Lecture 1.3

Python Specific Data Structures

Agenda

- 1. Python Lists
- 2. Python Tuples
- 3. Python Sets
- 4. Python Dictionaries
- 5. Arrays in Python (array module)
- 6. NumPy Arrays
- 7. Arrays vs Lists

Introduction to Python Built-in Data Structures

- Python provides four built-in data structures:
- List: Ordered and mutable.
- Tuple: Ordered and immutable.
- Set: Unordered and unindexed.
- Dictionary: Unordered (ordered in newer versions), mutable, and indexed by keys.

Lists in Python

- A list is created using square brackets: my_list = [1, 2, 3]
- Lists are ordered, meaning elements have a specific position.
- Lists are mutable: we can change, add, or remove items after creation.
- Can contain items of different data types.

List Example

- my_list = [10, 20, 'Python', 3.14]
- my_list.append(40) # Adds 40 to the end
- my_list[1] = 25 # Modifies the second element
- print(my_list) # Output: [10, 25, 'Python', 3.14, 40]

Tuples in Python

- A tuple is created using parentheses: my_tuple = (1, 2, 3)
- Tuples are ordered, just like lists.
- Tuples are immutable: once created, items cannot be changed, added, or removed.
- Often used for data that shouldn't change, like coordinates.

Tuple Example

- point = (10, 20)
- print(point[0]) # Accessing first element: 10
- # point[0] = 15 # This would raise a TypeError

Sets in Python

- A set is created using curly braces: my_set = {1, 2, 3}
- Sets are unordered, so items don't have a defined order or index.
- Sets do not allow duplicate values.
- Useful for membership testing and eliminating duplicate entries.

Set Example

- my_set = {1, 2, 2, 3, 4}
- print(my_set) # Output: {1, 2, 3, 4}
- my_set.add(5)
- my_set.remove(2)

Dictionaries in Python

- Created using curly braces with key:value pairs: my_dict = {'key': 'value'}
- Dictionaries are mutable and store data values in key:value pairs.
- Keys must be unique and immutable (e.g., strings, numbers, tuples).
- Optimized for retrieving data.

Dictionary Example

- student = {'name': 'Alice', 'age': 20, 'course': 'IT'}
- print(student['name']) # Output: Alice
- student['age'] = 21 # Modifying value
- student['grade'] = 'A' # Adding new key-value pair

Arrays in Python

- Unlike Lists, Arrays store elements of the same data type.
- Python does not have built-in support for Arrays, but we can use the 'array' module.
- To use arrays, we must import the module: import array
- Useful when you need to manipulate only specific data types.

Using the array Module

- import array as arr
- numbers = arr.array('i', [1, 2, 3, 4, 5]) # 'i' specifies integer type
- print(numbers[0]) # Accessing elements
- numbers.append(6)

Introduction to NumPy Arrays

- NumPy (Numerical Python) is a library for working with arrays.
- Provides an array object that is up to 50x faster than traditional Python lists.
- To use NumPy, you must import it: import numpy as np
- Supports multi-dimensional arrays and matrices.

Using NumPy Arrays

- import numpy as np
- arr = np.array([1, 2, 3, 4, 5])
- print(arr)
- print(arr * 2) # Vectorized operation: Output [2 4 6 8 10]

Operations on Arrays

- Traversal: Iterating through elements.
- Insertion & Deletion: Adding or removing elements.
- Searching: Finding an element by index or value.
- Updating: Modifying an existing element.
- Mathematical operations (especially with NumPy).

Arrays vs Lists

- Data Types: Lists can contain different types; Arrays contain elements of the same type.
- Memory: Arrays are more memory efficient.
- Operations: NumPy arrays support element-wise mathematical operations directly.
- Usage: Use lists for general purposes, arrays/NumPy for numerical computations.

Summary

- Python offers Lists (mutable, mixed), Tuples (immutable, mixed), Sets (unique), and Dictionaries (key-value).
- Arrays store elements of the same data type and require importing 'array' or 'numpy'.
- NumPy provides powerful, optimized multi-dimensional arrays.
- Arrays are more memory-efficient and suited for mathematical operations compared to Lists.

Next Lecture Preview

- Transition to Object-Oriented Programming (OOP).
- Understanding Classes and Objects.
- Constructors in Python.