Multidimensional Lists and Functions

Overview

Today's lecture focused on understanding multidimensional lists and functions in Python, both of which are fundamental to handling complex data structures and performing repetitive tasks efficiently. This knowledge is foundational for creating sophisticated applications and streamlining our code.

Section 1: Multidimensional Lists

1.1 What are Multidimensional Lists?

- **Definition**: A multidimensional list is essentially a list where each element is also a list. This creates layers, or "dimensions," within the main list.
- **Examples in Daily Life**: Think of a *spreadsheet*. Each row of the spreadsheet could be represented as a list, and the entire spreadsheet itself as a 2D list. Similarly, a *Rubik's Cube* can be thought of as a 3D list, with each side containing smaller units.

1.2 Types of Multidimensional Lists

- **2D Lists (Grids/Tables)**:
  - Structured like a grid, with rows and columns.
  - **Example**: A list representing a Tic-Tac-Toe board:

    ```
    board = [
        ["X", "O", "X"],
        ["O", "X", "O"],
        ["X", " ", "O"]
    ]
    ```

- **3D Lists**:
  - Represents a collection of 2D lists.
  - **Example**: A 3D list representing a Rubik's Cube:

    ```
    cube = [
        [[1, 2], [3, 4]],  # Side 1
        [[5, 6], [7, 8]],  # Side 2
    ]
    ```

1.3 Accessing Elements in Multidimensional Lists

- **2D Lists**: Access using two indices (e.g., `list[row][column]`).
  - **Example**:

    ```
    print(board[1][2])  # Outputs 'O'
    ```

- **3D Lists**: Access using three indices.
  - **Example**:

    ```
    print(cube[0][1][1])  # Outputs '4'
    ```

1.4 Iterating through Multidimensional Lists

- **Nested Loops**: Use a loop for each dimension.
  - **Example** for 2D list:

    ```
    for row in board:
        for cell in row:
            print(cell, end=" ")
    ```

1.5 List Manipulation

- **Adding Elements**:
  - **2D Example**: `list.append(["new", "row"])`
- **Removing Elements**:
  - **2D Example**: `del list[1][2]`

1.6 Applications of Multidimensional Lists

- **Matrices for Mathematical Operations**: Useful in data analysis and scientific computing.
- **Image Processing**: A grayscale image can be represented as a 2D list of pixel values; a colored image as a 3D list (width, height, color channels).
- **Game Boards**: Structuring board games like chess or Sudoku as a multidimensional list.

Quick Trick Questions

1. **Trick Question**: What will be the output of the following code?

   ```
   board = [[1, 2, 3], [4, 5, 6]]
   print(board[0][3])
   ```

   - **Answer**: An `IndexError` because there's no fourth element in the first list.

2. **Trick Question**: How can you replace the value `5` with `9` in the board example from section 1.2?
    - **Solution**:

      ```
      board[1][1] = 9
      print(board)  # Outputs [[1, 2, 3], [4, 9, 6]]
      ```

Section 2: Functions

2.1 What are Functions?

- **Definition**: Functions are reusable blocks of code designed to perform a specific task.
- **Syntax**:

  ```
  def function_name(parameters):
      # Code block
      return value  # Optional
  ```

2.2 Defining Functions

- **Function Declaration**: Use the `def` keyword, name the function, add any necessary parameters, and then the function body.
- **Example**:

  ```
  def greet(name):
      return f"Hello, {name}!"
  ```

2.3 Function Parameters

- **Positional Parameters**: Passed in order of declaration.
- **Keyword Parameters**: Specified by name, allowing flexibility.
    - **Example**:

      ```
      def display_info(name, age=18):
          print(f"Name: {name}, Age: {age}")
      ```

2.4 Return Values

- **Single Return Value**: `return` statement allows us to send data back from the function.
- **Multiple Return Values**: A function can return multiple values as a tuple.

    - **Example**:

      ```
      def divide(a, b):
          return a // b, a % b
      ```

2.5 Lambda Functions

- **Definition**: Anonymous (nameless) functions, often used for short, single-expression functions.
    - **Example**:

      ```
      square = lambda x: x ** 2
      ```

Quick Trick Questions

1. **Trick Question**: What is the difference between `return x` and `print(x)` in a function?
    - **Answer**: `return` sends the value back to the caller, allowing further use, while `print` just displays it on the screen.
2. **Trick Question**: What will this code output?

   ```
   def add(a, b):
       return a + b
   print(add(3, add(2, 1)))
   ```

    - **Solution**: `add(2, 1)` returns `3` , so `add(3, 3)` outputs `6` .

Preview of Next Lecture: Classes and Inbuilt Functions

In the next lecture, we will dive into **Classes**, which are the foundation of Object-Oriented Programming (OOP). We will explore how to create objects, use attributes, and define methods within classes. Additionally, we'll discuss **Inbuilt Functions** in Python that make our code more efficient, helping us avoid reinventing the wheel.

These notes, along with the provided examples and trick questions, will help solidify your understanding of multidimensional lists and functions.