coding temple

Software Engineering

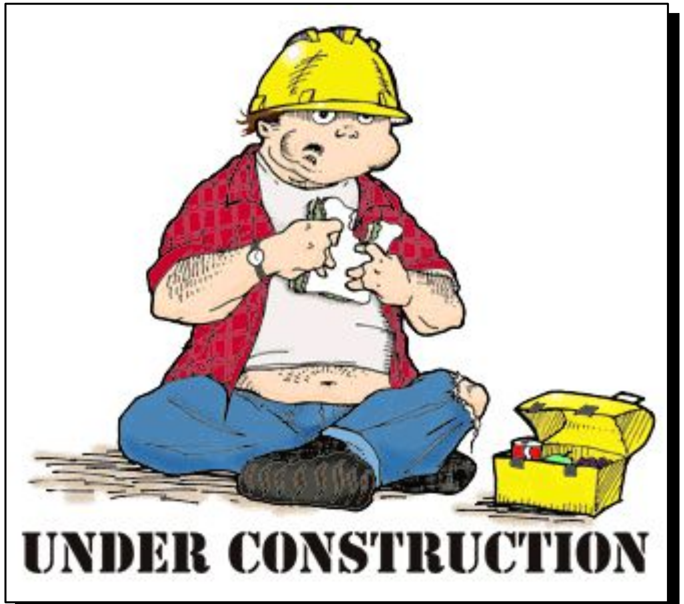
Module 4: Python Object-Oriented Programming (OOP)

OOP Fundamentals

Future Python city architects, get ready for an exhilarating adventure in Python's Object-Oriented Programming (OOP) to design a digital city by covering these topics:

- Classes and objects
- Python's data structures – lists, sets, dictionaries, and tuples
- File handling for efficient storage, and accessibility
- Error handling and modular construction for organized digital metropolis
- Practical exercises to build components like vehicle registration systems, solidifying Python OOP skills

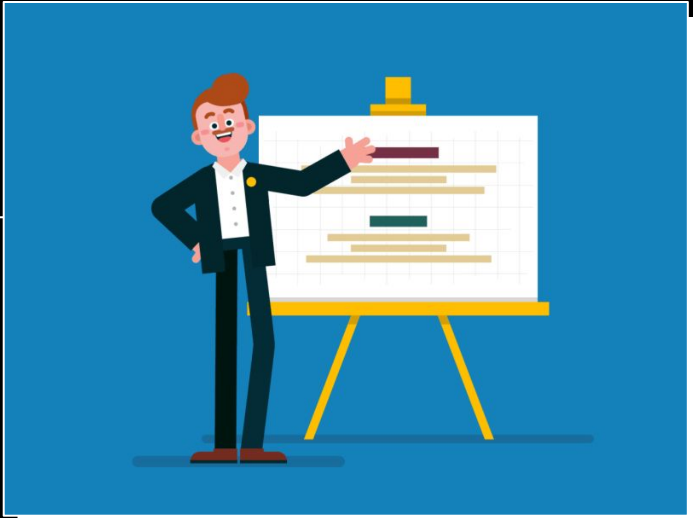
Gear up, future Python experts, let's start constructing! 🏗️🐍💻



Learning objectives

By the end of this lesson you should be able to:

- Apply the concepts of Object-Oriented Programming to create and manage Python classes and objects.
- Comprehend and implement constructors (__init__) in Python for the lifecycle of an object.
- Utilize methods within Python classes to effectively manage data within objects.
- Design and execute Python programs in real-world scenarios that incorporate fundamental data structures such as lists, sets, dictionaries, and tuples.

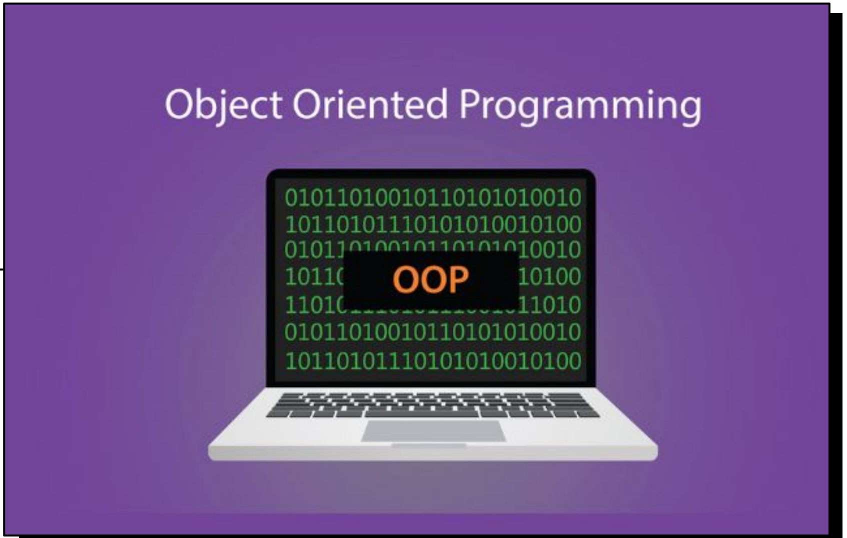


Understanding the Basics of OOP

OOP is considered the process of constructing a digital landscape with interconnected elements. Much like how buildings contribute to a city's overall functionality and structure, objects in OOP collaborate to enhance the efficiency, modularity, and maintainability of software systems.

Why OOP?

OOP, like a well-planned city, brings structure and clarity to your code. It offers the following benefits:

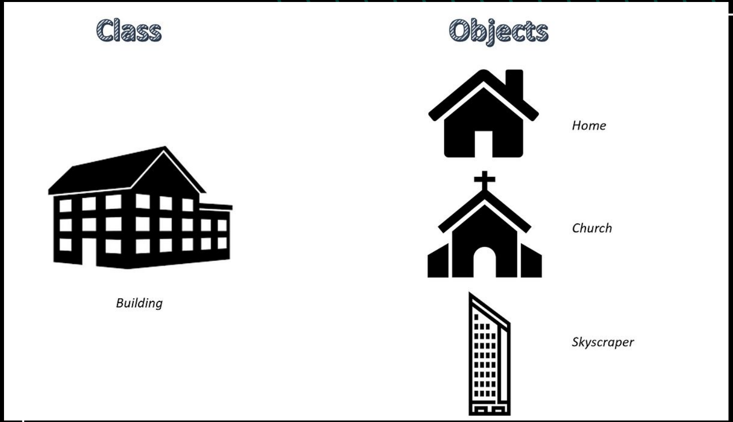


What are the Classes in OOP?

A class in Python defines the structure and behavior of an object.

What are objects?

An object is a specific instance of a class. It possesses the characteristics and behaviors defined by the class.



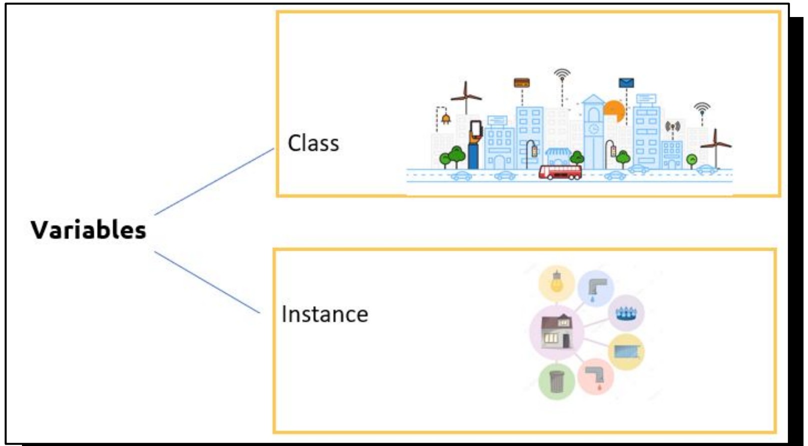
Python Classes and Objects

Remember! A class is a user-defined blueprint or prototype from which objects are created. It represents a set of attributes and methods that are common to all objects of one type.

Attributes: The Infrastructure of Our Code-City

Attributes are similar to the infrastructure of a building in our city. There are two types:

- **Class Variables:** These are like the public utilities in a city and are shared by all instances of a class.
- **Instance Variables:** These are private amenities in an apartment and are specific to an object.



Methods: The Services in Our Code-City

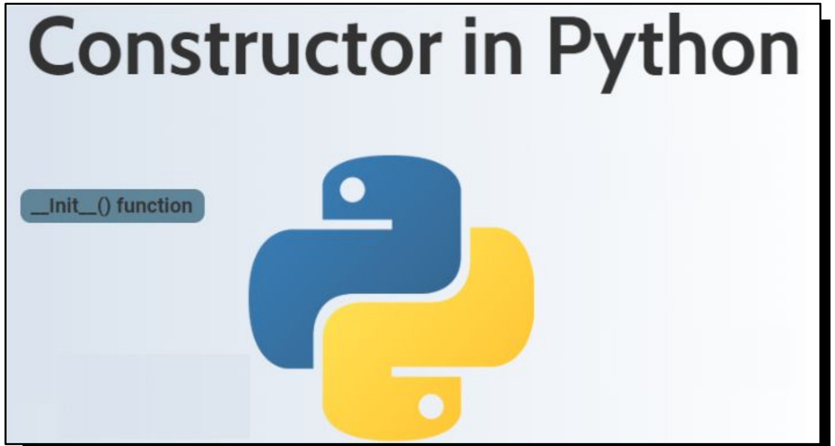
Methods are like the services offered in a building. There are two types:

- **Defining Methods:** This is like setting up services in a building.
- **Utilities of Methods:** They make it easier to manipulate and work with objects.

Constructors: The Groundbreaking Ceremony (init)

The constructor is a special method named __init__.

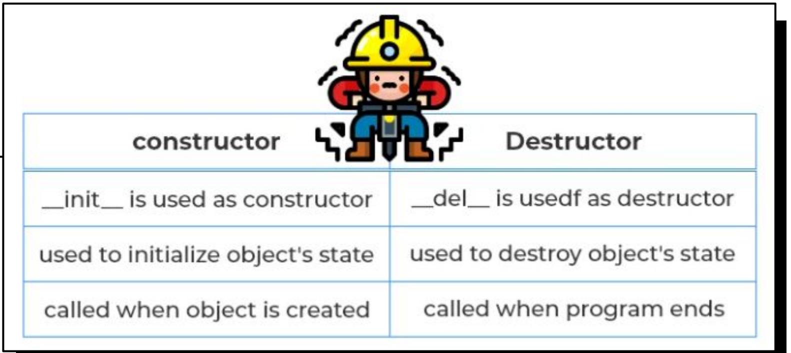
- **Purpose:** Constructors are used to initialize the data attributes of an object.
- **How it works:** When you create an object of a class, Python automatically calls the __init__ method to initialize the object.



Destructors: The Demolition Crew (del)

The destructor in Python is a method named `__del__`.

- **Purpose:** This is Python's way of tidying up, ensuring no loose ends remain.
- **Usage:** They're used in resource-intensive operations like closing file connections or releasing memory.



constructor	Destructor
__init__ is used as constructor	__del__ is usedf as destructor
used to initialize object's state	used to destroy object's state
called when object is created	called when program ends

Example: Defining Classes in Python

Imagine you're constructing a series of office buildings. Each building will have similar features: floors, offices, elevators, etc. In Python, you create a class `OfficeBuilding` that defines these common features and behaviors.

```
class OfficeBuilding:
def __init__(self, floors, offices):
self.floors = floors
self.offices = offices
def open_doors(self):
print("Doors are open for business!")
```

- `class` defines a new class. In this case, it's defining a class named `OfficeBuilding`.
- `def` defines a function (or method) in Python.
- `__init__` is a constructor. It is called when an object of the class is created. It initializes the object's attributes and sets up its initial state.
- `self` refers to the instance of the object itself. It represents the object being created.
- `floors` and `offices` represent the attributes of the `OfficeBuilding` class that will be initialized when an object is created from this class.
- `self.floors` and `self.offices` are instance variables or attributes of the `OfficeBuilding` class. They store the number of floors and offices in the office building, respectively. The values passed as arguments to the constructor (`floors` and `offices`) are assigned to these instance variables.
- `open_doors` is a method defined within the `OfficeBuilding` class. It's a function associated with instances of this class.
- `self` is the first parameter of this method, indicating that it's a member of the class and can access the instance's attributes and other methods.

Example: Creating Objects from Classes

From the OfficeBuilding class, you can create multiple office buildings, each with its own specific number of floors and offices.

```
building1 = OfficeBuilding(10, 200)
building2 = OfficeBuilding(20, 400)
building1.open_doors()
# This calls the open_doors method for
building1
```

Hence, building2 now refers to a different office building with 20 floors and 400 offices.

- `building1` is a variable that represents an instance of the `OfficeBuilding` class.
- `OfficeBuilding(10, 200)` is a constructor call. It creates a new instance of the `OfficeBuilding` class and initializes it with 10 floors and 200 offices.

So, `building1` now refers to an office building with 10 floors and 200 offices.

- Similarly, `building2` is a variable representing another instance of the `OfficeBuilding` class.
- `OfficeBuilding(20, 400)` creates a new instance of the class with 20 floors and 400 offices.

Hence, `building2` now refers to a different office building with 20 floors and 400 offices.

In-Class Built-in functions Exercises: Hands-on Coding!

💡 **Hey there! Before we unveil our solutions,** how about taking a shot at it yourself? Keep in mind that we've provided some broad strokes, so a detailed approach will help fine-tune the solution. But, this is all in good fun and a chance for you to flex those problem-solving muscles. Give it a go, and remember, it's all about learning and improving!

Exercise 1: City Library Management System

Introduction

In this exercise, you'll build a basic version of a City Library Management System using Python. This system will help you understand the application of Object-Oriented Programming concepts like classes, inheritance, and encapsulation, along with other fundamental Python features like lists, dictionaries, and exception handling.

Objective

To create a Python program that manages book records in a city library system, allowing for adding, searching, lending, and returning books.

Problem Statement

The city library needs a digital system to keep track of its books. The system should allow library staff to add new books, search for books by title, lend books to members, and receive returned books. Each book should have a title, author, and an availability status.

Hints

- When a book is checked out or returned, remember to update its availability status.
- Consider using a dictionary in the Library class for efficient book search by title.
- Use a loop to continuously allow the user to choose an action until they decide to exit.

Explanation

- The `Book` class represents individual books with methods to check them out and return them.
- The `Library` class manages a collection of books. Books can be added, searched by title, lent out, and returned.
- The main program loop allows library staff to perform actions based on user input.
- Exception handling ensures that the program can gracefully handle unexpected situations.

Exercise 1: City Library Management System - Instructions

Introduction

1. **Define a Book Class**:
    - Include attributes for title, author, and availability.
    - Define methods for checking out and returning a book, which update the availability status.
2. **Create a Library Class**:
    - Store a collection of books.
    - Include methods to add books, search for a book by title, lend a book, and return a book.
3. **Implement User Interaction**:
    - Use input functions to allow staff to add, search, lend, and return books.
    - Use loops and conditional statements to handle different user actions.
4. **Error Handling**:
    - Implement try-except blocks to handle situations like searching for a book that doesn't exist or lending out an already lent book.
5. **Data Structures**:
    - Use a list to store the collection of Book objects in the Library class.
    - Use a dictionary or a set for efficient searching of books by title.

Exercise 2: City Library Management System

Introduction

In this practical exercise, you'll develop a basic version of an Apartment Management System. This system will integrate the concepts of class and instance variables in Python, along with other key programming features such as loops, conditional statements, and exception handling.

Objective

To create a Python program that manages different aspects of apartment buildings in a city, including tracking the number of units, occupancy status, and facilities.

Problem Statement

A property management company needs a system to manage multiple apartment buildings in a city. Each building has a set number of apartment units, certain facilities (like a gym or pool), and a record of which units are occupied. The system should allow the management to add new buildings, update buildings, update occupancy, and list available facilities in each building.

Hints

- Remember to update the occupancy status when a new unit is filled or vacated.
- Consider implementing a method to calculate the percentage of occupied units in each building.

Explanation

- The ApartmentBuilding class represents each apartment building, with shared facilities as a class variable and individual attributes..
- The update_occupancy method allows updating the number of occupied units, within valid limits.
- The available_units and occupancy_rate methods provide information about each building.
- The main program loop lets the user add new buildings, update occupancy, and list all buildings with their details.
- Exception handling ensures input errors are managed effectively.

Exercise 2: City Library Management System - Instructions

Introduction

1. **Define an ApartmentBuilding Class**:
    - Include class variables for facilities shared across all buildings.
    - Use instance variables to track each building's name, total units, and occupied units.
2. **Building Management Methods**:
    - Implement methods to add new buildings, update occupancy, and list facilities.
    - Include functionality to check for available units in a building.
3. **User Interaction**:
    - Use input functions to allow the user to interact with the system.
    - Implement loops and conditional statements for different user actions.
4. **Data Structures and Error Handling**:
    - Use a list to manage a collection of ApartmentBuilding objects.
    - Apply exception handling to manage errors, such as trying to occupy an already full building.

Exercise 3: City Traffic Control System

Introduction

In this exercise, you'll develop a City Traffic Control System using Python. This system will help you understand how to effectively use methods and modules, along with core Python concepts like loops, conditionals, and data structures.

Objective

To create a Python program that simulates traffic control in a city, handling different types of vehicles and traffic signals.

Problem Statement

A city's traffic department needs a system to manage traffic flow. The system should track different types of vehicles (cars, bikes, buses) and control traffic signals (red, yellow, green) at intersections. It should allow for adding new vehicles, changing traffic signals, and displaying the current state of traffic at an intersection.

Hints

- Consider using a list or a set to manage the vehicles at the intersection.
- Implement a method to clear the intersection when the signal turns green.

Explanation

- The Vehicle class represents different types of vehicles, with a method to display the vehicle type.
- The TrafficSignal class manages the traffic signals, with methods to change and display the current signal.
- The main program loop allows user interaction for adding vehicles, changing signals, and displaying the current traffic state.
- The use of separate modules (vehicle.py and traffic_signal.py) organizes the code and improves readability.
- Exception handling ensures the system runs smoothly, even when faced with unexpected inputs or errors.

Exercise 3: City Traffic Control System - Instructions

Introduction

1. **Create Vehicle and TrafficSignal Classes**:
   - Vehicle class with methods to add a vehicle and display its type.
   - TrafficSignal class with methods to change the signal and display the current signal.
2. **Simulate Traffic at an Intersection**:
   - Use a loop to simulate time passing, changing traffic signals accordingly.
   - Allow adding vehicles to the intersection and display the current state of the intersection.
3. **Implement User Interaction**:
   - Use input functions for user actions like adding vehicles and changing signals.
   - Use conditionals to handle different traffic scenarios based on the signal.
4. **Use Modules and Exception Handling**:
   - Organize the classes into separate modules for better code management.
   - Apply try-except blocks for handling invalid inputs or operational errors.

coding temple

Exercise 4: City Event Planner System

Introduction

In this exercise, you're going to develop a City Event Planner System in Python. This application will help you understand how to effectively utilize methods, modules, and text file handling, combined with other Python programming essentials.

Objective

To build a Python program that helps in planning and managing events in various locations across the city, including scheduling, participant tracking, and storing event details.

Problem Statement

The city's event management team needs a digital system to organize multiple events. The system should allow the team to add new events, register participants, and save event details to a file. Each event should have a name, date, location, and a list of participants.

Hints

- Consider using a dictionary to manage events with their names as keys.
- Implement a function to format event details neatly before saving them to a file.

Explanation

- The Event class in the event_manager module handles individual event details and participant registration.
- The main program manages a collection of events, allowing the user to add events, register participants, and save or display event details.
- File handling is used to store event details in a file (events.txt) and load them when the program starts.
- Exception handling ensures the system is robust, especially in file operations and user inputs.
- Data structures like dictionaries (for managing events) and lists (for participants) efficiently organize the data.

Exercise 4: City Event Planner System - Instructions

Introduction

1. **Create Event Class and EventManager Module**:
   - The Event class should have methods to add an event, register participants, and display event details.
   - Organize the Event class in an EventManager module.
2. **Implement File Handling for Event Details**:
   - Use text file handling to save and read event details.
   - Ensure the system can store event information in a file and retrieve it when needed.
3. **User Interaction for Event Management**:
   - Use input functions to add events, register participants, and display event details.
   - Implement loops and conditional statements for handling various actions.
4. **Exception Handling and Data Structures**:
   - Use try-except blocks for managing file handling errors and invalid inputs.
   - Utilize lists, dictionaries, or tuples to manage events and participants.

Exercise 5: City Vehicle Registration System

Introduction

This exercise involves creating a City Vehicle Registration System in Python, putting into practice the concepts of creating and managing multiple objects from classes.

Objective

Develop a Python program to manage vehicle registrations in a city. The system will track vehicles by type, registration number, and owner details.

Problem Statement

The city's Department of Motor Vehicles (DMV) needs a system to handle the registration of various types of vehicles like cars, bikes, and trucks. Each vehicle should have a unique registration number, type, and owner information. The system should allow for registering new vehicles, updating owner information, and displaying all registered vehicles.

Hints

- Consider using the vehicle's registration number as a unique identifier.
- Ensure the registration number is not duplicated when adding a new vehicle.

Explanation

- The Vehicle class represents individual vehicles, each with a registration number, type, and owner.
- The DMV system uses a dictionary to manage vehicle objects, facilitating easy access and modification based on registration numbers.
- The user can register new vehicles, update owner information, and view all registered vehicles.
- Exception handling ensures the system can handle unexpected situations like duplicate registration numbers or invalid inputs.
- This system demonstrates the creation and management of multiple objects from a class, each representing a unique entity in the program.

Exercise 5: City Vehicle Registration System - Instructions

Introduction

1. **Define a Vehicle Class**:
    - Include attributes for registration number, type (car, bike, truck), and owner.
    - Include methods to update the owner and display vehicle details.
2. **Implement the DMV System**:
    - Create a system to register new vehicles and store them in a collection.
    - Allow updating the owner information for a registered vehicle.
    - Implement a method to display all registered vehicles.
3. **User Interaction for Vehicle Registration**:
    - Use input functions to register new vehicles, update owner details, and display all vehicles.
    - Use loops and conditional statements to handle different user actions.
4. **Data Structures and Error Handling**:
    - Use a dictionary to store vehicles, with registration numbers as keys.
    - Apply try-except blocks to handle errors, such as duplicate registrations or invalid inputs.

Exploring the Digital Metropolis: A Journey Through Python's Object-Oriented Programming

- On this journey, we’ve learned how to:
- Design and manage our digital metropolis using Python's OOP fundamentals.
- Understand classes as the blueprints of our city and objects as the unique buildings constructed from these blueprints.
- Explore the vital roles of constructors and destructors.
- Comprehend streets of data structures - lists, sets, dictionaries, and tuples.
- Manage file handling, much like managing the city's archives.
- Build and maintain different parts of our city, from managing vehicle registrations to planning city events.

✓ Design and manage our digital metropolis using Python's OOP fundamentals.
✓ Understand classes as the blueprints of our city and objects as the unique buildings constructed from these blueprints.
✓ Explore the vital roles of constructors and destructors.
✓ Comprehend streets of data structures - lists, sets, dictionaries, and tuples.
✓ Manage file handling, much like managing the city's archives.
✓ Build and maintain different parts of our city, from managing vehicle registrations to planning city events.