

Python Programming

Unit 05 – Lecture 01: OOP Basics, Classes, Objects, Constructors

Tofik Ali

School of Computer Science, UPES Dehradun

February 14, 2026

Repository: <https://github.com/tali7c/Python-Programming>

Quick Links

Core Concepts

Demo

Interactive

Summary

Agenda

1 Core Concepts

2 Demo

3 Interactive

4 Summary

Learning Outcomes

- Explain OOP concepts (class, object, encapsulation)

Learning Outcomes

- Explain OOP concepts (class, object, encapsulation)
- Define classes and create objects in Python

Learning Outcomes

- Explain OOP concepts (class, object, encapsulation)
- Define classes and create objects in Python
- Use constructors (`__init__`) and instance methods

Learning Outcomes

- Explain OOP concepts (class, object, encapsulation)
- Define classes and create objects in Python
- Use constructors (`__init__`) and instance methods
- Use special methods like `__str__`

Learning Outcomes

- Explain OOP concepts (class, object, encapsulation)
- Define classes and create objects in Python
- Use constructors (`__init__`) and instance methods
- Use special methods like `__str__`
- Distinguish class variables and instance variables

Why OOP?

- Models real-world entities (Student, Account, Car)

Why OOP?

- Models real-world entities (Student, Account, Car)
- Groups data + behavior together

Why OOP?

- Models real-world entities (Student, Account, Car)
- Groups data + behavior together
- Improves reusability and maintainability

Why OOP?

- Models real-world entities (Student, Account, Car)
- Groups data + behavior together
- Improves reusability and maintainability
- Supports inheritance and polymorphism

Class and Object

- **Class:** blueprint (defines attributes and methods)

Class and Object

- **Class:** blueprint (defines attributes and methods)
- **Object:** instance created from a class

Defining a Class

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

    def display(self):
        print(self.name, self.sapid)
```

Constructor: `__init__`

- Called automatically when you create an object

```
s1 = Student("Asha", "5001")
```

Constructor: `__init__`

- Called automatically when you create an object
- Initializes instance variables

```
s1 = Student("Asha", "5001")
```

Special Methods (`__str__`)

- `__str__` controls how an object prints

```
class Student:
    def __str__(self):
        return f"{self.name} ({self.sapid})"
```

Class vs Instance Variables

- Class variable: shared by all objects

```
class Student:
    college = "UPES" # class variable

    def __init__(self, name):
        self.name = name # instance variable
```

Class vs Instance Variables

- Class variable: shared by all objects
- Instance variable: unique per object (`self.x`)

```
class Student:
    college = "UPES" # class variable

    def __init__(self, name):
        self.name = name # instance variable
```

Demo: Student Class

- File: `demo/student_class_demo.py`

Demo: Student Class

- File: `demo/student_class_demo.py`
- Creates multiple objects and shows:

Demo: Student Class

- File: `demo/student_class_demo.py`
- Creates multiple objects and shows:
 - constructor usage

Demo: Student Class

- File: `demo/student_class_demo.py`
- Creates multiple objects and shows:
 - constructor usage
 - instance methods

Demo: Student Class

- File: `demo/student_class_demo.py`
- Creates multiple objects and shows:
 - constructor usage
 - instance methods
 - class variable behavior

Checkpoint 1

Question: What is the purpose of `self` in Python methods?

Checkpoint 2

Question: If `college` is a class variable, how many copies exist for 100 students?

Think-Pair-Share

Discuss:

- What attributes and methods should a `BankAccount` class have?

Key Takeaways

- Classes model data + behavior

Key Takeaways

- Classes model data + behavior
- `__init__` initializes objects

Key Takeaways

- Classes model data + behavior
- `__init__` initializes objects
- Special methods like `__str__` improve usability

Key Takeaways

- Classes model data + behavior
- `__init__` initializes objects
- Special methods like `__str__` improve usability
- Class variables are shared; instance variables are per object

Exit Question

Write the name of the special method used as a constructor in Python.