Chapter 6: Python Two Dimensional Lists

Contents

Red light: No AI

Time required: 120 minutes

DRY

Don't **R**epeat **Y**ourself

The Basics of 2-D Lists

```
# A simple 2D list example
simple_matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

- Think of a 2D list as a container that holds other lists
- Each inner list represents a row
- Elements at the same position across inner lists form columns

- The term "2D" comes from the two-dimensional nature of the data structure

Real-world Analogies

- Spreadsheet: Each row is a list, and the whole spreadsheet is a 2D list
- Game board: Chess or tic-tac-toe where each square has coordinates
- Digital photo: Each pixel represented by RGB values in rows and columns
- Seating chart: Rows and columns of seats in a theater or classroom

Rows: The horizontal lists

Columns: The vertical alignment of elements

Elements/Items: Individual values in the 2D list

Dimensions: Number of rows × number of columns

Indices: Row and column numbers (starting from 0)

```
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]
# Getting dimensions
num_rows = len(matrix)        # 2
num_cols = len(matrix[0])     # 3

# Accessing elements
first_element = matrix[0][0]  # 1
last_element = matrix[-1][-1] # 6
```

List Comprehensions

List comprehensions provide a concise way to create lists based on existing sequences or iterables. We will be using comprehensions to create 1 and 2D lists.

The basic syntax is:

```
new_list = [expression for item in iterable]
```

Let's compare traditional loops with list comprehensions:

```
# Traditional loop
squares = []
for x in range(5):
    squares.append(x ** 2)

# List comprehension
squares = [x ** 2 for x in range(5)]
# Result: [0, 1, 4, 9, 16]
```

You can add conditions.

```
# Get only even numbers
evens = [x for x in range(10) if x % 2 == 0]
# Result: [0, 2, 4, 6, 8]

# Numbers greater than 5
bigger_than_five = [x for x in range(10) if x > 5]
# Result: [6, 7, 8, 9]
```

You can use list comprehensions with strings.

```
# Convert to uppercase
words = ['hello', 'world', 'python']
upper_words = [word.upper() for word in words]
# Result: ['HELLO', 'WORLD', 'PYTHON']

# Get lengths of words
word_lengths = [len(word) for word in words]
# Result: [5, 5, 6]
```

You can use list comprehensions to create 2D lists.

```
# Create coordinate pairs
coords = [(x, y) for x in range(3) for y in range(2)]
# Result: [(0,0), (0,1), (1,0), (1,1), (2,0), (2,1)]

# Create a multiplication table
table = [[i * j for j in range(1, 4)] for i in range(1, 4)]
# Result: [[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

Creating 2D Lists

```
# Direct initialization
matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]

# Building row by row
rows = []
rows.append([1, 2, 3])
rows.append([4, 5, 6])
rows.append([7, 8, 9])
```

Here are a couple more examples of creating 2D lists with list comprehensions.

```
# Basic list comprehension for a matrix
n, m = 3, 4  # 3 rows, 4 columns
matrix = [[0 for j in range(m)] for i in range(n)]

# Creating multiplication table
mult_table = [[i * j for j in range(1, 6)] for i in range(1, 6)]

# Initializing with calculated values
squares = [[i**2 + j for j in range(3)] for i in range(3)]
```

Common Operations

Basic iteration.

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Iterate through all elements
for i in range(len(matrix)):
    for j in range(len(matrix[0])):
        print(f"Element at {i},{j} is {matrix[i][j]}")

# Iterate through rows
for row in matrix:
    print(row)  # prints each row list

# Iterate through columns
for j in range(len(matrix[0])):
    column = [matrix[i][j] for i in range(len(matrix))]
    print(f"Column {j}: {column}")
```

Finding dimensions and properties

```
# Get number of rows and columns
num_rows = len(matrix)
num_cols = len(matrix[0])

# Find maximum and minimum values
max_value = max(max(row) for row in matrix)
min_value = min(min(row) for row in matrix)

# Check if matrix is square
is_square = len(matrix) == len(matrix[0])

# Calculate sum of all elements
total_sum = sum(sum(row) for row in matrix)
```

Diagonal Operations

Main Diagonal (Top-Left to Bottom-Right)

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Get main diagonal elements
main_diagonal = [matrix[i][i] for i in range(len(matrix))]
# Result: [1, 5, 9]

# Alternative using traditional loop
def get_main_diagonal(matrix):
    diagonal = []
    for i in range(len(matrix)):
        diagonal.append(matrix[i][i])
    return diagonal
```

Secondary Diagonal (Top-Right to Bottom-Left)

```
# Get secondary diagonal elements
n = len(matrix)
secondary_diagonal = [matrix[i][n-1-i] for i in range(n)]
# Result: [3, 5, 7]

# Alternative using traditional loop
def get_secondary_diagonal(matrix):
    diagonal = []
    n = len(matrix)
    for i in range(n):
        diagonal.append(matrix[i][n-1-i])
    return diagonal
```

Game board analysis such as Tic Tac Toe.

```
def check_diagonal_win(board):
    """Check if there's a win on either diagonal"""
    n = len(board)

    # Check main diagonal
    if all(board[i][i] == board[0][0] and board[0][0] != ' '
          for i in range(n)):
        return True

    # Check secondary diagonal
    if all(board[i][n-1-i] == board[0][n-1] and board[0][n-1] != ' '
          for i in range(n)):
        return True

    return False
```

Common Algorithms and Operations

Problem:

Find the maximum and minimum values in a list or matrix.

Example:

```
# List example
numbers = [3, 1, 7, 9, 2]
max_value = max([num for num in numbers])
min_value = min([num for num in numbers])

print(f"Max: {max_value}, Min: {min_value}")

# Matrix example
matrix = [
    [3, 5, 9],
    [1, 7, 2],
    [4, 8, 6]
]
max_in_matrix = max([max(row) for row in matrix])
min_in_matrix = min([min(row) for row in matrix])

print(f"Max in matrix: {max_in_matrix}, Min in matrix: {min_in_matrix}")
```

Problem:

Find the sum of each row or column in a matrix.

Example:

```
matrix = [
    [3, 5, 9],
    [1, 7, 2],
    [4, 8, 6]
]

# Row sums
row_sums = [sum(row) for row in matrix]
print(f"Row sums: {row_sums}")

# Column sums
col_sums = [sum(matrix[row][col] for row in range(len(matrix))) for col in
range(len(matrix[0]))]
print(f"Column sums: {col_sums}")
```

Assignment Submission

1. Attach the pseudocode or create a TODO.
2. Attach all tutorials and assignments.
3. Attach screenshots showing the successful operation of each tutorial program.
4. Submit in Blackboard.