📝 Assignment: Python Sets & Frozensets

📝 Assignment: Python Sets & Frozensets

Q1: Why are sets useful in Python (e.g., removing duplicates)?

✅ Answer:

- Sets automatically **remove duplicates** because they only store **unique elements**.
- Useful in data cleaning when you need to ensure no repeated values.
- Example:
    - A list [1,2,2,3,3] becomes {1,2,3} when converted to a set.
- Also useful for **mathematical operations** like union, intersection, and difference.

Q2: Explain the difference between .remove() and .discard().

✅ Answer:

- .remove(x) → Deletes element x. If x is **not found**, it raises an **error** (KeyError).
- .discard(x) → Deletes element x. If x is **not found**, it does **nothing** (no error).
- So, .discard() is **safer**, while .remove() is **stricter**.

Q3: What does immutability mean for a frozenset?

✅ Answer:

- **Immutability** means the object **cannot be changed** after creation.
- A frozenset is like a normal set, but you **cannot add, remove, or modify** elements inside it.
- This makes frozenset useful as a **dictionary key** or an element inside another set (because it's fixed).

Q4: Why might .pop() on a set return an unpredictable element?

✅ Answer:

- Sets in Python are **unordered collections**.
- Since they don't maintain sequence (like lists), .pop() removes and returns a **random element**.
- The element depends on Python's internal storage, so you **cannot predict** which one will be popped.

Assignment: Python Sets

Assignment: Python Sets & Frozensets

Q5: How does .clear() differ from creating a new empty set?

✅ **Answer:**

- .clear() → Empties the **existing set** in-place (the same object becomes empty).
- set() → Creates a **new empty set object**, while the old one still exists.
- Use .clear() if you want to **reuse** the same set; use set() if you want a **fresh new set**.

Assignment: Python Sets