Data Structures in Python: Complex Data Types- PART 1

John Karuitha, PhD

February 27, 2025

1 1. Lists (`list`)

1.1 Definition:

A **list** is an ordered, mutable collection of items. Lists can contain elements of different data types (integers, floats, strings, etc.) and can be changed after they are created (mutable). Lists are written with square brackets `[]`.

1.2 Example:

```python
fruits = ["apple", "banana", "cherry"]
numbers = [1, 2, 3, 4, 5]
mixed = ["hello", 42, 3.14, True]
```

In the above example:

- `fruits` is a list of strings.
- `numbers` is a list of integers.
- `mixed` is a list containing different data types.

1.3 Methods and Operations for Lists:

- **Accessing Elements:** You can access individual elements in a list using their index. Indexing starts from `0`.

```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])  # Output: apple
print(fruits[2])  # Output: cherry
```

- **Negative Indexing:** You can also access list elements using negative indices, where `-1` refers to the last element, `-2` to the second-to-last, and so on.

```
print(fruits[-1])  # Output: cherry (last element)
print(fruits[-2])  # Output: banana (second-to-last element)
```

- **Modifying Elements:** Lists are mutable, meaning you can change the value of elements after the list is created.

```
fruits[1] = "orange"
print(fruits)  # Output: ['apple', 'orange', 'cherry']
```

- **Appending and Extending:** You can add elements to the end of a list using `append()`, or add multiple elements with `extend()`.

```
fruits.append("mango")
print(fruits)  # Output: ['apple', 'orange', 'cherry', 'mango']

fruits.extend(["pear", "grape"])
print(fruits)  # Output: ['apple', 'orange', 'cherry', 'mango', 'pear', 'grape']
```

- **Inserting Elements:** The `insert()` method allows you to add an element at a specific index.

```
fruits.insert(1, "blueberry")
print(fruits)  # Output: ['apple', 'blueberry', 'orange', 'cherry', 'mango', 'pear', 'gr
```

- **Removing Elements:** You can remove elements from a list using methods like `remove()` (removes the first occurrence of a value) or `pop()` (removes by index).

```
fruits.remove("orange")
print(fruits)  # Output: ['apple', 'blueberry', 'cherry', 'mango', 'pear', 'grape']

fruits.pop(2)
print(fruits)  # Output: ['apple', 'blueberry', 'mango', 'pear', 'grape']
```

- **Slicing Lists:** You can access parts of a list by using slicing. Slicing allows you to retrieve a range of elements using the syntax `my_list[start:end:step]`.

```
fruits = ["apple", "banana", "cherry", "mango", "pear", "grape"]

# Basic slicing
print(fruits[1:4])  # Output: ['banana', 'cherry', 'mango']

# Using a step value
print(fruits[0:5:2])  # Output: ['apple', 'cherry', 'pear'] (selects every second elemen

# Omitting start and end values with a step
print(fruits[::2])  # Output: ['apple', 'cherry', 'pear'] (all elements, every second or
```

- **Checking Membership:** Use the `in` operator to check if an element is in the list.

```
print("apple" in fruits)  # Output: True
print("orange" in fruits)  # Output: False
```

- **`len()` Function:** Returns the number of elements in the list.

```
print(len(fruits))  # Output: 5
```

---

2 2. Tuples (`tuple`)

2.1 Definition:

A **tuple** is similar to a list, but it is immutable, meaning its elements cannot be changed after creation. Tuples are defined using parentheses `()` instead of square brackets.

2.2 Example:

```python
colors = ("red", "green", "blue")
numbers = (1, 2, 3)
```

In the above example: - `colors` is a tuple of strings. - `numbers` is a tuple of integers.

2.3 Methods and Operations for Tuples:

- **Accessing Elements:**

Like lists, you can access elements of a tuple using indexing.

```python
print(colors[0])  # Output: red
print(numbers[2])  # Output: 3
```

- **Immutable Nature:**

  Tuples cannot be modified after they are created. If you try to change an element, Python will raise an error.

  ```python
  # This will raise an error
  # colors[0] = "yellow"  # Error: 'tuple' object does not support item assignment
  ```

- **Tuple Slicing:**

Just like lists, you can slice tuples.

```python
print(colors[:2])  # Output: ('red', 'green')
```

- **Unpacking Tuples:**

You can assign elements of a tuple to multiple variables at once.

```python
red, green, blue = colors
print(red)   # Output: red
print(green) # Output: green
print(blue)  # Output: blue
```

- **Checking Membership:**

Use the `in` operator to check if an element exists in the tuple.

```
print("blue" in colors)  # Output: True
print("yellow" in colors)  # Output: False
```

- **`len()` Function:**

Returns the number of elements in the tuple.

```
print(len(colors))  # Output: 3
```

---

3 Key Differences Between Lists and Tuples:

Feature	Lists	Tuples
Mutability	Mutable (can be changed)	Immutable (cannot be changed)
Syntax	Defined with `[]`	Defined with `()`
Methods	More methods like `append()`, `remove()`, etc.	Fewer methods due to immutability
Use Cases	Used when elements may change over time	Used when elements should remain constant

---

4 Conclusion

Both lists and tuples are essential data structures in Python, but they serve different purposes. **Lists** are ideal when you need a collection of items that might change, while **tuples** are useful for collections that should remain constant throughout the program. Understanding when to use lists versus tuples, and how to manipulate these data types, is crucial for efficient Python programming.

References