

# Data Structures in Python

---

Python provides powerful built-in data structures to store and manipulate collections of data efficiently.

---

## 1. Lists and List Methods

---

Lists are ordered, mutable (changeable) collections of items.

### Creating a List:

```
numbers = [1, 2, 3, 4, 5]
mixed = [10, "hello", 3.14]
```

### Common List Methods:

```
my_list = [1, 2, 3]

my_list.append(4)    # [1, 2, 3, 4]
my_list.insert(1, 99) # [1, 99, 2, 3, 4]
my_list.remove(2)    # [1, 99, 3, 4]
my_list.pop()        # Removes last element -> [1, 99, 3]
my_list.reverse()    # [3, 99, 1]
my_list.sort()        # [1, 3, 99]
```

### List Comprehensions (Efficient List Creation)

```
squared = [x**2 for x in range(5)]
print(squared) # Output: [0, 1, 4, 9, 16]
```

---



## 2. Tuples and Operations on Tuples

---

Tuples are ordered but **immutable** collections (cannot be changed after creation).

### Creating a Tuple:

```
my_tuple = (10, 20, 30)
single_element = (5,) # Tuple with one element (comma required)
```

### Accessing Tuple Elements:

```
print(my_tuple[1]) # Output: 20
```

### Tuple Unpacking:

```
a, b, c = my_tuple
print(a, b, c) # Output: 10 20 30
```

### Common Tuple Methods:

Method	Description	Example	Output
<code>count(x)</code>	Returns the number of times <code>x</code> appears in the tuple	<code>(1, 2, 2, 3).count(2)</code>	<code>2</code>
<code>index(x)</code>	Returns the index of the first occurrence of <code>x</code>	<code>(10, 20, 30).index(20)</code>	<code>1</code>

```
my_tuple = (1, 2, 2, 3, 4)
print(my_tuple.count(2)) # Output: 2

print(my_tuple.index(3)) # Output: 3
```



## Why Use Tuples?

- Faster than lists (since they are immutable)
- Used as dictionary keys (since they are hashable)
- Safe from unintended modifications

## 3. Sets and Set Methods

---

Sets are **unordered, unique collections** (no duplicates).

### Creating a Set:

```
fruits = {"apple", "banana", "cherry"}
```

### Key Set Methods:

```
my_set = {1, 2, 3, 4}

my_set.add(5)          # {1, 2, 3, 4, 5}
my_set.remove(2)       # {1, 3, 4, 5}
my_set.discard(10)     # No error if element not found
my_set.pop()           # Removes random element
```

### Set Operations:

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a.union(b))      # {1, 2, 3, 4, 5}
print(a.intersection(b)) # {3}
print(a.difference(b))  # {1, 2}
```

**Use Case:** Sets are great for eliminating duplicate values.

---



## 4. Dictionaries and Dictionary Methods

---

Dictionaries store key-value pairs and allow fast lookups.

### Creating a Dictionary:

```
student = {"name": "Alice", "age": 21, "grade": "A"}
```

### Accessing & Modifying Values:

```
print(student["name"]) # Output: Alice
student["age"] = 22    # Updating value
student["city"] = "New York" # Adding new key-value pair
```

### Common Dictionary Methods:

```
print(student.keys())    # dict_keys(['name', 'age', 'grade', 'ci
print(student.values())  # dict_values(['Alice', 22, 'A', 'New Yo
print(student.items())   # dict_items([('name', 'Alice'), ('age',

student.pop("age") # Removes "age" key
student.clear()    # Empties dictionary
```

### Dictionary Comprehensions:

```
squares = {x: x**2 for x in range(5)}
print(squares) # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

## 5. When to Use Each Data Structure?

---

Data Structure	Features	Best For
List	Ordered, Mutable	Storing sequences, dynamic data
Tuple	Ordered, Immutable	Fixed collections, dictionary keys



Data Structure	Features	Best For
Set	Unordered, Unique	Removing duplicates, set operations
Dictionary	Key-Value Pairs	Fast lookups, structured data