DATA STRUCTURES - I

What are Data Structures? 🤓

Data structures are specific ways of organizing and storing data in a computer so it can be accessed and used efficiently. Think of them as different types of containers, each designed for a specific purpose. Using the right data structure can make your programs faster and more memory-efficient.
The main reasons we use them are for:

- **Organization**: To arrange data logically.
- **Efficiency**: To speed up tasks like searching, adding, and deleting data.
- **Memory Management**: To use computer memory effectively.

Python's four most common built-in data structures are **Lists**, **Tuples**, **Sets**, and **Dictionaries**. This summary covers Lists and Tuples.

Lists 📝

A **list** is an ordered and changeable collection of items that can be of any data type. Because they are ordered, the items maintain a defined sequence, and because they are **mutable**, you can add, remove, or change items after the list has been created. Lists also allow duplicate values.

Creating and Working with Lists

Python
```python
# Creating lists
empty_list = []
fruits = ["apple", "banana", "cherry", "apple"] # Allows duplicates
mixed_list = [1, "hello", 3.14]

# Accessing items with indexing and slicing
print(fruits[0])       # Output: apple
print(fruits[-1])      # Output: apple
print(fruits[1:3])     # Output: ['banana', 'cherry']

# Getting the length
print(len(fruits))     # Output: 4
```

Key List Methods

- **Adding Items**
  - .append(item): Adds a single item to the end of the list.
  - .extend(another_list): Adds all items from another list to the end.
  - .insert(index, item): Inserts an item at a specific position.

  Python
  ```python
  fruits.append("orange")
  print(fruits) # Output: ['apple', 'banana', 'cherry', 'apple', 'orange']
  ```
- **Removing Items**
  - .remove(item): Removes the *first* occurrence of a specific item.
  - .pop(index): Removes and returns the item at a specific index. If no index is given, it removes the last item.
  - .clear(): Removes all items from the list, making it empty.
  - del my_list[index]: Deletes the item at a specific index.

Python
```
fruits.pop(1) # Removes 'banana'
print(fruits) # Output: ['apple', 'cherry', 'apple', 'orange']
```

- **Organizing Lists**
  - `.sort()`: Sorts the list in place (alphabetically or numerically). Use `sort(reverse=True)` for descending order.
  - `.reverse()`: Reverses the order of the items in the list in place.

Python
```
numbers = [3, 1, 4, 2]
numbers.sort()
print(numbers) # Output: [1, 2, 3, 4]
```

- **Other Useful Methods**
  - `.copy()`: Returns a shallow copy of the list.
  - `.count(item)`: Returns the number of times an item appears in the list.
  - `.index(item)`: Returns the index of the first occurrence of an item.

Tuples 📦

A **tuple** is an ordered collection of items, very similar to a list. The key difference is that tuples are **immutable**, meaning once a tuple is created, you cannot change, add, or remove its elements. Tuples are often used for fixed data that shouldn't be modified.

Creating and Working with Tuples

Tuples are created with parentheses `()`. For a tuple with a single item, you must include a trailing comma.

Python
```
# Creating tuples
my_tuple = (1, "hello", 3.14)
single_item_tuple = ("apple",) # The comma is essential!

# Accessing items works just like with lists
print(my_tuple[1])    # Output: hello
print(my_tuple[:2])   # Output: (1, 'hello')

# Getting the length
print(len(my_tuple))   # Output: 3
```

Modifying Tuples (The Workaround)

Since tuples are immutable, you can't change them directly. The common workaround is to convert the tuple to a list, modify the list, and then convert it back into a tuple.

Python
```
fruits_tuple = ("apple", "banana", "cherry")

# Convert to list, modify, then convert back
temp_list = list(fruits_tuple)
temp_list.append("orange")
fruits_tuple = tuple(temp_list)

print(fruits_tuple) # Output: ('apple', 'banana', 'cherry', 'orange')
```

Unpacking Tuples

You can assign the items of a tuple to multiple variables in a single line. This is called "unpacking."

```
Python
person = ("John", 30, "USA")
(name, age, country) = person

print(name) # Output: John
print(age)  # Output: 30
```

Comparison: Lists vs. Tuples

Feature	Lists	Tuples
Mutability	**Mutable** (can be changed)	**Immutable** (cannot be changed)
Syntax	Created with square brackets `[]`	Created with parentheses `()`
Performance	Slightly slower, use more memory	Faster and more memory-efficient
Use Case	For collections that need to be modified	For fixed data that should not be modified