



Object-Oriented Programming (OOP)



About the Author



- Created By: Mohammad Salman
- Experience: 19 Years +
- Designation: Corporate Trainer .NET



Icons Used



Questions



Tools



Hands-on Exercise



Coding Standards



Questions?



Reference



Try it Out



Informative
Slide



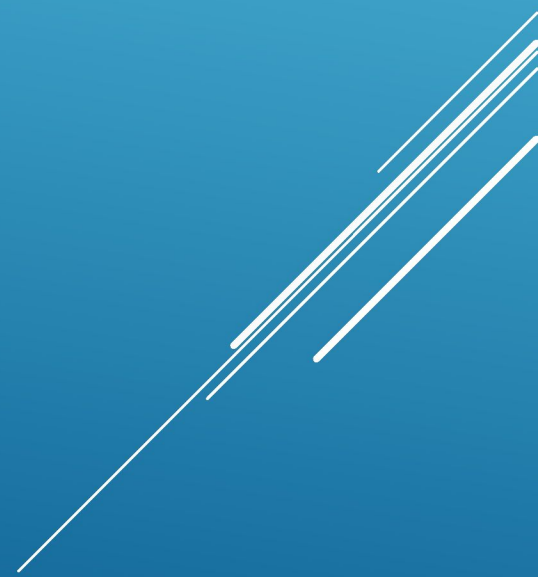
Mandatory
Slide



Welcome Break

PYTHON OBJECT-ORIENTED PROGRAMMING (OOP)

- ▶ Learn Python Object-Oriented Programming (OOP)
By: Mohd Salman

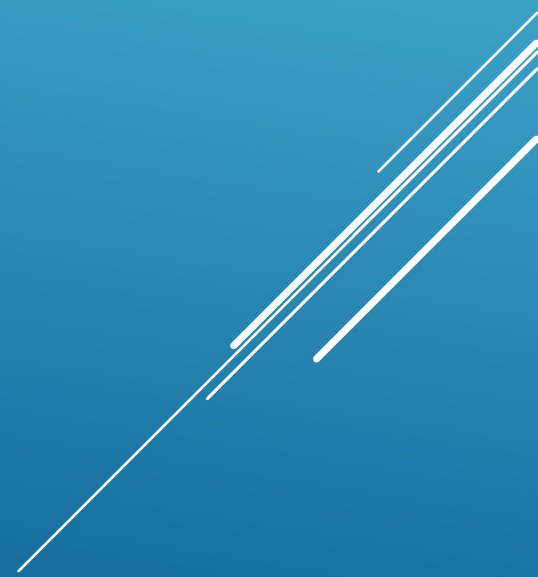


- 1. What is OOP?
- 2. Defining a Class
- 3. Creating Objects
- 4. Methods in Classes
- 5. Object Constructors (__init__)
- 6. Hands-on Labs
- 7. Assessment Quiz

AGENDA

- ▶ Object-Oriented Programming (OOP) is a programming paradigm based on the concept of 'objects'.
- ▶ Key Concepts:
 - ▶ - Class
 - ▶ - Object
 - ▶ - Method
 - ▶ - Encapsulation
 - ▶ - Inheritance
 - ▶ - Polymorphism
- ▶ Python fully supports OOP principles.

WHAT IS OBJECT-ORIENTED PROGRAMMING?



A class is a blueprint for creating objects.

Syntax:

```
class ClassName:  
    # class attributes and methods  
    pass
```

Example:

```
class Car:  
    def start(self):  
        print('Car started')
```

DEFINING A CLASS



- Objects are instances of a class.

Example:

```
car1 = Car()  
car1.start()
```

Output:

Car started

CREATING OBJECTS

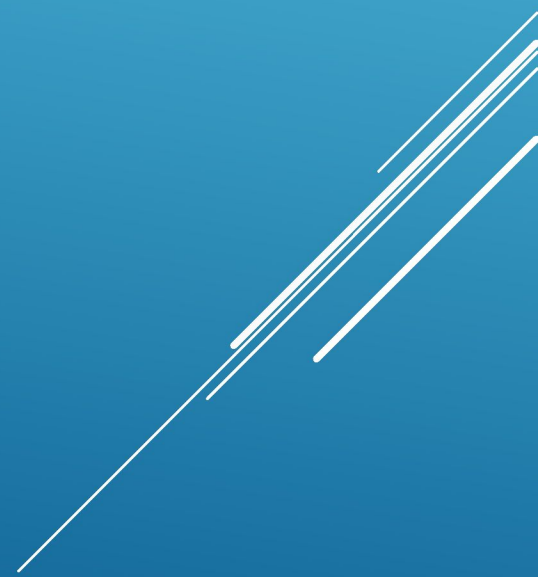


- ▶ Methods are functions defined inside a class.
- ▶ They describe the behavior of objects.

▶ Example:

```
class Dog:  
    def bark(self):  
        print('Woof! Woof!')  
  
d = Dog()  
d.bark()
```

METHODS IN CLASSES



- ▶ The `__init__()` method automatically runs when an object is created.
- ▶ It is used to initialize object attributes.

Example:

```
class Student:
```

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

```
s1 = Student('Alice', 22)
```

```
print(s1.name, s1.age)
```

OBJECT CONSTRUCTOR (`__INIT__`)



Example: Bank Account

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print('Deposited:', amount)

    def display(self):
        print(f'Owner: {self.owner}, Balance: {self.balance}')

acc = BankAccount('John', 1000)
acc.deposit(500)
acc.display()
```

REAL-WORLD EXAMPLE

- ▶ Object-Oriented Programming (OOP) is a paradigm based on the concept of 'objects'.
- ▶ It focuses on data and the methods that operate on that data.

- ▶ Key Advantages:
 - ▶ - Code Reusability
 - ▶ - Better Maintainability
 - ▶ - Scalability and Modularity

KEY ADVANTAGES OF OOP

- ▶ A class is a blueprint for creating objects.

- ▶ Example:

- ▶ `class Person:`

- ▶ `def __init__(self, name, age):`

- ▶ `self.name = name`

- ▶ `self.age = age`

- ▶ `p1 = Person('Sam', 25)`

- ▶ `print(p1.name, p1.age)`

DEFINING CLASSES AND CREATING OBJECTS



- ▶ Methods define behaviors of an object.
- ▶ `__init__()` is a special constructor method used to initialize objects.
- ▶ Example:
- ▶ class Employee:

```
def __init__(self, name, salary):  
    self.name = name  
    self.salary = salary  
  
def display(self):  
    print(f'Employee: {self.name}, Salary:  
{self.salary}')
```

METHODS AND CONSTRUCTORS

- ▶ Inheritance allows one class to derive properties and methods from another.

- ▶ Example:

```
class Animal:
```

```
    def speak(self):
```

```
        print('Animal speaks')
```

```
class Dog(Animal):
```

```
    def speak(self):
```

```
        print('Dog barks')
```

```
obj = Dog()
```

```
obj.speak()
```

INHERITANCE



- ▶ Encapsulation restricts direct access to object data.
- ▶ We use private attributes and getter/setter methods.

▶ Example:

```
class Bank:
```

```
    def __init__(self, balance):  
        self.__balance = balance
```

```
    def get_balance(self):  
        return self.__balance
```

```
acc = Bank(1000)
```

```
print(acc.get_balance())
```

ENCAPSULATION



- ▶ Polymorphism allows methods to have different implementations based on the object.

- ▶ Example:

```
class Bird:
```

```
    def fly(self):
```

```
        print('Bird is flying')
```

```
class Airplane:
```

```
    def fly(self):
```

```
        print('Airplane is flying')
```

```
for obj in [Bird(), Airplane()]:
```

```
    obj.fly()
```

POLYMORPHISM



class Account:

```
def __init__(self, owner, balance=0):
```

```
    self.owner = owner
```

```
    self.__balance = balance
```

```
def deposit(self, amount):
```

```
    self.__balance += amount
```

```
def withdraw(self, amount):
```

```
    if amount <= self.__balance:
```

```
        self.__balance -= amount
```

```
    else:
```

```
        print('Insufficient funds')
```

```
def display(self):
```

```
    print(f'Owner: {self.owner}, Balance: {self.__balance}')
```

ADVANCED EXAMPLE: BANKING SYSTEM

- ▶ - OOP enhances code modularity and reusability.
- ▶ - Classes define the structure; objects bring them to life.
- ▶ - Inheritance promotes hierarchy and shared behavior.
- ▶ - Encapsulation ensures data protection.
- ▶ - Polymorphism enables flexible and scalable design.

KEY TAKEAWAYS

- ▶ 1. What is a class in Python?
- ▶ 2. What is the purpose of the `__init__` method?
- ▶ 3. How does inheritance improve code reuse?
- ▶ 4. Give an example of encapsulation.
- ▶ 5. What is polymorphism? Provide a short example.

ASSESSMENT QUIZ

- ▶ A class is a blueprint or template for creating objects.
- ▶ It defines the attributes (data) and methods (functions) that describe the behavior of an object.
- ▶ You can think of a class as a blueprint, and an object as the actual thing built from that blueprint

THANK YOU!

Several white diagonal lines of varying lengths and thicknesses are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.