Tutorial IV - Handling Data in more than one Dimension

Programming with Python

Calculating distances on a 2D grid

In this exercise, you'll work with tuples representing points in 2D space. You'll create functions to calculate distances from the origin, find the farthest point, and practice working with lists of tuples. This task will help you understand how to manipulate multi-dimensional data and perform calculations based on coordinates.

```
# TODO: Calculate distances in 2 dimensions
# - You are working with tuples representing points in 2D space (x, y).
# - Write a function that takes a tuple and returns the distance from the origin (0, 0).
# - Create a list of 5 tuples representing multiple points and calculate the distance for
 ↪  each point.
# - Create a function that takes a list of tuples and returns the point that is farthest
 ↪  from the origin.
# - Print the result to the console.
```

A phonebook application

In this exercise, you'll create a simple phonebook application that demonstrates the use of dictionaries in Python. This task will help you practice working with key-value pairs, user input handling, and basic file I/O operations. By implementing functions to add, update, and delete entries, you'll gain hands-on experience with common dictionary operations. Additionally, saving the phonebook to a text file will introduce you to persisting data between program executions.

```
# TODO: Create a phonebook application
# - Create a dictionary to store names and phone numbers.
# - Add at least 5 entries to the dictionary in the initialisation.
# - Write a function to look up a phone number by name based on a user input.
# - Write a function to update a phone number based on a user input.
# - Write a function to delete an entry by name based on a user input.
# - Write a function that saves the phonebook to a text file.
# - Write a small program that asks for a user input on whether the user wants to add,
 ↪  remove, or update a phone number.
# - After each operation, the phonebook should be saved to the text file.
```

Treasure Hunt Game

In this task on a treasure hunt game, you'll repeat how to handle user input and learn how to manage game states in several dimensions. Furthermore, you'll have the opportunity to enhance the game with additional features, making it more dynamic and challenging.

```
# a) TODO: Take a look at the code below and the instructions and add the missing code to
 ↪  make the game work.
# - The game should be played on a 3x3 grid.
# - The treasure is located at position (3, 3).
# - There is an obstacle at position (2, 2).
# - The player starts at position (1, 1) in the upper left corner.
# - The player can move up, down, left, or right.
# - The player cannot move outside the boundaries of the grid.
# - If the player hits the obstacle, the game is over.
# - If the player finds the treasure, the game is won.
# - Continuously prompt the player to enter a move (up, down, left, right).

grid_size = 3 # Size of the grid
treasure = (3,3) # Tuple for the treasure
obstacle = (2,2) # Tuple for the obstacle

# Player's starting position in a dictionary
player_position = {"x": 1, "y": 1}

# Function to move the player
def move_player(direction):
    if direction == "up" and player_position["y"] > 1:
        player_position["y"] -= 1
    # TODO: Add a move down
    # TODO: Add a move left
    elif direction == "right" and player_position["x"] < grid_size:
        player_position["x"] += 1
    else:
        print("Invalid move. Try again.")

# Function to check the player's position
def check_position():
    pos = (player_position["x"], player_position["y"])
    if pos == treasure:
        print("You found the treasure and won!")
        return False
    # TODO: Check if the player hit the obstacle and return False if so
    else:
        return True
```

```
# Main game loop
def play_game():
    print("Welcome to the Mini Treasure Hunt Game!")
    while True:
        print(f"Current position: {player_position}")
        move = input("Enter move (up, down, left, right): ").strip().lower()
        move_player(move)
        if check_position() == False:
            break

# Start the game
play_game()

# b) TODO: Improve the game by adding further functionality as described below.
# - There should be a game master who can set the size of the grid, the treasure and the
 ↪  obstacle.
# - Use the input to initialise the game
# - Print all information to the console at the start of the game
# Your code here

# c) TODO: Add a feature to allow for multiple obstacles on the grid.
# - The game master should be able to set the number of obstacles in the grid and their
 ↪  positions.
# - Print the position of the obstacles and the treasure after the game has been
 ↪  finished.
# - Try to make the printout as nice as possible.
# Your code here
```

That’s it!

You can find the solutions to these exercises online in the associated GitHub repository, but we will also quickly go over them in next week’s tutorial. To access the solutions, click on the Github button on the lower right and search for the folder with today’s lecture and tutorial. Alternatively, you can ask ChatGPT or Claude to explain them to you. Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities.