{ OOP In Python }

- What are functions?

A function is a reuseable block of code that perform a specific task when called. They are two types of functions:

- Built in function (e.g print(), len(), input())
- User-defined functions (functions you define yourself)

Functions Vs Methods

Functions: Pre-defined or user-defined blocks of code that can be called independently

print("Yasir") # built is a built-in Function

Methods: Functions associated with objects / data types.

Example: list.append(), str.upper(), dict.get()

User-defined function Example

```
def greet():
    print("My name is Yasir Abdullah")
    print("I am a Software Engineer")
greet() # Calling the function
```

Variable Scope in Functions

```
def addiction():
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the Second number: "))
    print(f"The sum is: {num1 + num2}")
addiction()
```

Using Multiple Functions

```
def addiction():
    num1 = int(input("Enter the 1st no: "))
    num2 = int(input("Enter the 2nd no: "))
    print(f"The sum is: {num1 + num2}")

def substraction():
    num1 = int(input("Enter the 1st no: "))
    num2 = int(input("Enter the 2nd no: "))
    print(f"The difference is: {num1 - num2}")
addiction()
substraction()
```

Scope Example with Same Variable Name

```
def greet():
    name = Input("Enter your name: ")
    print("Inside function, name is:", name)
name = "Abdullah"
greet()
print("outside function, name is:", name)
```

Types of Functions

Based on Parameters and Return Values:

- No parameters, no return
- With parameters, no return
- No parameters, with return
- With parameters, with return

Function with parameters (No Return)

```
def addiction(n1, n2):
    total = n1 + n2
    print(f"Sum is: {total}")
addiction(2, 4)
```

Function with return value

```
def addiction (n1, n2):
    total = n1 + n2
    return total
result = addiction (2, 4)
print( f" Direct return: {addiction (2, 4)}")
```

Function without Return

```
def addiction (n1, n2):
    total = n1 + n2
    print( f" Printed result: {total}")
x = addiction (2, 4)
print( f" Returned value : {x}")
```

When to use , Return Vs Print

use return when you need to use the result later

use print when you just want to display something immediately

Positional Vs Named Arguments

```
def total_marks ( maths, science, english, urdu, history):
    total = maths + science + english + urdu + history
total_marks ( 84, 65, 67, 54, 75)
total_marks( maths,= 100, urdue = 14, english = 13, science=29, his=32)
```

Default Arguments

```
def add_default(n1, n2):
    print(n1 + n2)
add_default( 11, 32)
```

* add_default ( ) * Error: missing required positional Arg n1

*args (Variable Length Positional Argu)

```
def add_fixed(n1, n2, n3):
    total = n1 + n2 + n3
    print(total)
add_fixed(23, 243, 2)
```

Using *args to accept variable no. of arguments

```
def add_varargs(*args):
    print(sum(args))
add_varargs(23, 243, 2)
add_varargs(1, 3, 5, 3, 6, 3, 3, 5)
add_varagrs(2, 4)
```

**Kwargs (Variable-length Keyword Argu)

```
def add_kwargs(**kwargs):
    print(kwargs)
add_kwargs(name = "Yasir", age = "24", gender = "male")
```

```
def display_kwargs(**kwargs):
    for k, v in kwargs.items():
        print(k, v)
display_kwargs(name = "yasir", age = "24", gender = "male")
```

Scope In Python

Local vs Global

Global Scope: Define outside any function

```
a = 15
def num():
    a = 30
    print(a)
print(a)
num(a)
Print(a)
```

Using global Keyword

```
a = 15
def num()
    global a
    a = 30
    print(a)
print(a)
num()
print(a)
```

Object Oriented Programing (OOP)

OOP is a way to structure your code using classes & objects
It helps manage complexity in large codebases

Key Features of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Creating a Class

```
class Student:
    def info(self):
        print(f"Name = {self.name}")
        print(f"Age = {self.age}")
    def set_info(self):
        self.name = input("Enter your name: ")
        self.age = Input ("Enter your age = ")
S1 = Student()
S2 = Student()
S1.set_info()
S2.set_info()
S1.info()
S2.info()
```

Better Approach: Use __init__() Constructor

__int__() is called automatically when an object is created. It helps initialize values during object creation.

```
class Student():
    def __init__(self):
        self.name = input("Enter your name: ")
    def info(self):
        print(f"Name = {self.name}")
S1 = Student()
S1.info()
S2 = Student()
S2.info()
```

Modules In Python

A module in python is a file containing python definations and functions. You can reuse the fonctions in another python file by importing the module

Concept: if __name__ = "__main__"

This bolo block runs only when the file is executed directly. It does not run when the file is imported as a module into another script. This is useful for testing or demo purposes.

Inheritance In Python

It means u can inherit the properity of class a in class b.

Example

```
class Car():
    def __init__(self, color: str, type: str, milage: float):
```

```
        self.color = color
        Self.type = type
        self.milage = milage
def base_info(self):
        print(self.color)
        print(self.type)
        print(self.milage)
class Audi(Car):
    def __init__(self):
        print("hello world")
cl= Audi()
cl.color = "Red"
cl.milage = 23.4
cl.type = "Pebrol"
cl.base_info()
```

Multiple Inheritance

Multiple Inheritance means a class can inherit from more than one parent class. This allows the child class to access attributes and methods from all parent classes.

Multilevel Inheritance

Multilevel Inheritance means a class is derived from a class which is alway/already derived from another class. It forms a "chain of inheritance" like grandparent → par → child

Syntax

```
Class A:
    Pass
Class B(A):
    pass
Class C(B):
    Pass
```

Hierarchical Inheritance

It occurs when multiple child classes inherit from a single parent class.

Syntax:

```
class Parent:
    pass
class Child1(Parent):
    pass
class Child2(Parent):
    pass
```

Hybrid Inheritance

It is a combination of two or more types of inheritance in a single program.

It's called "hybrid" because it mixes different inheritance str

Encapsulation In Python:

It is the concept of wrapping data (variables) and code (methods) into a single unit _a class

It restricts direct access to some of an object components, which is a way of preventing accidental modification.

What are Access Modifiers?

Access Modifier define the numbers of a class can be accessed.

Public: Accessible from anywhere

Protected: Indicated by single underscore _var but accessible

Private: Indicated by double underscore __var name mangled to prevent direct access.

Method Overriding

It means providing a new definition for a method in a subclass that is already defined in the parent class. It allows different behaviours for the same method name

What are dunder Methods

Dunder (Double Underscore) methods are special method in python that start and end with double underscore (--)
Example: --init--, --str--, --len--, --add-- etc.
These methods allow us to define custom behaviour for built-in operations like object creation, printing and addition etc.

Abstraction In Python

Abstraction is a concept in oop. where we hide complex implementation details and show only the essential features to the users. It helps in reducing complexity and increases code reuseability and readability.

Real-life Example

- Think of car, You can drive it without knowing how the engine works.
- The internal complexity is hidden, you only interact with car interface

In python, abstraction is achieved used.

- Abstract Base Class (ABC)
- @abstractmethod decorator

Why we use abstraction?

- To define a common interface for all subclasses

- To enforce certain methods in child classes
- To hide internal details & show only necessary parts

Decorators In Python

Decorators in python are functions that modify the behaviour of another function without changing its structure. They allows us to 'wrap' another function to add extra functionality before or after the orignal function runs.

Where Can we use decorators?

- Logging Function execution
- Measuring execution time
- Access control / authentication
- Caching or memorization
- Pre/Post-processing around Functions.

Syntax

A decorator is usually applied using the '@decorator name' above the Function defination

Generators In Python

Generators are functions that return an interator and allow us to iterate through a sequence of values, one at a time, using the 'Yield' keyword instead of 'return'. Unlike lists, generators don't store the entire store sequence in memory. they generate values on the fly.

Why / where do we use Generators?

- To Save memory when dealing with large data streams
- To Improve performance with lazy evaluation
- To create custom Iterators easily

Generator Vs Normal Function

- Normal Function → returns result using 'return'
- Generator function → uses 'Yield' to return values one by one on demand

Static Method

Static methods in Pypthon are methods that do not requires access to the instance (self) or class (cls). These methods are bound to a class, not to its object. They are defined using the @staticmethod decorator

Why do we use static method?

When the method does not need to access or modify instance/class attributes. To perform utility operations related to the Class. Helps in keeping related logic within the class without using its state.

Class Method

A class method is a method that is bound to the class and not the instance of class.

It takes 'cls' (the class itself) as its first parameter instead of 'self'.

It is defined using @classmethod decorator

Why do we use Class Methods / Factory Methods?

To create multiple ways of initializing objects

When you want to perform some operations that relate to the class as a whole.

Useful for alternative constructors like creating objects from files, strings etc

Serialization / Deserialization

What is Serialization?

Serialization means converting a python object (like a dictionary, list, class object) into a byte stream that can be saved to a file or sent over a network

What is deserialization?

Deserialization is the reverse process - converting a byte stream back into the original python objects.

Why use Serialization?

- To save program state to disk
- To send python objects between systems.
- To store trained ML model
- To cache results for faster processing

Example:

Imagine you trained a Machine Learning Model. You want to save it and use it later

You serialize (Pickle) it

Regular Expressions

A regular expression is a sequence of characters that forms a search pattern. It is used for string searching and manipulation, such as validation, searching, extracting, or replacing text.

Uses Cases of Regx in Python

- Validation email, phone no, password, Data cleaning, Tokenizing text, log parsing, web scraping, Text replacement