**✅ set – Mutable Unordered Collection of Unique Elements**

**Syntax:**

s = set([1, 2, 3])

or

s = {1, 2, 3}

**Common Operations:**

| **Operation** | **Syntax** | **Result** |
| --- | --- | --- |
| Add | s.add(4) | Adds 4 to set |
| Remove (error) | s.remove(2) | Removes 2, raises error if not present |
| Remove (safe) | s.discard(2) | Removes 2, no error if missing |
| Membership | 3 in s | True or False |
| Length | len(s) | Number of elements |
| Clear | s.clear() | Empties the set |

**Set Theory Ops:**

a = {1, 2, 3}

b = {3, 4, 5}

| **Operation** | **Syntax** | **Output** |
| --- | --- | --- |
| Union | `a | b` |
| Intersection | a & b | {3} |
| Difference | a - b | {1, 2} |
| Symmetric Difference | a ^ b | {1, 2, 4, 5} |
| **# Create sets**  **s1 = {1, 2, 3}**  **s2 = set([3, 4, 5])**  **# Add elements**  **s1.add(4) # {1, 2, 3, 4}**  **s1.update([5,6]) # {1, 2, 3, 4, 5, 6}**  **# Remove elements**  **s1.remove(6) # Raises KeyError if not found**  **s1.discard(7) # No error if not found**  **s1.pop() # Removes and returns arbitrary element**  **# Set operations**  **s1 | s2 # Union {1, 2, 3, 4, 5}**  **s1 & s2 # Intersection {3, 4}**  **s1 - s2 # Difference {1, 2}**  **s1 ^ s2 # Symmetric difference {1, 2, 5}**  **# Set comparisons**  **s1 <= s2 # Subset check**  **s1 >= s2 # Superset check** |  |  |

**✅ frozenset – Immutable Set**

Like set, but **immutable** (hashable, usable as a dict key or set element).

**Syntax:**

fs = frozenset([1, 2, 3])

**Differences from set:**

| **Feature** | **set** | **frozenset** |
| --- | --- | --- |
| Mutable | ✅ | ❌ |
| Hashable | ❌ | ✅ |
| Add/Remove Elements | ✅ | ❌ |
| Can be dict key | ❌ | ✅ |

**Frozenset Example:**

fs = frozenset({1, 2, 3})

print(fs.union({3, 4})) # frozenset({1, 2, 3, 4})

print(fs.intersection({2, 4})) # frozenset({2})

**# Create frozensets**

**fs1 = frozenset([1, 2, 3])**

**fs2 = frozenset([3, 4, 5])**

**# Frozenset operations (same as set but no modification)**

**fs1 | fs2 # Union**

**fs1 & fs2 # Intersection**

**fs1 - fs2 # Difference**

**fs1 ^ fs2 # Symmetric difference**

**Example1:**

**fs1 = frozenset([1, 2, 3])**

**fs2 = frozenset([3, 4, 5])**

**result = fs1 ^ fs2**

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

**Example 2:**

**teamA\_roles = frozenset(['dev', 'qa', 'admin'])**

**teamB\_roles = frozenset(['qa', 'pm', 'support'])**

**unique\_roles = teamA\_roles ^ teamB\_roles**

**print(unique\_roles) # frozenset({'dev', 'admin', 'pm', 'support'})**

**# Frozensets can be used as dictionary keys**

**d = {fs1: "value"}**

| **Operation** | **Symbol** | **Example** | **Output** |
| --- | --- | --- | --- |
| Union | ` | ` | ` fs1 |
| Intersection | & | fs1 & fs2 | Common elements |
| Difference | - | fs1 - fs2 | Only in fs1 |
| Symmetric Difference | ^ | fs1 ^ fs2 | In either, but not both |

| **Concept** | **Difference (-)** | **Symmetric Difference (^)** |
| --- | --- | --- |
| Meaning | Elements **only in the first set** | Elements in **either set, but not in both** |
| Removes common items? | Yes, **but only from the first set** | Yes, **from both sets** |
| Asymmetric? | ✅ Yes (A - B ≠ B - A) | ❌ No (A ^ B == B ^ A) |
| Result set direction | One-sided | Two-sided |

## 💡 Real-Life Analogy

* A - B: What I have that you **don’t**.
* A ^ B: What we **don’t have in common**.

## ⚠️ Common Interview Trap

**Q:** What is the difference between A - B and A ^ B?

**A:**

* A - B excludes only elements that are in **both** from the first set.
* A ^ B excludes **all common elements** from **both sets**.

### Key differences between sets and frozensets:

1. Sets are mutable, frozensets are immutable
2. Sets cannot be used as dictionary keys, frozensets can
3. Sets have add/remove methods, frozensets don't

**✅ Why frozenset Can Be Used in Dict or Set**

* set is **mutable** and **unhashable** → ❌ cannot be a dict key or added to a set.
* frozenset is **immutable** and **hashable** → ✅ can be used in both.

**🔹 Example 1: frozenset as Dictionary Key**

# Define frozenset as key

permissions = {

frozenset(['admin', 'read']): 'Full Access',

frozenset(['read']): 'Read Only'

}

# Access using equivalent frozenset

key = frozenset(['admin', 'read'])

print(permissions[key]) # Output: Full Access

**✅ Why it works:**

* frozenset(['admin', 'read']) is hashable and immutable.
* Can be used reliably as a key in a dictionary.

**🔹 Example 2: frozenset in a Set (Nested Set)**

# Trying to create a set of sets (not allowed)

# s = {set([1, 2]), set([3, 4])} # ❌ TypeError

# Using frozenset instead

nested\_sets = {frozenset([1, 2]), frozenset([3, 4])}

print(nested\_sets)

# Output: {frozenset({1, 2}), frozenset({3, 4})}

**🔸 Summary**

| **Feature** | **set** | **frozenset** |
| --- | --- | --- |
| Mutable | ✅ | ❌ |
| Hashable | ❌ | ✅ |
| Dict key allowed | ❌ | ✅ |
| Nested set allowed | ❌ | ✅ |

**📘 range –**

The range() function in Python is used to generate an **immutable sequence of numbers**, often used in loops and iterations.

**🔹 Syntax:**

range(start, stop, step)

* **start** *(optional)* – Beginning of the sequence (default = 0)
* **stop** *(required)* – End of the sequence (exclusive)
* **step** *(optional)* – Difference between each number (default = 1)

**Key Characteristics:**

* Lazy / memory-efficient (does **not** store all values at once)
* Supports indexing and slicing like a list
* Works well in for loops

**🔹 Examples:**

# Basic usage

list(range(5)) # [0, 1, 2, 3, 4]

# Custom start and stop

list(range(2, 7)) # [2, 3, 4, 5, 6]

# With step

list(range(1, 10, 2)) # [1, 3, 5, 7, 9]

# Reverse counting

list(range(10, 0, -2)) # [10, 8, 6, 4, 2]

| **Use Case** | | | **Code Example** | |
| --- | --- | --- | --- | --- |
| Pagination | | | range(page\_start, page\_end) | |
| Game Grid / Animation Frames | | | for i in range(frame\_count): | |
| Timed Sensor Reading | | | for i in range(0, duration, interval) | |
| Batch Processing | | | for i in range(0, len(data), batch\_size) | |
| **Feature** | **range()** | **list(range(...))** | |
| Memory usage | Low (lazy eval) | High (storesnumbers) | |
| Mutability | Immutable | Mutable | |
| Speed | Fast iteration | Slightly slower | |

**Common Use Case:**

for i in range(5):

print(i)

**Output:**

0

1

2

3

4

**Syntax:**

range(start, stop, step)

* start: beginning of sequence (default = 0)
* stop: **exclusive** end
* step: increment (default = 1)

**Examples:**

list(range(5)) # [0, 1, 2, 3, 4]

list(range(2, 10, 2)) # [2, 4, 6, 8]

**Operations:**

| **Operation** | **Supported?** | **Notes** |
| --- | --- | --- |
| Indexing | ✅ | range(5)[2] → 2 |
| Slicing | ✅ | range(10)[1:5] → range(1, 5) |
| Iteration | ✅ | Efficient, lazy |
| Contains | ✅ | 3 in range(10) |
| Length | ✅ | len(range(5)) → 5 |

**✅ When to Use:**

| **Use Case** | **Choose** |
| --- | --- |
| Iterate over a range efficiently | range |
| Unique mutable elements | set |
|  |  |

* **Range**: Iterating a specific number of times, creating sequences
* **Set**: Removing duplicates, membership testing, mathematical operations
* **Frozenset**: When you need an immutable set, dictionary keys with set characteristics

**🔹 range – Practical Examples**

**✅ Example 1: Sum of Even Numbers**

even\_sum = sum(range(0, 101, 2))

print(even\_sum) # Output: 2550

**✅ Example 2: Check if a Number Is in Range**

n = 50

if n in range(1, 100):

print("In range")

**🔹 set – Practical Examples**

**✅ Example 3: Remove Duplicates from List**

nums = [1, 2, 2, 3, 4, 4, 5]

unique\_nums = list(set(nums))

print(unique\_nums) # [1, 2, 3, 4, 5]

**✅ Example 4: Find Common Elements Between Two Lists**

a = [1, 2, 3, 4]

b = [3, 4, 5, 6]

common = set(a) & set(b)

print(common) # {3, 4}

**🔹 frozenset – Practical Examples**

**✅ Example 5: Use frozenset as Dictionary Key**

fs = frozenset([1, 2, 3])

d = {fs: "value"}

print(d[frozenset([1, 2, 3])]) # value

**✅ Example 6: Immutable Config**

allowed\_roles = frozenset(['admin', 'editor', 'viewer'])

# Cannot modify: allowed\_roles.add('guest') → ❌ error

**💼 Interview-Style Questions**

**🔸 range**

**Q1:** What will list(range(10, 0, -2)) return?

**A1:** [10, 8, 6, 4, 2]

**🔸 set**

**Q2:** How do you find elements present in list A but not in B?

A = [1, 2, 3, 4]

B = [3, 4, 5]

**Answer:**

difference = set(A) - set(B) # {1, 2}

**Q4:** How to check if two sets are disjoint?

**A4:**

set1.isdisjoint(set2)

**🔸 frozenset**

**Q5:** Why might you use a frozenset over a set?

**A5:**

* You want the set to be **immutable**
* You want to use it as a **key in a dictionary** or **element of another set**

**Q6:** What will happen?

fs = frozenset([1, 2, 3])

fs.add(4)

**A6:** ❌ AttributeError: 'frozenset' object has no attribute 'add'

**Q7: Frequency Counter Using Set**

def unique\_count(s):

return len(set(s))

print(unique\_count("hellooo")) # Output: 4

**✅ Summary Table**

| **Concept** | **Mutable** | **Hashable** | **Common Use** |
| --- | --- | --- | --- |
| range | ❌ | ✅ | Efficient looping |
| set | ✅ | ❌ | Unique collection, set operations |
| frozenset | ❌ | ✅ | Dict keys, immutability in sets |