Exercise 1: Basic List Methods

Create a list of numbers. Use list methods to append new elements, insert an element, remove a value, and pop an item from the list. Finally, print the resulting list along with the popped item.

This exercise demonstrates how to add elements at the end and at a specific position, remove an element by value, and remove the last element using pop().

Exercise 2: Changing Values in a List

Given a list of fruit names, change all the strings in the list to uppercase letters by modifying the list in place. Then, print the updated list.

This exercise practices iterating over list indices to update each element. In-place modifications are common when you need to transform list data without creating a new list.

Exercise 3: List Slicing

Using a list of numbers from 0 to 9, create slices of the list to print:

- The first five elements.
- The last three elements.
- Every other element (starting from index 0).

This exercise reinforces the concept of slicing — a powerful way to access sublists by specifying start, stop, and step values in a compact format.

Exercise 4: Counting Occurrences in a List

Given a list that includes some duplicate elements, count how many times specific items appear using the list count method, and print the counts.

This exercise shows how to use the count method to determine how many times a specific element appears in a list. It's useful for frequency comparisons or simple data analyses.

Exercise 5: Two-Dimensional List (Matrix) Manipulation

Create a two-dimensional list (matrix) that represents a 3x3 grid. Print the original matrix, then update a specific element (for instance, change the middle element to 99), and print the updated matrix.

This exercise introduces the concept of two-dimensional lists. It shows how to access and modify elements using two indices and reinforces the practice of iterating over a 2D list for display purposes.

Exercise 6: Creating Tuples with Parentheses

Write a program that creates a tuple of four integers using parentheses. Then, print both the tuple and its type.

Exercise 7: Creating Tuples Without Parentheses

Write a program that creates a tuple by listing values separated by commas (i.e., without using parentheses). Also, create a single-element tuple (don't forget the trailing comma). Print their values and types.

Exercise 8: Adding to a Tuple via Concatenation

Since tuples are immutable and individual elements cannot be changed, you cannot append to a tuple directly. Instead, demonstrate how to "add" an element by concatenating another tuple.

Exercise 9: Demonstrating Tuple Immutability

Try to change an element of a tuple. This exercise demonstrates that tuples cannot be modified once created.

Exercise 10: Transforming a Tuple into a List and Back

Since tuples are immutable, sometimes you might need to update them. Write a program that converts a tuple to a list to add or modify an element, and then convert it back to a tuple.

Solutions

Exercise 1: Basic List Methods

Example Solution:

```
# Create a list of numbers
numbers = [1, 2, 3, 4, 5]

# Append 6 to the end of the list
numbers.append(6)

# Insert value 0 at the beginning of the list (index 0)
numbers.insert(0, 0)

# Remove the value 3 from the list
numbers.remove(3)

# Pop the last item from the list and store it in a variable
last_item = numbers.pop()

# Print the modified list and the popped item
print("Modified list:", numbers)
print("Last popped item:", last_item)
```

Exercise 2: Changing Values in a List

Example Solution:

```
# Initial list of fruit names
fruits = ['apple', 'banana', 'cherry']

# Convert each fruit name to uppercase by modifying the list in-place
for i in range(len(fruits)):
    fruits[i] = fruits[i].upper()

# Print the updated list
print("Fruits in uppercase:", fruits)
```

Exercise 3: List Slicing

Example Solution:

```
# Create a list of numbers from 0 to 9
numbers = list(range(10))

# Slice for the first five elements
first_five = numbers[:5]

# Slice for the last three elements
last_three = numbers[-3:]

# Slice for every other element, starting at index 0
every_other = numbers[::2]

# Print the slices
print("First five elements:", first_five)
print("Last three elements:", last_three)
print("Every other element:", every_other)
```

Exercise 4: Counting Occurrences in a List

Example Solution:

```
# List containing duplicate elements
colors = ['red', 'blue', 'green', 'red', 'blue', 'red']

# Count the occurrences of each color
red_count = colors.count('red')
blue_count = colors.count('blue')
green_count = colors.count('green')

# Print the counts for each color
print("Count of red:", red_count)
print("Count of blue:", blue_count)
print("Count of green:", green_count)
```

Exercise 5: Two-Dimensional List (Matrix) Manipulation

Example Solution:

```
# Define a 3x3 matrix (two-dimensional list)
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Print the original matrix
print("Original Matrix:")
for row in matrix:
    print(row)

# Modify the middle element (row index 1, column index 1)
matrix[1][1] = 99

# Print the modified matrix
print("\nModified Matrix:")
for row in matrix:
    print(row)
```

Exercise 6: Creating Tuples with Parentheses

Example Code:

```
# Creating a tuple using parentheses
tuple_with_parentheses = (1, 2, 3, 4)
print("Tuple created with parentheses:", tuple_with_parentheses)
print("Type of tuple_with_parentheses:", type(tuple_with_parentheses))
```

Exercise 7: Creating Tuples Without Parentheses

Example Code:

```
# Creating a tuple without parentheses
tuple_without_parentheses = 5, 6, 7, 8
```

```
print("Tuple without parentheses:", tuple_without_parentheses)
print("Type of tuple_without_parentheses:", type(tuple_without_parentheses))

# Creating a single-element tuple
single_element_tuple = (9,)  # The comma is essential here!
print("Single-element tuple:", single_element_tuple)
print("Type of single_element_tuple:", type(single_element_tuple))
```

Exercise 8: Adding to a Tuple via Concatenation

Example Code:

```
# Initial tuple
initial_tuple = (10, 20, 30)
print("Initial tuple:", initial_tuple)

# "Adding" an element by creating a new tuple and concatenating
new_tuple = initial_tuple + (40,)
print("Tuple after concatenation:", new_tuple)
```

Exercise 9: Demonstrating Tuple Immutability

Example Code:

```
# Creating a tuple
immutable_tuple = (100, 200, 300)
print("Immutable tuple:", immutable_tuple)
# Attempting to change an element
immutable_tuple[0] = 999
```

Exercise 10: Transforming a Tuple into a List and Back

```
# Original tuple
original_tuple = (1, 2, 3)
print("Original tuple:", original_tuple)

# Convert the tuple to a list
temp_list = list(original_tuple)
print("Converted to list:", temp_list)
```

```
# Modify the list by appending an element
temp_list.append(4)
print("List after appending:", temp_list)

# Convert the list back to a tuple
modified_tuple = tuple(temp_list)
print("Modified tuple:", modified_tuple)
```