Day 5: Tuples and Sets

I. Tuples

- Definition:
    - An ordered, immutable collection of elements.
    - Immutable means you cannot change the elements within a tuple after it's created.
    - Enclosed in parentheses ().
- Key Characteristics:
    - Ordered: Elements have a specific position within the tuple.
    - Immutable: Cannot be modified (no append(), remove(), etc.).
    - Heterogeneous: Can hold elements of different data types (integers, strings, floats, etc.).
    - Used for:
        - Representing fixed data, such as coordinates (x, y) or database records.
        - Improving code efficiency in some cases, as tuples are generally faster than lists.
- Example:

Python

```
my_tuple = (1, 2.5, "hello", True)

print(my_tuple)

print(my_tuple[0])  # Accessing the first element
```

- Important Methods:
    - count(): Returns the number of occurrences of a specified value.
    - index(): Returns the index of the first occurrence of a specified value.[1]

Tuples in Python have **limited built-in methods** since they are **immutable** (i.e., they cannot be changed after creation).

1. Built-in Tuple Methods

Python tuples provide only two methods:

1.1 count() - Counts occurrences of an element in a tuple

```
my_tuple = (1, 2, 3, 1, 4, 1)

count_ones = my_tuple.count(1)

print(count_ones)  # Output: 3
```

- Returns the number of times a specified value appears in the tuple.

1.2 index() - Finds the first occurrence of a value

```
my_tuple = (10, 20, 30, 40, 20)

index_20 = my_tuple.index(20)

print(index_20)  # Output: 1
```

- Returns the **first index** of the specified value.
- Raises a **ValueError** if the value is not found.

---

2. Other Tuple Operations

Even though tuples are immutable, we can **manipulate them indirectly** using various techniques.

2.1 Concatenation (+) - Joining Two Tuples

```
tuple1 = (1, 2, 3)

tuple2 = (4, 5, 6)

result = tuple1 + tuple2

print(result)  # Output: (1, 2, 3, 4, 5, 6)
```

- Creates a **new tuple** by combining two tuples.

2.2 Repetition (*) - Repeating Tuple Elements

```
my_tuple = ("A", "B") * 3
print(my_tuple)  # Output: ('A', 'B', 'A', 'B', 'A', 'B')
```

- Creates a **new tuple** by repeating elements.

2.3 Slicing ([:]) - Extracting Elements

```
my_tuple = (10, 20, 30, 40, 50)
print(my_tuple[1:4])   # Output: (20, 30, 40)
print(my_tuple[:3])    # Output: (10, 20, 30)
print(my_tuple[-2:])   # Output: (40, 50)
```

- Returns a **new tuple** containing the selected range.

2.4 Unpacking - Assigning Tuple Values to Variables

```
my_tuple = ("Apple", "Banana", "Cherry")
fruit1, fruit2, fruit3 = my_tuple
print(fruit1)  # Output: Apple
```

- Allows extracting tuple values into separate variables.

2.5 Length (len()) - Finding Tuple Size

```
my_tuple = (1, 2, 3, 4, 5)
print(len(my_tuple))  # Output: 5
```

- Returns the **number of elements** in a tuple.

2.6 Checking Membership (in, not in)

```
my_tuple = (1, 2, 3, 4)

print(3 in my_tuple)   # Output: True

print(5 not in my_tuple)  # Output: True
```

- Checks if an **element exists** in the tuple.

2.7 Looping Through a Tuple (for loop)

```
my_tuple = ("Python", "Java", "C++")

for item in my_tuple:
  print(item)
```

- Iterates through each element in the tuple.

2.8 Sorting a Tuple (Using sorted())

```
my_tuple = (4, 2, 8, 1)

sorted_tuple = tuple(sorted(my_tuple))  # Sorting returns a list, so we convert it back

print(sorted_tuple)  # Output: (1, 2, 4, 8)
```

- Returns a **sorted list**; convert it back to a tuple if needed.

2.9 Finding Minimum (min()) and Maximum (max())

```
my_tuple = (10, 5, 20, 15)

print(min(my_tuple))  # Output: 5

print(max(my_tuple))  # Output: 20
```

- Finds the **smallest** and **largest** element in a tuple.

2.10 Finding Sum (sum())

```
my_tuple = (1, 2, 3, 4, 5)

print(sum(my_tuple))  # Output: 15
```

- Returns the **sum** of all elements (works only for numeric values).

2.11 Converting a List to a Tuple (tuple())

```
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(my_tuple)  # Output: (1, 2, 3)
```

- Converts a **list** into a **tuple**.

2.12 Converting a Tuple to a List (list())

```
my_tuple = (10, 20, 30)
my_list = list(my_tuple)
my_list.append(40)  # Now we can modify it
print(my_list)  # Output: [10, 20, 30, 40]
```

- Allows modification by **converting to a list** first.

Summary of Tuple Methods & Operations

Method/Operation	Description	Example
count(x)	Counts occurrences of x	my_tuple.count(1)
index(x)	Finds the first index of x	my_tuple.index(20)
+ (Concatenation)	Joins two tuples	(1, 2) + (3, 4)
* (Repetition)	Repeats elements	("A",) * 3
[:] (Slicing)	Extracts part of a tuple	my_tuple[1:4]
len()	Finds the length of a tuple	len(my_tuple)
in / not in	Checks membership	3 in my_tuple
for loop	Iterates over tuple	for item in my_tuple:

Method/Operation	Description	Example
sorted()	Returns a sorted list	sorted(my_tuple)
min()	Finds the smallest element	min(my_tuple)
max()	Finds the largest element	max(my_tuple)
sum()	Returns the sum of elements	sum(my_tuple)
tuple()	Converts a list to a tuple	tuple(my_list)
list()	Converts a tuple to a list	list(my_tuple)

Note :

Tuples **do not have** many built-in methods because they are **immutable**.

- You can still perform various operations **without modifying** the original tuple.
- If modifications are needed, **convert the tuple into a list** first.

Use tuples when you need an immutable, fast, and memory-efficient data structure!

II. Sets

- Definition:
  - An unordered collection of unique elements.
  - No duplicates allowed.
  - Enclosed in curly braces {} (empty set: set()).
- Key Characteristics:
  - Unordered: No specific order is maintained.
  - Unique: Only one instance of each element is allowed.
  - Mutable: Elements can be added or removed.
  - Used for:
    - Membership testing (checking if an element exists quickly).
    - Removing duplicates from a list.

    - Set operations (union, intersection, difference).

- Example:

Python

```
my_set = {1, 2, 2, 3, "hello"}

print(my_set)  # Output: {1, 2, 3, 'hello'}
```

- Set Operations:
    - Union: | or union(): Combines elements from both sets.
    - Intersection: & or intersection(): Finds elements common to both sets.
    - Difference: - or difference(): Finds elements in the first set but not in the second.
    - Symmetric Difference: ^ or symmetric_difference(): Finds elements in either set, but not in both.

Python Set Methods and Operations with Examples

Unlike tuples, **sets in Python are mutable** and provide various methods for manipulation. Sets are **unordered collections** of unique elements, meaning they do **not allow duplicates**.

---

1. Creating a Set

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

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

- **Sets do not allow duplicate values**.
- They are **unordered**, meaning elements do not maintain a specific order.

---

2. Built-in Set Methods

Python provides several built-in methods for working with sets.

2.1 add() - Adds an element to the set

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

my_set.add(4)

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

- Adds a **single element** to the set.

2.2 update() - Adds multiple elements to the set

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

my_set.update([4, 5, 6])

print(my_set)  # Output: {1, 2, 3, 4, 5, 6}
```

- Adds multiple values (from lists, tuples, sets, etc.).

2.3 remove() - Removes a specific element (raises an error if not found)

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

my_set.remove(2)

print(my_set)  # Output: {1, 3}
```

- Raises a **KeyError** if the element is not found.

2.4 discard() - Removes an element (does not raise an error if not found)

```
my_set = {1, 2, 3}
my_set.discard(2)
print(my_set)  # Output: {1, 3}
```

- **Safer alternative** to remove().

2.5 pop() - Removes and returns a random element

```
my_set = {1, 2, 3, 4}
removed_element = my_set.pop()
print(removed_element)  # Output: Random element
print(my_set)  # Output: Set after removal
```

- Removes an **arbitrary** element (since sets are unordered).

2.6 clear() - Removes all elements from the set

```
my_set = {1, 2, 3}
my_set.clear()
print(my_set)  # Output: set()
```

- Empties the set.

2.7 copy() - Returns a shallow copy of the set

```
original_set = {1, 2, 3}
copied_set = original_set.copy()
print(copied_set)  # Output: {1, 2, 3}
```

- Creates a **new set** with the same elements.

3. Set Operations (Mathematical & Logical Operations)

3.1 union() - Returns the union of two sets

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

- Combines elements from both sets **without duplicates**.

3.2 intersection() - Returns common elements of two sets

```
set1 = {1, 2, 3}
set2 = {2, 3, 4}
intersection_set = set1.intersection(set2)
print(intersection_set)  # Output: {2, 3}
```

- Returns only **common elements**.

3.3 difference() - Returns elements in set1 but not in set2

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

- Finds elements that exist in set1 but **not in set2**.

3.4 symmetric_difference() - Returns elements in either set but not both

set1 = {1, 2, 3}

set2 = {2, 3, 4}

sym_diff_set = set1.symmetric_difference(set2)

print(sym_diff_set) # Output: {1, 4}

- Returns **elements that are not common** in both sets.

3.5 issubset() - Checks if one set is a subset of another

set1 = {1, 2}

set2 = {1, 2, 3, 4}

print(set1.issubset(set2)) # Output: True

- Returns **True** if all elements of set1 exist in set2.

3.6 issuperset() - Checks if one set is a superset of another

set1 = {1, 2, 3, 4}

set2 = {1, 2}

print(set1.issuperset(set2)) # Output: True

- Returns **True** if set1 contains all elements of set2.

3.7 isdisjoint() - Checks if two sets have no common elements

set1 = {1, 2, 3}

set2 = {4, 5, 6}

print(set1.isdisjoint(set2)) # Output: True

- Returns **True** if there are **no common elements**.

---

4. Other Useful Operations

4.1 Checking Membership (in, not in)

```
my_set = {1, 2, 3}
print(2 in my_set)  # Output: True
print(4 not in my_set)  # Output: True
```

- Checks if an **element exists** in the set.

4.2 Looping Through a Set

```
my_set = {"Python", "Java", "C++"}
for item in my_set:
    print(item)
```

- Iterates through each element.

4.3 Finding Length (len())

```
my_set = {1, 2, 3, 4}
print(len(my_set))  # Output: 4
```

- Returns the **number of elements** in a set.

4.4 Converting List to Set (set())

```
my_list = [1, 2, 2, 3, 3, 4]
my_set = set(my_list)
print(my_set)  # Output: {1, 2, 3, 4}
```

- Removes duplicates from a list.

4.5 Converting Set to List (list())

```
my_set = {1, 2, 3}
my_list = list(my_set)
print(my_list)  # Output: [1, 2, 3]
```

- Allows **indexing and modification**.

Summary of Set Methods & Operations

Method/Operation	Description	Example
add(x)	Adds x to the set	my_set.add(4)
update(iterable)	Adds multiple elements	my_set.update([4, 5])
remove(x)	Removes x (error if missing)	my_set.remove(2)
discard(x)	Removes x (no error)	my_set.discard(2)
pop()	Removes a random element	my_set.pop()
clear()	Removes all elements	my_set.clear()
copy()	Creates a copy of the set	my_set.copy()
union()	Combines two sets	set1.union(set2)
intersection()	Common elements	set1.intersection(set2)
difference()	Elements in set1 but not in set2	set1.difference(set2)
symmetric_difference()	Elements in either set but not both	set1.symmetric_difference(set2)

Method/Operation	Description	Example
issubset()	Checks if a set is a subset	set1.issubset(set2)
issuperset()	Checks if a set is a superset	set1.issuperset(set2)
isdisjoint()	Checks if sets have no common elements	set1.isdisjoint(set2)

Note:

Sets provide **efficient operations for unique collections** and support **mathematical set operations** like union, intersection, and difference. 🚀