# Oops

### Object-Oriented Programming (OOPS)

- Object-oriented programming (OOP) is a programming style that organizes code around objects, rather than functions and logic.
- Main Concepts of Object-Oriented Programming (OOPs)
- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction

### Ques. What is the use of self in Python?

- The Self parameter is a reference to the current instance of the class, we can access the attributes and methods of the class in python.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("Mohit", 36)
p1.myfunc()     # Output:- Hello my name is Mohit
```

- We can give any name in place of self but first parameter is compulsory.

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

### Ques. What is pass in Python?

- The pass keyword represents a null operation in Python.

- Without the pass statement in the following code, we may run into some errors during code execution.

```
def myEmptyFunc():
   # do nothing
   pass
myEmptyFunc()    # nothing happens
## Without the pass keyword
# File "<stdin>", line 3
# IndentationError: expected an indented block
```

### Ques. What is break, continue and pass in Python?

- **Break:-** The break statement terminates the loop immediately and the control flows to the statement after the body of the loop.
- **Continue:-** The continue statement terminates the current iteration of the statement, skips the rest of the code in the current iteration and the control flows to the next iteration of the loop.
- **Pass:-** As explained above, the pass keyword in Python is generally used to fill up empty blocks and is similar to an empty statement represented by a semi-colon in languages such as Java, C++, Javascript, etc.

```
pat = [1, 3, 2, 1, 2, 3, 1, 0, 1, 3]
for p in pat:
   pass
   if (p == 0):
       current = p
       break
   elif (p % 2 == 0):
       continue
   print(p)    # output => 1 3 1 3 1
print(current)    # output => 0
```

### Ques. What is issubclass()?

- we can use Python built-in function issubclass(). This function returns True if the given class is the subclass of the specified class. Otherwise, it returns False.

```
Syntax:- issubclass(class, classinfo)
```

### Ques. What is str and repr?

- The **str** method also known as a "dunder" method (double underscore method), that defines the string representation of an object. It's used to return a **human-readable** string when the built-in functions str() or print() are called on an instance of a class.

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
    def __str__(self):
        return f'"{self.title}" by {self.author}'

book = Book("C++", "E balaguswami")
print(book) # Output: "C++" by E balaguswami
print(str(book))   # Output: '"C++" by E balaguswami'


# Without a __str__ method, Python uses the default representation, like
# <__main__.Person object at 0x000001>.
```

- The repr() method returns a string containing a printable representation of an object. The repr() function calls the underlying **repr**() function of the object.
- **repr** method returns a string representation of an object that is **machine-readable**.

```
import datetime
today = datetime.datetime.now()
print(str(today))   # 2025-05-11 10:47:08.923663 (Readable end user format)
print(repr(today))  # datetime.datetime(2025, 5, 11, 10, 47, 8, 923663) (official
developmrnt format)
```

### Ques. What is Abstract Method?

- To define an abstract **method** we use the **@abstractmethod** decorator of the abc module.

```
from abc import ABC, abstractmethod
class DemoAbstractClass(ABC):
    @abstractmethod
    def abstract_method_name(self):
        Pass
```

### Ques. What is Concrete Method?

- A concreate method is a method whose action is defined in the abstract class itself.

```
from abc import ABC, abstractmethod
class Father(ABC):
    @abstractmethod
    def disp(self):
        pass                                # method without body
    def show(self):
        print("concrete method")            # concrete Method / method with body
```

```
class Father:
    def __init__(self, fname, lname):
        self.first_name = fname
        self.last_name = lname

    def show_data(self):
        print(self.first_name)

obj = Father('mohit','saxena')
obj.show_data()
```

## Ques. Difference between method and function?

#### Function

- A function is a block of code that performs a specific task and can be called independently from anywhere in your program.
- It is defined using the def keyword.
- It can take arguments (input values) and return values (output values).

```
def add_numbers(x, y):
    return x + y

result = add_numbers(5, 3)
print(result)
```

#### Method

- A method is a function that is associated with an object or a class.
- It is defined within a class and operates on the data or attributes of that class.
- It is called using the dot notation on an object of the class.

```
class Dog:
    def bark(self):
        print("Woof!")

my_dog = Dog()
my_dog.bark()
```

## What is Python Enumeration?

- In Python, an enumeration (or "enum") is a class that defines a set of symbolic names (constants) that are bound to unique, constant values. Enums are created using the enum module and are typically used to represent groups of related constants, making code more readable and maintainable.

```python
from enum import Enum

class TrafficLight(Enum):
    RED = 1
    YELLOW = 2
    GREEN = 3

# Accessing enum members by name:
print(TrafficLight.RED.name)  # Output: RED
print(TrafficLight.GREEN.value) # Output: 3

# Accessing enum members by value:
print(TrafficLight(1).name) # Output: RED
print(TrafficLight(3).value) # Output: 3
```

# class

### Ques. What is Class?

- Class is a template/blueprint/prototype for creating objects.
- class is a collection of attributes and method.
- The class is a collection of objects.
- It is a logical entity that has some specific attributes and methods.
- To define a class in Python, you can use the class keyword, followed by the class name and a colon. Inside the class, an **init** method has to be defined with def. This is the initializer that you can later use to instantiate objects. It's similar to a constructor in Java. **init** must always be present! It takes one argument: self, which refers to the object itself. Inside the method, the pass keyword is used as of now, because Python expects you to type something there.
- **Example:-** Email is a class and headding, particiant, attachment is object

#### Define a class

- We use the class keyword followed by the class name and a colon. The following example defines a Person class:

```
class Person:
    pass
```

- If the class name contains multiple words, you use the **CamelCase** format, for **example:- SalesEmployee**.
- When printing out the person object, you'll see its name and memory address:

```
class Person:
    pass

print(person)

Output:- <__main__.Person object at 0x000001C46D1C47F0>
```

- To get an identity of an object, you use the id() function. For example:

```
print(id(person)) # 1943155787760
```

### Python get Class Variables/attributes

- Get the values of **class variables**

```
# using calss name
class Student:
    name = 'mohit saxena'
    roll_no = '12845678'

print(Student.name)     # mohit saxena
print(Student.roll_no)  # 12845678


# using **getattr()** function:- The getattr() function accepts an object and a
variable name. It returns the value of the class variable.
name = getattr(Student, 'name')
rollno = getattr(Student, 'roll_no')

print(name)     # mohit saxena
print(rollno)   # 12845678

# using object:- Accessing through object instantiation.
obj= Student()
print(obj.name) # mohit saxena
```

- If you access a class variable that doesn't exist, you'll get an AttributeError exception.

### Set/Change values for class variables

- To set a value for a class variable, you use the dot notation:

```
# set the value using class
class Student:
    name = 'mohit saxena'
    roll_no = 12845678

#  Changing value using Class Name
Student.roll_no = 10
print(Student.roll_no)    # output:- 10

# using setattr() built-in function
setattr(Student, 'roll_no', 10)
print(Student.roll_no)  # output:- 10

# using object
obj= Student()
obj.name = 'saxena mohit'
print(obj.name) # Output:- saxena mohit
```

### Delete class variables

```
class Student:
    name = 'mohit saxena'
    roll_no = '12845678'

print(Student.name)     # mohit saxena
print(Student.roll_no)  # 12845678

# using **delattr()** function:
delattr(Student, 'roll_no') # Output:- AttributeError: type object 'Student' has
no attribute 'roll_no'

# using del keyword
del Student.roll_no
print(Student.roll_no)  # Output:- AttributeError: type object 'Student' has no
attribute 'roll_no'
```

Ques. What is **init** Method?

- **init** is a constructor method in Python and is automatically called to allocate memory when a new object/instance is created.
- All classes have a function called **init**() function, which is always excuted when the object is being initiated.
- Python always calls init function whatever you create them or not.

```
class Student:
   def __init__(self, fname, lname):
       self.firstname = fname
       self.lastname = lname
       print("My name is " + self.firstname + " " + self.lastname)
# creating a new object
stu1 = Student("Mohit", "Saxena") #output:- My name is Mohit Saxena
```

- Since Python is a dynamic language, you can add a class variable to a class at runtime after you have created it.

```
class HtmlDocument:
    extension = 'html'
    version = '5'

HtmlDocument.media_type = 'text/html'
print(HtmlDocument.media_type)

Output:- text/html
```

- **Example 1**

```
class Person:
  def __init__(self, name, age):
    self.f_name = name
    self.age = age

p1 = Person("John", 36)

print(p1.f_name) # John
print(p1.age)    #  36
```

- **Example 2**

```
class Dog:

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

    def bark(self):
        print("bark bark!")

    def doginfo(self):
        print(self.name + " is " + str(self.age) + " year(s) old.")

ozzy = Dog("Ozzy", 2)
skippy = Dog("Skippy", 12)
filou = Dog("Filou", 8)

ozzy.bark()
skippy.doginfo()
filou.doginfo()

Output:-
bark bark!
Skippy is 12 year(s) old.
Filou is 8 year(s) old.
```

# Object

### Ques. What is Object?

- An object is a container that contains data and functionality.
- The object is an entity that has state and behavior. It may be any real-world object like the mouse, keyboard, chair, table, pen, etc.
- Before creating objects, you define a class first. And from the class, you can create one or more objects. The objects of a class are also called **instances** of a class.
- **For example:** if you have an employee class, then it should contain an attribute and method, i.e. an email id, name, age, salary, etc.

```
class Person:
    pass

person = Person()
print(person)
```

```
class car:
    a = "mohit"
    def __init__(self,modelname, year):
        self.modelname = modelname
        self.year = year
    def display(self):
        print(self.modelname,self.year)

c1 = car("Toyota", 2016)
print(c1.a)       # Accessing Object's variables    Output:- mohit
c1.display()      # Accessing Object's functions    Output:- Toyota 2016
```

### Modifying Object's properties

```
class Scaler:
  a = 10

# Declaring an object
obj1 = Scaler()
print(obj1.a)   # output:- 10

#Modifying value
obj1.a = 200
print("After modifying the object properties")  # output:- After modifying the
object properties
print(obj1.a)   # output:- 200
```

- Here, the **self** is used as a reference variable, which refers to the current class object. It is always the first argument in the function definition. However, using self is optional in the function call.
- The **self-parameter** refers to the current instance of the class and accesses the class variables. We can use anything instead of self, but it must be the first parameter of any function which belongs to the class

### Ques. Delete the Object?

- We can **delete the properties of the object** or object itself by using the **del** keyword.

```
class Student:
    def __init__(self,name):
        self.name = name

# delete the properties of the object
obj = Student("mohit")
print(obj.__dict__)   # Output:- {'name': 'mohit'}
del obj.name
print(obj.__dict__)   # Output:- {}


# Deleting the object itself by del keyword
obj = Student("mohit")
print(obj.__dict__)   # Output:- {'name': 'mohit'}
del obj
print(obj.__dict__)   # Output:- ERROR!
```

### Ques. Counting the Number of objects of a Class?

```
class Employee:
    count = 0
    def __init__(self):
        Employee.count = Employee.count + 1


# creating objects
e1 = Employee()
e2 = Employee()
e2 = Employee()
print("The number of Employee:", Employee.count)

Output:- The number of employee: 3
```

### Ques. other example:

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def get_avg(self):
        sum = 0
        lgh = len(self.marks)
        for val in self.marks:
            sum = sum+val
            avg = sum/lgh
        print("my name is",self.name,"and my marks is", avg)
#  + self.name + " and my marks is " + avg
# creating a new object
stu1 = Student("Mohit", [97,98,96])
stu1.get_avg()

stu1.name = "saxena"
stu1.get_avg()

Output:-
my name is Mohit and my marks is 97.0
my name is saxena and my marks is 97.0
```

Ques.

```
class Account:
    def __init__(self, acc_no, bal):
        self.acc_no = acc_no
        self.bal = bal

    def debit(self, amount):
        self.bal -= amount
        print("debited amount is ", amount, "my bal is", self.finalAmount())

    def credit(self,amount):
        self.bal += amount
        print("credit amount is ", amount, "my bal is", self.finalAmount())

    def finalAmount(self):
        return self.bal

# creating a new object
stu1 = Account(564654,10000)
stu1.debit(900)
stu1.credit(800)

Output:-
```

```
debited amount is  900 my bal is 9100
credit amount is  800 my bal is 9900
```