Exercises

Exercise 1: You are planning a weekend trip and considering visiting one of several cities. Each city has specific weather conditions and travel costs. Your decision depends on two factors: weather and budget. You will only visit a city if the weather is suitable (sunny or cloudy), and the cost of the trip must be within your budget.

Set the budget by asking users to input the amount they are willing to spend. Use the predefined weather conditions and costs for each city: Paris (sunny, €200), New York (rainy, €150), Tokyo (cloudy, €300), and Barcelona (rainy, €100). Check each city to see if it meets the weather and budget conditions. If the weather is sunny or cloudy, and the cost is within the budget, the program should suggest visiting the city. Otherwise, the program should explain why the destination is not suitable (either due to weather or cost).

Write a Python program that figures out where you can travel based on these criteria.

Exercise 2: Imagine you are building a simple program that suggests songs for a playlist based on your mood and activity. The user can input their mood and activity, and the program will suggest a song genre to fit their vibe.

Write a program that prompts the user to input their current mood (happy, sad, energetic, relaxed) and asks for their activity (studying, working out, chilling, partying).

Use conditional statements to match the mood and activity with a suitable music genre, Mood & Activity Combinations.

- Happy + Chilling: Pop
- Happy + Partying: Dance
- Sad + Studying: Jazz
- Sad + Chilling: Indie
- Energetic + Working Out: Hip-Hop
- Energetic + Partying: Electronic
- Relaxed + Studying: Classical
- Relaxed + Chilling: Acoustic

Print a message with the suggested genre based on the mood and activity the user selected.

Exercise 2: Imagine you are designing a workout app that creates a quick workout plan for the user. The user inputs how many sets they want for each exercise, and the program will generate a list of exercises with repetitions for each set.

Write a program that asks the user how many sets they want to do for each exercise (e.g., three sets). For each set, prints the repetitions for each exercise.

Repeat the number of sets and show each one. Use a loop to iterate through a list of exercises.

Set up a predefined list of exercises and repetitions for each exercise: Push-ups (10 reps), Squats (15 reps), Jumping jacks (20 reps), and Lunges (12 reps).

Print the workout plan in a structured way.

Exercise 3: Create a program that allows users to input their chosen exercises from a predefined list, ask for the number of sets and repetitions for each exercise, and generate a simple workout plan based on user input.

Choose your exercises from the list: Push-ups, Squats, Jumping jacks, and Lunges.

Exercise 4: This exercise will help you practice creating and using functions in Python by building a basic personal assistant program. The assistant will perform tasks like giving you a greeting, providing the current date and time, and telling jokes.

Imagine you are creating a simple text-based assistant that you can interact with through the console. The assistant will have several functions you can use for various tasks.

Define a `greet_user` function. This function should greet the user based on the time of day. If it is before noon, say "Good morning!"; if it is between noon and 6 PM, say "Good afternoon!"; and if it is after 6 PM, say "Good evening!".

Define a `get_date_time` function. This function should print the current date and time in a readable format.

Define a `tell_joke` function**.** This function should randomly select a joke from a list of three or more jokes and print it to the console.

Define a `main` function. This function will act as the main menu, allowing the user to choose between greeting, getting the date/time, and hearing a joke. The function should continuously prompt the user until they choose to quit.

To reach the earlier result, write each of the functions described above. In the main function, prompt the user to select an option. For example: "1" to get a greeting, "2" to get the current date and time, "3" to hear a joke, and "4" to quit. Call the selected function based on the user's input. Exit the program if the user enters "4" for quit.

Exercise 5. Create a text-based adventure game where players can explore a dungeon, meet monsters, and collect treasures. The player can choose to explore different rooms, and the game will continue until they decide to stop or meet a monster.

1. The player starts at the entrance of the dungeon.
2. The player can enter one of three rooms (Room 1, Room 2, Room 3).
3. Each room may have either a treasure or a monster.
4. The player can continue exploring rooms until they find a monster or choose to leave the dungeon.
5. If the player meets a monster, the game ends.
6. The player's treasure count is displayed at the end of each exploration.

Exercise 6. This exercise introduces you to the basics of object-oriented programming (OOP) in Python, which uses classes and objects to simulate a digital pet. You will create a pet with properties like energy, hunger, and methods to interact with it.

Imagine you have a digital pet that you need to take care of. This pet has attributes (like `energy` and `hunger`) that change over time and are based on actions. The goal is to keep the pet happy by feeding it and letting it play or rest. To do this, it is suggested to create a class called `Pet`. Initialize the pet with a name (string), an energy level (integer, starting at 50, maximum 100), and a hunger level (integer, starting at 50, maximum 100). Add methods to interact with the pet: `play()`: Increases hunger by 10 and decreases energy by 15; feed `()`: Decreases hunger by 15, but not below 0, and `rest()`: Increases energy by 20, but not above 100. Add a `status()` method to display the pet's name, hunger level, and energy level. After creating the pet, use a loop to interact with it, letting the user decide whether to play, feed, or rest the pet. Show the pet's status after each interaction and stop if the pet's energy or hunger goes out of bounds (either too low or too high).