Data Structures in Python

Difference between Different Data Structures

Contents:

- Lists and its inbuilt methods like
  - append()
  - extend()
  - insert()
  - max()
  - min()
  - pop()
  - len()
- Sets
  - intersection()
  - difference()
  - add()
- Dictionaries
- Tuples
  - count()
  - index()

1. Lists

- **Ordered**: Elements in a list are ordered by their position, and you can access elements by their index.
- **Mutable**: You can change, add, or remove elements after the list is created.
- **Duplicate Elements**: Lists can contain duplicate elements.
- **Syntax**: Defined using square brackets `[]`.

```python
# Example of a list
my_list = [1, 2, 3, 4, 5]
```

2. Dictionaries

- **Key-Value Pairs**: Dictionaries store data as key-value pairs.
- **Unordered**: Elements are not stored in a specific order (insertion order is preserved in Python 3.7+).
- **Mutable**: You can change, add, or remove key-value pairs after the dictionary is created.
- **Unique Keys**: Keys must be unique and immutable (e.g., strings, numbers, tuples).
- **Syntax**: Defined using curly braces `{}` with colons separating keys and values.

```python
# Example of a dictionary
my_dict = {"name": "Alice", "age": 25, "city": "New York"}
```

3. Sets

- **Unordered**: Elements in a set are not stored in a specific order.
- **Mutable**: You can add or remove elements from a set.
- **Unique Elements**: Sets do not allow duplicate elements.
- **Syntax**: Defined using curly braces `{}` or the `set()` function.

```
# Example of a set
my_set = {1, 2, 3, 4, 5}
```

4. Tuples

- **Ordered**: Elements in a tuple are ordered by their position, similar to lists.
- **Immutable**: Once a tuple is created, you cannot change, add, or remove elements.
- **Duplicate Elements**: Tuples can contain duplicate elements.
- **Syntax**: Defined using parentheses `()`.

```
# Example of a tuple
my_tuple = (1, 2, 3, 4, 5)
```

Summary of Differences

Feature	List	Dictionary	Set	Tuple
Order	Ordered	Unordered (ordered in 3.7+)	Unordered	Ordered
Mutability	Mutable	Mutable	Mutable	Immutable
Duplicates	Allows duplicates	Unique keys	No duplicates	Allows duplicates
Access Method	Index-based	Key-based	Value-based	Index-based
Syntax	`[]`	`{}` with key-value pairs	`{}` or `set()`	`()`

Use Cases

- **List**: When you need an ordered collection that may contain duplicates and you need to frequently modify the collection.
- **Dictionary**: When you need to associate values with unique keys for fast lookups, updates, and deletions.
- **Set**: When you need a collection of unique elements and you need to perform fast membership tests or set operations like union, intersection, etc.
- **Tuple**: When you need an ordered collection that should not change after creation, often used for fixed collections of items.

Choosing the appropriate data structure depends on your specific needs regarding order, mutability, uniqueness, and performance characteristics.