Sets, Membership & Mathematical Set Operations

AiCE Software Engineering Foundations — Python Programming (Intermediate)

This lesson introduces sets — Python's fourth fundamental data structure, built specifically for uniqueness and speed. Sets remove duplicates automatically, test membership dramatically faster than lists, and support genuine mathematical operations like union and intersection directly.

1. What Is a Set?

A set is an unordered collection of unique items. Two properties define sets completely:

Unordered — items have no position. There is no first item, no last item, no index. You cannot access set items by position.

Unique — a set never contains duplicate values. If you add an item that already exists, nothing happens. The set stays the same.

Sets are written with curly braces, items separated by commas:

```
african_nations = {"Nigeria", "Ghana", "Kenya", "Senegal",
"Rwanda"}
prime_numbers = {2, 3, 5, 7, 11, 13}
tech_skills = {"Python", "SQL", "Git", "Linux"}
```

The primary use cases for sets are: removing duplicates from a collection, membership testing (extremely fast), and mathematical set operations — union, intersection, difference, symmetric difference.

2. Creating Sets

```
# Creating sets
cities = {"Lagos", "Nairobi", "Accra", "Kigali"}
print(cities)
```

```
print(type(cities))

# Sets automatically remove duplicates
with_dupes = {"Nigeria", "Ghana", "Nigeria", "Kenya", "Ghana",
"Nigeria"}
print(with_dupes)

# Creating a set from a list (deduplication)
enrolment_list = ["Amara", "Kofi", "Amara", "Ngozi", "Kofi",
"Fatima", "Amara"]
unique_students = set(enrolment_list)
print(f"List length : {len(enrolment_list)}")
print(f"Unique count: {len(unique_students)}")
print(f"Unique names: {unique_students}")

# Empty set — must use set(), NOT {}
empty_set = set()     # Correct
empty_dict = {}       # This is an empty DICTIONARY, not a
set!
print(type(empty_set))
print(type(empty_dict))
```

Notice the output order may differ from input order — sets are unordered, so Python makes no guarantee about which order items appear when printed. Every run might show a different order.

> **Empty Set vs. Empty Dictionary**
> *{} creates an empty dictionary, not an empty set. To create an empty set, always use set(). This is a common mistake that produces confusing bugs — the empty dict and empty set look similar but behave completely differently.*

3. Set Operations — Adding and Removing Items

Unlike tuples, sets are mutable — you can add and remove items. But you cannot update an existing item by index, since there are no indexes:

```
skills = {"Python", "SQL", "Git"}
```

```
# Add a single item
skills.add("Linux")
print(f"After add   : {skills}")

# Add multiple items using update()
skills.update(["JavaScript", "Docker"])
print(f"After update: {skills}")

# Remove an item — raises KeyError if not found
skills.remove("Docker")
print(f"After remove: {skills}")

# Discard — does NOT raise error if item missing
skills.discard("Rust")    # Rust was not in the set — no error
print(f"After discard: {skills}")

# Pop — removes and returns an arbitrary item
removed = skills.pop()
print(f"Popped: {removed}")
print(f"Remaining: {skills}")

# Clear — removes all items
skills.clear()
print(f"After clear: {skills}")
```

.remove() raises an error if the item doesn't exist; .discard() does not — choose based on whether a missing item should be treated as an error in your program.

4. Set Membership Testing

Membership testing is where sets truly shine. The in keyword works with lists, tuples, dictionaries, and sets — but with sets, it is dramatically faster. For a list, Python checks each item one by one until it finds a match or reaches the end. For a set, Python uses a mathematical technique called hashing that finds the answer in constant time — the same speed whether the set has 10 items or 10 million.

```
# Set membership testing
registered_ids = {"AiCE001", "AiCE002", "AiCE003", "AiCE004",
"AiCE005"}

# Check membership
print("AiCE003" in registered_ids)        # True
print("AiCE999" in registered_ids)        # False
print("AiCE999" not in registered_ids)    # True

# Practical: verify student ID before granting access
def check_access(student_id, valid_ids):
    if student_id in valid_ids:
        return f"Access granted: {student_id}"
    return f"Access denied: {student_id} not registered"

print(check_access("AiCE002", registered_ids))
print(check_access("AiCE010", registered_ids))

# Deduplication use case
submitted_emails = [
    "amara@aice.africa", "kofi@aice.africa",
"amara@aice.africa",
    "ngozi@aice.africa", "kofi@aice.africa",
]
unique_emails = set(submitted_emails)
duplicates = len(submitted_emails) - len(unique_emails)

print(f"Total submitted : {len(submitted_emails)}")
print(f"Unique emails   : {len(unique_emails)}")
print(f"Duplicates found: {duplicates}")
print(f"Unique list     : {sorted(unique_emails)}")
```

The sorted() call on the set produces an alphabetically ordered list — useful whenever you need set contents in a predictable order for display.

5. Mathematical Set Operations

Sets support four fundamental mathematical operations built directly into Python — widely used in data analysis, access control, and comparison logic:

```
# Set operations demonstrated with AiCE course data
python_students = {"Amara", "Kofi", "Ngozi", "Fatima",
"Kwame"}
sql_students = {"Kofi", "Fatima", "Zainab", "Emeka", "Diallo"}

# 1. UNION — students enrolled in EITHER course (or both)
all_students = python_students | sql_students
# Also: python_students.union(sql_students)
print(f"Union (|)        : {sorted(all_students)}")

# 2. INTERSECTION — students enrolled in BOTH courses
both_courses = python_students & sql_students
# Also: python_students.intersection(sql_students)
print(f"Intersection (&) : {sorted(both_courses)}")

# 3. DIFFERENCE — in Python but NOT in SQL
python_only = python_students - sql_students
# Also: python_students.difference(sql_students)
print(f"Difference (-)   : {sorted(python_only)}")

# 4. SYMMETRIC DIFFERENCE — in ONE course but NOT both
exclusive = python_students ^ sql_students
# Also: python_students.symmetric_difference(sql_students)
print(f"Sym. Diff (^)    : {sorted(exclusive)}")

# Subset and superset checks
core_students = {"Kofi", "Fatima"}
print(f"core is subset of python  :
{core_students.issubset(python_students)}")
print(f"python is superset of core:
{python_students.issuperset(core_students)}")

# Disjoint — do the sets share any items?
new_batch = {"Aisha", "Suleiman", "Dayo"}
print(f"python & new_batch disjoint:
{python_students.isdisjoint(new_batch)}")
```

Operation	Symbol	Method	What It Returns
Union	A | B	A.union(B)	All items from both sets (no duplicates)
Intersection	A & B	A.intersection(B)	Only items in both A and B
Difference	A - B	A.difference(B)	Items in A but not in B
Symmetric Difference	A ^ B	A.symmetric_difference(B)	Items in A or B but not both
Subset	A <= B	A.issubset(B)	True if all of A is in B
Superset	A >= B	A.issuperset(B)	True if B is entirely within A
Disjoint	N/A	A.isdisjoint(B)	True if A and B share no items

Choose the Structure That Fits

"The right data structure solves the problem before you write a single line of logic. A list when order matters. A dictionary when names matter. A tuple when data must not change. A set when uniqueness matters. Choose the structure that fits the problem and the code writes itself." — Adesina Olagunju, Founder, AiCE

Key Takeaways

A set is an unordered collection of unique items — adding a duplicate silently does nothing, and membership testing with in is dramatically faster than on a list, regardless of size. .add() and .update() insert items, .remove() and .discard() take them out (only .remove() raises an error if the item is missing), and the four mathematical operations — union, intersection, difference, and symmetric difference — answer real questions about overlapping collections directly. The next lesson closes this unit — and the entire Intermediate Python section — with a full comparison of lists, tuples, sets, and dictionaries, and a capstone that uses tuples and sets together in a complete enrollment analysis system.