coding temple

Python Basics

Module 4:

Python Modules

Learning Objectives

- Understand the concept and functionality of Python modules, including both built-in and custom modules.
- Apply the process of creating custom modules in Python by writing their own code for specific functionalities.
- Demonstrate the ability to import entire modules and specific functions from those modules into their Python scripts.
- Understand and apply the use of aliasing in importing modules and functions to enhance code readability and efficiency.

Overview of Modules in Python:

Unpacking Your Digital Toolbox

What Are Python Modules?
File containing a Python code (with or without functions or variables) designed to perform specific tasks.

Why Use Modules?

- Efficiency
- Organization
- Reusability

Let’s Recap! Exploring Standard Python Modules

math: For mathematical tasks.

datetime: When working with dates and times.

os: To interact with your operating system.

math:

```
import math
print(math.sqrt(16)) # Outputs: 4.0
```

datetime:

```
import datetime
current_time = datetime.datetime.now()
print(current_time) # Outputs the current date and time
```

os:

```
import os
print(os.getcwd()) # Prints the current working directory
```

Importing Specific Functions in Python:

Selecting the Right Tools for the Job

Importing only the sqrt function from math

```
from math import sqrt
print(sqrt(16)) # Directly using the sqrt tool,
no need to specify the toolbox
```

Using as

```
import datetime as dt
now = dt.datetime.now()
print(now) # Using a nicknamed tool for convenience
```

Creating Custom Modules in Python:
Tailoring Your Own Tools

But what happens when you need a tool that's not in your standard toolbox? This is where the art of creating custom modules in Python comes in. It's akin to a craftsman crafting their own unique tools to perfectly fit the needs of a specific project.

Advantages of Custom Modules

1. **Precision:** It does what you need.
2. **Efficiency:** Module can be reused in multiple projects, saving time.
3. **Understanding:** understanding of Python and programming put into practice.

Creating Custom Modules in Python:
Tailoring Your Own Tools

Step-by-Step Guide to Creating a Custom Module

Step 1: Start with a Blueprint
planning what the module will contain (functions, classes, or variables)

Common Pitfalls to Avoid

1. **Overloading the Toolbox:** Avoid unnecessary memory usage and potential conflicts by importing only what you need.
2. **Misplacing Tools:** Be precise with your import statements.
3. **Neglecting to Return Tools:** Be cautious about using from module import *.
4. **Ignoring the Manual:** Always take the time to understand the modules you are using to avoid misuse.

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: Building a Contact Management System

OBJECTIVES

To develop a basic contact management system in Python that allows users to add, view, delete, and search for contacts. This exercise will reinforce concepts of Python modules, custom functions, data structures (lists, sets, dictionaries, tuples), string manipulation, and basic error handling using try-except blocks.

PROBLEM STATEMENT

You are tasked with creating a simple contact management system for a small business. The system should enable adding new contacts, viewing all contacts, searching for a specific contact, and deleting a contact. Each contact should include a name, phone number, and email address.

HINTS

- Remember to import your `contact_manager` module in `main.py`.
- Use a loop to continuously display the menu until the user decides to exit.
- For searching and deleting contacts, consider iterating over the list and matching the contact's name.

EXPLANATION

This exercise integrates various Python concepts. The `contact_manager` module contains functions for managing contacts, using lists and dictionaries to store and manipulate contact data. The `main.py` script provides a user interface for interacting with the contact management system. Exception handling is used to manage potential errors gracefully, and string manipulation is utilized in searching and deleting contacts. This system demonstrates the practical application of Python's data structures, functions, and error handling in developing a real-world application.

Exercise 1: Building a Contact Management System - Instructions

INSTRUCTIONS

1. **Create a Module:**
   - Start by creating a Python module (`contact_manager.py`) that will contain all the functions for the contact management system.
2. **Design Functions:**
   - Implement the following functions in your module:
     - `add_contact(contacts, name, phone, email)`: Adds a new contact to the contacts list.
     - `view_contacts(contacts)`: Displays all the contacts.
     - `search_contacts(contacts, name)`: Searches for a contact by name.
     - `delete_contact(contacts, name)`: Deletes a contact by name.
3. **Implement Data Structures:**
   - Use a list to store all contacts. Each contact should be a dictionary with keys `name`, `phone`, and `email`.
4. **User Interface:**
   - In your main Python file (`main.py`), create a user interface using the `input` function that allows users to choose an action (add, view, search, delete contacts).
5. **Error Handling:**
   - Include try-except blocks to handle errors, such as searching for a non-existent contact or entering invalid data.
6. **Test Your System:**
   - Add at least 5 contacts, search for a couple of them, delete one, and view the updated contact list to ensure your system is working correctly.

Exercise 2: Weather Data Analyzer

OBJECTIVES

Develop a Python application that analyzes weather data from text files. This exercise aims to strengthen understanding of file handling, custom and built-in modules, data parsing, and basic data analysis techniques in Python.

PROBLEM STATEMENT

You are tasked to build a weather data analyzer that can read weather data from text files, each containing daily temperature (in Celsius) and rainfall (in mm) data. The system should be able to calculate the average temperature, total rainfall, and find the day with the highest temperature for each file.

HINTS

- Use Python's built-in `open()` function for file handling.
- Remember to convert string data read from the file into appropriate data types (int or float) for calculations.
- Iterate through the data list to implement the calculation functions.

EXPLANATION

This exercise integrates file handling, custom module creation, and basic data analysis. The `weather_analyzer` module contains specific functions to process and analyze weather data from text files. The `main.py` script provides an interface for users to input file names and displays calculated weather statistics. Exception handling ensures that the program can gracefully handle errors like missing files or data format issues. This system exemplifies practical file handling and data processing using Python's built-in capabilities and custom modules.

Exercise 2: Weather Data Analyzer - Instructions

INSTRUCTIONS

1. **Create a Custom Module:**
   - Create a Python module (`weather_analyzer.py`) that contains functions to process weather data.
2. **Design Functions in the Module:**
   - Implement functions in your module:
     - `read_weather_data(filename)`: Reads weather data from a file and returns it as a list of tuples.
     - `calculate_average_temperature(data)`: Calculates and returns the average temperature.
     - `calculate_total_rainfall(data)`: Calculates and returns the total rainfall.
     - `find_highest_temperature_day(data)`: Finds and returns the day with the highest temperature.
3. **Data Format in Text Files:**
   - Assume each line in the text files follows this format: `Day, Temperature, Rainfall`
   - Example of a line in a file: `1, 23.5, 5`
4. **User Interface:**
   - In your main Python file (`main.py`), create an interface that allows the user to input the filename and then displays the calculated weather statistics.
5. **Error Handling:**
   - Include try-except blocks to handle potential errors, such as file not found or incorrect data format.
6. **Test Your Analyzer:**
   - Create a few sample text files with weather data and use them to test your weather data analyzer.

Exercise 3: Recipe Ingredient Converter

OBJECTIVES

To create a Python application that converts the quantities of ingredients in a recipe from one unit to another (e.g., from cups to milliliters). This exercise will focus on importing specific functions from custom modules, reinforcing skills in modular programming, function usage, and unit conversion in Python.

PROBLEM STATEMENT

You are to develop a small Python program that helps users convert ingredient quantities in a recipe from one unit to another. The application should include a module with specific conversion functions and a main script that allows users to input ingredient quantities and select units for conversion.

HINTS

- Remember to use `from unit_converter import specific_function` to import only the needed functions.
- Use float conversion (`float()`) to handle numeric inputs for the ingredient quantities.

EXPLANATION

This exercise demonstrates the practical application of importing specific functions from a custom module in Python. The `unit_converter` module contains specialized functions for converting units, which are imported and utilized in the `main.py` script based on user input. This structure highlights the effectiveness of modular programming in Python, promoting code reusability and organization. The inclusion of error handling ensures that the program can manage incorrect user inputs gracefully.

Exercise 3: Recipe Ingredient Converter - Instructions

INSTRUCTIONS

1. **Create a Conversion Module:**
   - Create a Python module (`unit_converter.py`) with functions for different unit conversions, such as cups to milliliters, teaspoons to milliliters, etc.
2. **Implement Conversion Functions:**
   - In `unit_converter.py`, write specific functions like:
     - `cups_to_milliliters(cups)`: Converts cups to milliliters.
     - `teaspoons_to_milliliters(teaspoons)`: Converts teaspoons to milliliters.
   - Assume 1 cup = 237 milliliters and 1 teaspoon = 4.93 milliliters.
3. **User Interaction in Main Script:**
   - In the main script (`main.py`), use the `input` function to ask the user for the ingredient quantity and the unit they want to convert from.
   - Import and use the specific functions from the `unit_converter` module based on the user's choice.
4. **Error Handling:**
   - Include try-except blocks in `main.py` to handle invalid inputs like non-numeric values.
5. **Test the Converter:**
   - Run the program to test various conversions, ensuring each function in the module works as expected.

Exercise 4: Currency Exchange Calculator

OBJECTIVES

Develop a Python application to calculate currency exchange rates. This exercise emphasizes the use of aliasing in Python to rename imported functions or modules, enhancing code readability and efficiency.

PROBLEM STATEMENT

Your task is to create a currency exchange calculator that converts amounts between different currencies. The calculator should use a module containing various currency conversion functions, each of which will be aliased when imported into the main program.

HINTS

- Use float conversion (`float()`) to handle numeric inputs for the currency amounts.
- Aliasing can make your code more readable, especially if the original function names are long or not intuitive.

EXPLANATION

This exercise introduces the concept of aliasing in Python, where functions from a module are imported with more readable or shorter names (`to_euro` and `to_usd`). The `currency_converter` module contains functions for converting between different currencies. In the `main.py` script, these functions are imported with aliases to simplify the code and improve readability. The application allows users to input an amount and select a currency to convert from, demonstrating practical usage of aliased functions in a real-world scenario.

Exercise 4: Currency Exchange Calculator - Instructions

INSTRUCTIONS

1. **Create a Currency Conversion Module:**
   - Develop a Python module (`currency_converter.py`) with functions for different currency conversions, like USD to EUR, GBP to USD, etc.
2. **Implement Conversion Functions:**
   - In `currency_converter.py`, define functions such as:
     - `usd_to_eur(usd_amount)`: Converts USD to EUR.
     - `gbp_to_usd(gbp_amount)`: Converts GBP to USD.
   - Assume fixed exchange rates, for instance, 1 USD = 0.85 EUR, 1 GBP = 1.30 USD.
3. **Using Aliasing in Main Script:**
   - In the main script (`main.py`), import the conversion functions with aliases for simplicity and clarity.
   - Example: `from currency_converter import usd_to_eur as to_euro, gbp_to_usd as to_usd`.
4. **User Interaction:**
   - Use the `input` function to ask users for the amount and the currency they want to convert from.
   - Based on the input, use the aliased functions to perform the currency conversion.
5. **Test the Calculator:**
   - Run the program to ensure the currency conversion and aliasing are working correctly.

Exercise 5: Personal Fitness Tracker

OBJECTIVES

Create a Python application that functions as a personal fitness tracker. This exercise is designed to practice creating and using multiple custom modules in Python.

PROBLEM STATEMENT

Develop a fitness tracker application that calculates and tracks users' daily calorie intake and exercise routines. The application will utilize two custom modules: one for managing calorie intake and another for exercise routines.

HINTS

- Use dictionaries in the modules to store food items and exercises along with their calorie counts and durations.
- Remember to reset the total calories and exercise time at the start of each day if needed.

EXPLANATION

This exercise demonstrates the creation and use of multiple custom modules in a Python application. The `calorie_tracker` module manages calorie intake, while the `exercise_tracker` module tracks exercise routines. Both modules use dictionaries to store and calculate data. The `main.py` script serves as the user interface, importing functions from both modules to allow users to input their daily food intake and exercise routines, as well as view the total calories and exercise time. This setup exemplifies the modularity and organization of Python programming, showing how different aspects of an application can be managed in separate, focused modules.

Exercise 5: Personal Fitness Tracker - Instructions

INSTRUCTIONS

1. **Create Two Custom Modules:**
    - Develop a module for calorie tracking (`calorie_tracker.py`) and another for exercise routines (`exercise_tracker.py`).
2. **Implement Functions in Calorie Tracker Module:**
    - In `calorie_tracker.py`, write functions like:
        - `add_food_calories(food, calories)`: Adds calories for a given food item.
        - `get_total_calories()`: Returns the total calories consumed.
3. **Implement Functions in Exercise Tracker Module:**
    - In `exercise_tracker.py`, write functions such as:
        - `log_exercise(exercise, duration)`: Logs an exercise with its duration.
        - `get_total_exercise_time()`: Returns the total time spent on exercises.
4. **User Interaction in Main Script:**
    - In the main script (`main.py`), create an interface that allows users to input their daily food intake and exercises.
    - Import and use the functions from both `calorie_tracker` and `exercise_tracker` modules.
5. **Test the Fitness Tracker:**
    - Run the program to ensure that calorie intake and exercise routines are tracked and totaled correctly.

Conclusion

Modules: well-organized toolbox: functions, classes, and variables for specific tasks.

- ❑ Understand what modules are and why they're essential.
- ❑ Custom modules.
- ❑ Import modules and specific functions.
- ❑ Best practices and common pitfalls in using Python modules.