



Software Engineering

Module 4: Python Object-Oriented Programming (OOP)

Clean Code



Intro

Hello, Coding Templers! 🌟

Get ready for a coding adventure in the kitchen!

Picture a futuristic culinary space where Python recipes unfold the world of Object-Oriented Programming. Like crafting a gourmet meal, each line of code adds flavor to your digital creation.

Let's do our aprons and explore the art of clean coding in Python. Discover the secrets of crafting functional and beautifully written code. Bon appétit and happy coding! 🚀🔍👩👨🐍



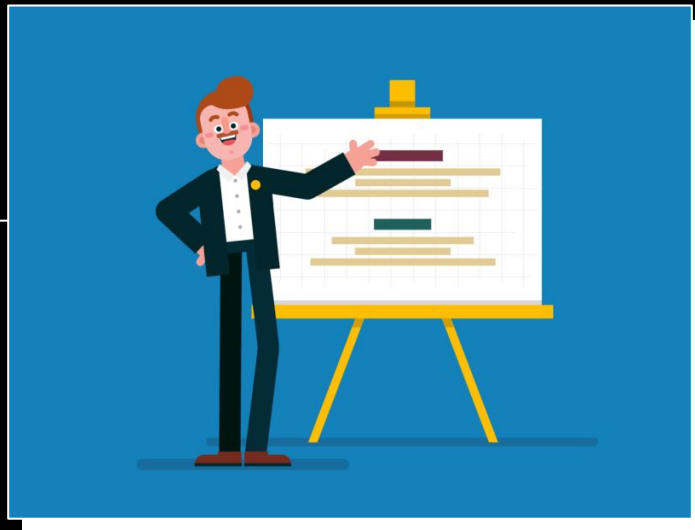
coding
temple



Learning objectives

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

- Apply clean coding principles to enhance the readability and maintainability of Python code.
- Understand the significance of clear and concise naming conventions for variables, functions, and classes.
- Demonstrate the ability to refactor a Python class for improved cleanliness and efficiency.
- Differentiate between well-organized, modular code and code that is cluttered.



Embracing the Art of Clean Coding

Clean code, like an organized kitchen, is structured, readable, and easy to navigate. Let's put on our chef hats and explore this culinary-inspired coding journey!



**coding
temple**



Importance of Clean Code: The Well-Organized Kitchen

This is the essence of clean code. It's about writing code that's not just functional, but also:

- Neat
- Understandable
- Easy to maintain



Example: The Messy vs. Clean Kitchen

MESSY KITCHEN

It's represented like a function with poorly named variables, like a cluttered kitchen.

```
def calc(a, b):  
    return a * b + 100 - a
```



Example: The Messy vs. Clean Kitchen

CLEAN KITCHEN

On this occasion we name our variables clearly, making the function's purpose obvious.

```
def calculate_discount(price, discount_rate):  
    return price * discount_rate + 100 - price
```



Clear and Concise Variable and Function Naming: Precisely Named Ingredients

Precision enhances not only code readability but also facilitates a deeper understanding of the logic and functionality within the software.



Example: Ingredient Labels

VAGUE LABELS

Refer to poorly chosen or ambiguous names for variables, functions, classes, or other elements within code.

```
def process(data):  
    # ... some code ...
```



Example: Ingredient Labels

CLEAR LABELS

Clear labels refer to well-chosen and descriptive names for variables, functions, classes, or other elements within the code.

```
def process_customer_order(order_details):  
    # ... some code ...
```



Organizing and Formatting Code Effectively: Strategically Arranged Kitchen

In the world of coding, adopting a similar strategic mindset means structuring code with intention, ensuring that elements are logically arranged for optimal efficiency and ease of use.



Disorganized Space

Code with inconsistent spacing and poor grouping.

```
def bakeCake():  
eggs=3  
flour=1  
sugar=2  
mix(flour, sugar)  
    add(eggs)
```



pixtastock.com - 67017529

Organized Space

Code with proper indentation and grouping is far more readable.

```
def bake_cake():  
    eggs = 3  
    flour = 1  
    sugar = 2  
    mix(flour, sugar)  
    add(eggs)
```



Writing Reusable Functions and Classes: The Master Recipe

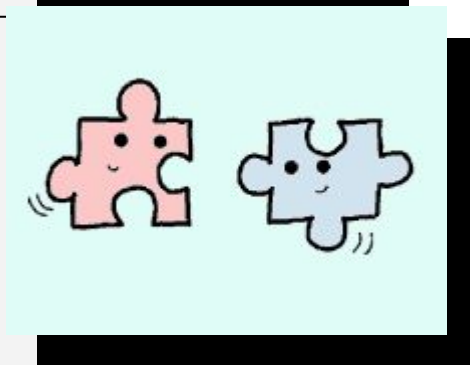
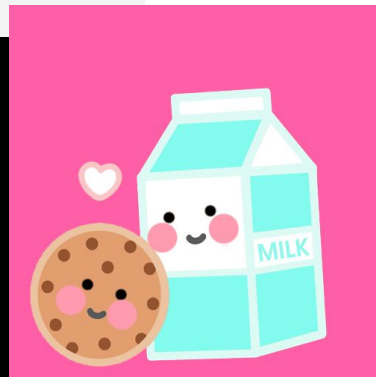
Reusable functions and classes in coding can be adapted for multiple uses.



One-Time Dish

It's like writing a function for a very specific.

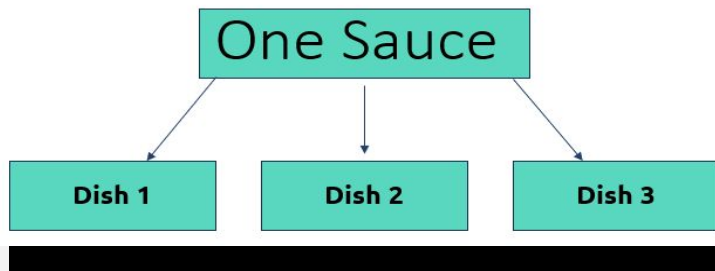
```
def calculate_20_percent_discount(price):  
    return price * 0.8
```



Master recipe

This function calculates any percentage discount.

```
def calculate_discount(price, percentage):  
    return price * (1 - percentage / 100)
```



Applying Clean Code Principles in Python OOP

On this occasion, our goal is to make each component as clean and efficient as possible. Let's explore the principles of crafting Clean Classes.



Designing Clean Classes: Crafting the Perfect Ingredient

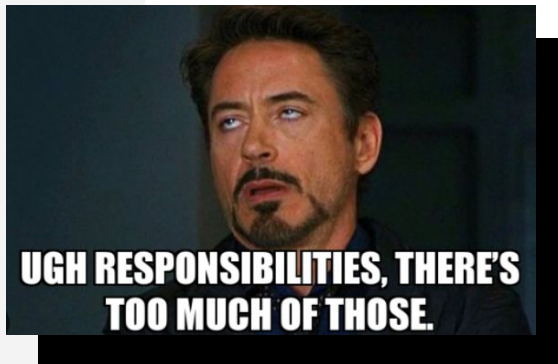
A well-designed class is like a high-quality ingredient; it should be pure, without unnecessary additives.



Mixed Ingredient (Poor Class Design)

These are difficult to use in various dishes.

```
class UserAuthDatabaseLogger:  
    # Handles authentication, database operations, and logging
```



Pure Ingredient (Clean Class Design)

It's like separating ingredients turning them more versatile and understandable.

```
class UserAuthenticator:  
    # Handles user authentication  
class DatabaseManager:  
    # Handles database operations  
class Logger:  
    # Handles logging
```



Principles of Modularity in Classes: Building a Menu of Dishes

In Modular components, we create a versatile "menu" of classes, allowing us to combine and customize features for a wide array of coding scenarios.



Menu Variety

Classes that are too specific.

```
class EventOrganizer:
    # A class with very specific and
    # intertwined methods
    def send_invitations(self, guest_list):
        # Code to send invitations
        pass
    def book_venue(self, venue):
        # Code to book venue
        pass
    def arrange_catering(self, menu):
        # Code to arrange catering
```



Diverse Menu (Modular Classes)

Modular classes offer the flexibility to mix and match, creating a variety of solutions.

```
class InvitationManager:
    def send_invitations(self, guest_list):
        # Code focused only on sending invitations
        pass

class VenueBooker:
    def book_venue(self, venue):
        # Code focused only on booking venues
        pass

class CateringService:
    def arrange_catering(self, menu):
        # Code focused only on catering services
        pass
```



Best Practices and Common Pitfalls in Clean Coding and Python OOP



It's time to unravel the best practices that pave the way to code brilliance while navigating common pitfalls that can obscure the path to programming excellence:

1. Clarity in Naming
 - **Best practice:** Ensures that your code is easily understandable and maintainable.
 - **Common pitfall:** Using generic names like `data`, `process`, or `x` can lead to confusion and make the code hard to follow.
1. Single Responsibility Principle
 - **Best practice:** Design classes to have a single, well-defined purpose.
 - **Common pitfall:** Overloading a class with multiple responsibilities.
1. Modularity in Class Design
 - **Best practice:** This enhances flexibility and reusability.
 - **Common pitfall:** Specific classes limit usability and adaptability.
1. Readable and Organized Code
 - **Best practice:** Format your code with consistent indentation, spacing and coding-grouping.
 - **Common pitfall:** hard to navigate and maintain.
1. Reusability and Efficiency
 - **Best practice:** Write reusable functions and classes.
 - **Common pitfall:** Writing code for one-time use can lead to redundancy and inefficiencies.

| | Best practice | Common pitfall |
|---------------------------------|--|--|
| Clarity in Naming | Understandable and to maintain | Generic names lead to confusion |
| Single Responsibility Principle | Single, well-defined purpose classes | Overloading a class with multiple responsibilities |
| Modularity in Class Design | Flexibility and reusability | Specific classes limit usability and adaptability |
| Readable and Organized Code | Consistent indentation, spacing, and coding-grouping | Hard to navigate and maintain |
| Reusability and Efficiency | Write reusable functions and classes | One-time use codes can lead to redundancy and inefficiencies |

Common Pitfalls

Let's uncover the common pitfalls that can hinder your journey toward code excellence:

Overcomplication

Pitfall: Adding unnecessary complexity to code.

Avoidance: Keep solutions as simple as possible.

Neglecting Code Reviews and Refactoring

Pitfall: Regularly review and refactor code.

Avoidance: Regularly review and refactor code to improve its clarity, efficiency, and maintainability.

Inconsistent Coding Standards

Pitfall: Lack of consistent coding standards.

Avoidance: Establish and adhere to coding standards.

Ignoring Documentation


Pitfall: Overlooking the importance of documenting code.

Avoidance: Regularly document code.

| | Pitfalls | Avoidance |
|---|--|---|
| Overcomplication | Adding unnecessary complexity to code | Keep solutions as simple as possible |
| Neglecting Code Reviews and Refactoring | Regularly review and refactor code | Code revision to improve its clarity, efficiency, and maintainability |
| Inconsistent Coding Standards | Lack of consistent coding standards | Establish and adhere to coding standards |
| Ignoring Documentation | Overlooking the importance of documenting code | Regularly document code |

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. Restaurant Reservation System Refactor

| Objective | Problem Statement | Hints | Script Description | Explanation |
|--|--|--|---|---|
| To apply clean coding principles and Python OOP concepts to refactor and enhance a basic restaurant reservation system. This exercise will focus on improving readability, modularity, and efficiency of the code. | You have inherited a Python script for a restaurant reservation system. The script is functional but poorly organized, with convoluted code and unclear variable names. Your task is to refactor this script, applying clean coding principles, including clear naming, organizing code into classes and functions, and ensuring efficient use of data structures. | <ul style="list-style-type: none">• Look for long functions that can be broken down into smaller, more focused functions.• Identify repeated code blocks that can be turned into reusable functions or methods.• Use dictionaries for storing restaurant tables and reservations for efficient lookups.• Utilize lists or sets for handling collections of data like available menus or customer information.• Implement try-except blocks for handling user input errors or file handling exceptions. | This script represents a basic restaurant reservation system. It uses a dictionary to keep track of reservations for three tables. The functions <code>book_table</code> and <code>cancel_booking</code> allow users to book or cancel reservations, respectively. The main function offers a simple text-based interface for interaction. However, the script has several limitations in terms of organization, modularity, and error handling, making it a good candidate for refactoring to apply clean coding principles. | <ul style="list-style-type: none">• The <code>Restaurant</code> class is responsible for managing reservations. It uses a dictionary for table reservations, allowing for efficient lookups and updates.• Functions <code>reserve_table</code> and <code>cancel_reservation</code> handle reservation logic, adhering to the single responsibility principle.• The <code>main</code> function provides an interactive user interface, handling inputs and outputs.• Error handling is implemented to manage user input errors, such as invalid table numbers or actions. |

Exercise 1. Restaurant Reservation System Refactor - Instructions

Instructions

1. Analyze the provided script to identify areas that lack clarity and efficiency.
2. Rename variables and functions to be descriptive and meaningful.
3. Break down the script into classes and functions, ensuring each has a single responsibility.
4. Use appropriate data structures (lists, sets, dictionaries, tuples) to handle restaurant data effectively.
5. Implement error handling using try-except blocks where necessary.
6. Add input functionality to interact with the user for reservations.
7. (Optional) Incorporate text file handling to store and retrieve reservation data.

Exercise 2. Refining a Library Management System - Instructions

| Objective | Problem Statement | Hints | Script Description | Explanation |
|---|--|---|--|--|
| To apply clean coding principles and Python OOP techniques in restructuring and enhancing a basic library management system. This exercise will focus on improving code clarity, efficiency, and organization, utilizing various Python constructs. | You are provided with a rudimentary Python script for managing a small library. The script, while functional, suffers from poor organization, unclear variable names, and lack of modularity. Your task is to refactor the script to make it more readable, efficient, and maintainable, adhering to clean coding practices. | <ul style="list-style-type: none">● Identify code blocks that can be encapsulated into separate functions or classes.● Consider using a dictionary for managing book inventory and user data.● Utilize lists or sets for tracking borrowed books or categories.● Error handling is crucial for inputs like book IDs or user actions.● Clean up any repetitive code segments into reusable functions or methods. | <p>This script is a basic implementation of a library management system. It maintains a dictionary of books with quantities and another for book names. The functions <code>add</code> and <code>borrow</code> allow users to add or borrow books by their IDs, but the code suffers from unclear variable names (e.g., <code>b</code>) and a lack of proper error handling. The use of book IDs instead of titles might confuse users. Additionally, the system lacks modularity, with all functionality being handled in a monolithic structure. This script is a suitable candidate for refactoring to enhance clarity, structure, and user-friendliness.</p> | <ul style="list-style-type: none">● The <code>Library</code> class manages the library's book inventory. It uses a dictionary for efficient tracking of book titles and quantities.● Methods <code>add_book</code> and <code>borrow_book</code> handle specific library operations, aligning with the single responsibility principle.● The <code>main</code> function serves as the user interface, handling inputs and directing library operations.● Error handling is implemented for user inputs, enhancing the robustness of the application. |

Exercise 2. Refining a Library Management System - Instructions

Instructions

1. Examine the provided script to identify areas that can be improved for better readability and structure.
2. Rename variables and functions to be more descriptive and meaningful.
3. Reorganize the script into classes and functions with clear, single responsibilities.
4. Utilize appropriate data structures (lists, sets, dictionaries, tuples) for efficient data management.
5. Implement error handling using try-except blocks for robustness.
6. Use input functions to interact with the user for managing library operations.
7. Optionally, integrate text file handling for persistent data storage.

Exercise 3. Building an Online Shopping Cart System

Objective

Develop an online shopping cart system using Python, applying clean coding principles and OOP techniques. This exercise will focus on constructing a modular, efficient, and user-friendly shopping cart, incorporating various Python features.

Problem Statement

Your task is to create a Python-based online shopping cart system for a retail store. The system should allow users to add items to their cart, view the cart, remove items, and check out. The system should be designed with clarity, efficiency, and user interaction in mind.

Hints

- Consider the use of a dictionary within the shopping cart class for efficient management of items.
- Utilize classes and OOP principles to encapsulate related attributes and methods.
- Remember to handle edge cases, such as attempting to remove an item not in the cart.

Explanation

- The `Item` class in `item.py` defines the structure for items in the store, with attributes like name, price, and category.
- The `ShoppingCart` class in `cart.py` manages the shopping cart, including adding, viewing, and removing items, as well as checkout functionality.
- The main program allows user interaction with the shopping cart system, handling different actions like adding or removing items.
- The modular design enhances readability and maintainability, with each module handling a specific aspect of the system.
- The system utilizes OOP principles for organization, and try-except blocks ensure robust error handling for user inputs.

Exercise 3. Building an Online Shopping Cart System - Instructions



Instructions

1. Design a class to represent individual items with attributes like name, price, and category.
2. Create a shopping cart class that can hold multiple items and manage cart operations.
3. Implement functions for adding items to the cart, viewing the cart, removing items, and checking out.
4. Use lists or dictionaries to manage the items in the cart.
5. Implement user input functionality for interacting with the shopping cart system.
6. Apply try-except blocks for handling potential input errors.
7. Modularize the system by separating functionalities into different Python modules.

Exercise 4. Developing a Movie Theater Seat Reservation System



Objective

Create a Python application to manage seat reservations in a movie theater, emphasizing clean coding practices, particularly in the use of clear and descriptive variable and function names. This exercise will also incorporate OOP, conditional and loop statements, string manipulation, error handling, and Python data structures.

Problem Statement

You are tasked with developing a seat reservation system for a movie theater. The system should allow users to view the seating arrangement, select seats, and confirm their reservations. The focus is on creating a user-friendly interface and backend logic using clear, descriptive naming conventions for variables and functions.

Hints

- Consider representing each seat as a dictionary with keys like 'row', 'number', and 'reserved'.
- Use a list of lists or a dictionary to represent the arrangement of seats in the theater.
- Implement loops for navigating through the seating chart and conditional statements for reservation logic.

Explanation

- The `Theater` class in `theater.py` manages the seating arrangement. Each seat is a dictionary with details like row number, seat number, and reservation status.
- The `display_seating_chart` method visualizes the seating arrangement, using 'X' for reserved seats and 'O' for available seats.
- The `reserve_seat` method handles the reservation of a seat, checking its current status and updating it if available.
- The `main.py` module provides the user interface, allowing the user to interact with the theater system, choose seats, and handle their reservation.
- The use of clear and descriptive names for classes, methods, variables, and modules enhances the readability and maintainability of the code.

Exercise 4. Developing a Movie Theater Seat Reservation System - Instructions

Instructions

1. Design a class to represent the theater, including attributes for the seating arrangement.
2. Implement methods for displaying the seating chart, selecting seats, and confirming reservations.
3. Create a user interface using input functions to interact with the theater system.
4. Use appropriate data structures to manage the seating arrangement (consider lists or dictionaries).
5. Implement error handling for user inputs, such as selecting already reserved seats or invalid seat numbers.
6. Organize your code into modules to separate the user interface from the backend logic.
7. Ensure all variables and functions have descriptive and clear names for easy understanding and maintenance.

Exercise 5. Crafting a Weather Forecasting App

Objective

Develop a Python application to provide weather forecasts, focusing on clean coding principles, including clear naming conventions, modular design, and effective use of Python features like OOP, data structures, and exception handling.

Problem Statement

You are tasked with creating a weather forecasting app that allows users to check the weather forecast for different cities. The app should be user-friendly, efficient, and well-structured, showcasing clean coding practices.

Hints

- Consider using a dictionary with city names as keys and weather data as values.
- Implement loops for user interaction and conditional statements to handle different user choices.
- Utilize external modules or APIs for fetching weather data if desired, though this can be simulated for the exercise.

Explanation

- The `WeatherForecast` class in `weather_forecast.py` handles the storage and retrieval of weather data for different cities.
- The `get_weather` method fetches the weather for a given city, while `display_weather` prints the forecast to the user.
- The `main.py` module provides the user interface for the app, allowing users to input city names and receive weather forecasts.
- Clean coding principles are applied throughout the app, with clear naming for classes, methods, and variables, and a modular design separating data handling from user interaction.
- Exception handling ensures robust user input processing and error management.

Exercise 5. Crafting a Weather Forecasting App - Instructions

Instructions

1. Create a class to represent the weather forecasting functionality, including methods for fetching and displaying weather data.
2. Implement user interaction for selecting cities and viewing their weather forecasts.
3. Use dictionaries or lists to manage the data related to cities and their weather.
4. Apply exception handling for user input validation and error scenarios.
5. Structure your code into different modules to separate the user interface from the logic handling the weather data.
6. Ensure all variables, functions, and classes have descriptive and meaningful names for clarity.

Cooking Up Excellence: Savoring the Art of Clean Coding in Python

On this journey, we've learned about:

- The importance of writing clear and concise code.
- The significance of organizing and formatting code effectively.
- The art of designing clean classes and the principles of modularity.
- The value of simplicity and focus.

In the end, clean coding in Python, much like exquisite cooking, is about creating something that is not only effective and functional but also a joy to behold and experience.

We hope you've enjoyed this lesson. Happy coding!

