Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

Collection Type	Description	Mutable	Ordered	Duplicate Elements
`List[]`	Ordered, changeable, allows duplicates	✅	✅	✅
`Tuple()`	Ordered, unchangeable (immutable)	❌	✅	✅
`Set{}`	Unordered, no duplicates, changeable	✅	❌	❌
`Dict{}`	Key-value pairs, changeable	✅	✅	❌ (unique keys)

Note:

- Set itself is mutable, the elements contained within a set must be immutable. For example, you can add numbers, strings, or tuples to a set, but you cannot add mutable objects like lists or other sets directly as elements within a set. Also, you cannot modify the elements of sets.
- Set are unordered , so you cannot be sure in which order the items will appear.

1. List

A `list` is an **ordered**, **mutable** (changeable) collection.

Syntax:

```
my_list = ["apple", "banana", "cherry"]
print(my_list[0])          # apple
my_list.append("mango")    # Add item at last
my_list.insert (1,"mango")    # Add item at index 1
my_list.[0] = 'grapes'    # Replace item of index 0
my_list[1:2] = ['grapes'] # Replace item of index 1
my_list[1:3] = ['grapes','pear'] #Replace item of index 1 and 2 excluding 3
my_list.remove("banana")  # Remove specified item
my_list.pop ()  # Remove last item
my_list.pop (1)  # Remove specified index
my_list.clear ()  # Empties all items in list
```

2. Tuple

A `tuple` is like a list but **immutable** (cannot be changed).

Syntax:

```
my_tuple = ("apple", "banana", "cherry")
print(my_tuple[1])  # banana


# my_tuple[1] = "mango" ❌ Error: Tuples are immutable
```

But,You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
 print(x)
```

3. Set

A `set` is an **unordered**, **mutable** collection that does **not allow duplicates**.

Syntax:

```
my_set = {"apple", "banana", "apple"}
print(my_set)  # {'banana', 'apple'} — no duplicates
 my_set.add("mango")
my_set.remove("banana")
```

Joining Sets

Joining sets means combining **two or more sets** into **one**, using **set operations**. Since sets **don't allow duplicates**, the result automatically removes repeated elements.

Let's say we have:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
```

1. Union (| or .union()) : Combines elements from both sets, without duplicates.

```
result = set1.union(set2)
# OR result = set1 | set2

print(result)  # Output: {1, 2, 3, 4, 5}
```

2. Update (.update()): Adds elements of another set into the first set (modifies in place).

```
set1.update(set2)
print(set1)  # Output: {1, 2, 3, 4, 5}
```

3. Intersection (& or .intersection()) : Returns only elements **common to both sets**.

```
result = set1 & set2
# OR result = set1.intersection(set2)

print(result)  # Output: {3}
```

---

4. Difference (- or .difference()) : Returns items **only in the first set**, not in the second.

```
result = set1 - set2
# OR result = set1.difference(set2)
 print(result)  # Output: {1, 2}
```

5. Symmetric Difference (^ or .symmetric_difference()) : Keep only the elements that are not present in both sets.

```
result = set1 ^ set2
# OR result = set1.symmetric_difference(set2)

print(result)  # Output: {1, 2, 4, 5}
```

4. Dictionary (dict)

A `dict` stores data in **key-value** pairs.

Syntax:

```
my_dict     =     {
"name": "Raj",
    "age": 24,
    "country": "Nepal"
}  print(my_dict["name"])        # Raj
my_dict["age"] = 25           #Update value
my_dict["email"] = "raj@example.com"  # Add new key-value
```

Summary Table

Type	Example	Use Case
`list`	`["red", "green", "blue"]`	Order matters, can change items
`tuple`	`("sun", "moon", "stars")`	Fixed data (e.g., coordinates)
`set`	`{"apple", "banana"}`	Unique items, fast lookup
`dict`	`{"name": "Raj", "age": 24}`	Key-value storage