

**myString = 'osama'**

**print(type(myString)) # <class 'str'>**

**print(myString.\_\_class\_\_) # <class 'str'>**

**# ---------------**

**# -- Note That --**

**# ---------------**

**print(str.upper('osama')) # OSAMA**

**print(myString.upper()) # OSAMA**

**# get info about the object**

**# get the address on the memory of the object**

**print(profile) # <\_\_main\_\_.Skill object at 0x000001EBDB79E750>**

**print(profile.skills) # ['html', 'css', 'js']**

**# get human readable about the object using [ \_\_str\_\_ ]**

**print(profile) # this is my skills ['html', 'css', 'js']**

**# determine the number of skills that exist**

**# print(len(profile)) # error**

**print(profile.\_\_len\_\_()) # the number of the skills are: 3**

**# add skill to the skills**

**profile.skills.append('php')**

**profile.skills.append('mysql')**

**print(len(profile.skills)) # 5**

**print(profile.\_\_len\_\_()) # the number of the skills are: 5**

OOP Part 8 inheritance

**# =================**

**# == inheritance ==**

**# =================**

**class Food: # base class**

**def \_\_init\_\_(self, name):**

**self.name = name**

**print(f'{self.name}: is created from base class')**

**def eat(self):**

**print('eat method from base class')**

**# inherit all methods from other class**

**# using (class\_name) 'inheritance methods'**

**# inherit the attributes that exist on the constructor**

**class Apple(Food): # derived class**

**def \_\_init\_\_(self, name):**

**# inherit the attributes of constructor**

**# create instance from base class**

**Food.\_\_init\_\_(self, name)**

**print(f"{self.name}: is created from derived class")**

**# the following is other way to inherit the constructor**

**# the two ways behave the same job**

**# super().\_\_init\_\_(name)**

**food\_one = Food('Pizza') # Pizza :is created from derviced class**

**print("="\*33)**

**print("="\*33)**

**print("="\*33)**

**print("="\*33)**

**food\_two = Apple('pizza') # apple is created from derived class**

**print("="\*33)**

**# get methods from base class, and use it**

**food\_two.eat() # eat method from base class**