Python Set Operations

In the previous topic, you learned what sets are and why they are useful for storing **unique values**.

The real power of sets appears when you have **two or more groups of data** and need to compare them.

For example:

```
python_students = {"Ali", "Hafsa", "Sara"}
javascript_students = {"Hafsa", "Sara", "Usman"}
```

You may want to know:

- Who is in either group?
- Who is in both groups?
- Who is only learning Python?
- Who is only learning JavaScript?
- Who is in exactly one group?
- Is one group completely contained inside another?

Python provides built in **set operations** for these problems.

The five major operations are:

```
Union
Intersection
Difference
Symmetric Difference
Subset / Superset
```

---

1. Why Set Operations Matter

Suppose a website has two groups of users:

```
python_users = {"Ali", "Hafsa", "Sara"}
javascript_users = {"Hafsa", "Sara", "Usman"}
```

Instead of writing complicated loops and conditions, set operations let you express the problem directly.

For example:

```
python_users & javascript_users
```

means:

> Give me users who are in both groups.

This makes set operations especially useful for:

- user data
- permissions
- tags
- categories
- courses
- skills
- database results
- filtering
- analytics
- recommendation systems

---

2. The Four Core Set Operations

Python provides operators for the main set operations:

Operation	Operator	Method
Union	`|`	`union()`
Intersection	`&`	`intersection()`
Difference	`-`	`difference()`
Symmetric Difference	`^`	`symmetric_difference()`

Example:

```
A = {1, 2, 3}
B = {3, 4, 5}
```

Then:

```
A | B
```

```
A & B
```

```
A - B
```

```
A ^ B
```

Each answers a different question.

---

3. Union

What Is Union?

Union combines all unique values from two sets.

Suppose:

```
A = {1, 2, 3}
B = {3, 4, 5}
```

Use:

```
result = A | B

print(result)
```

Result:

```
{1, 2, 3, 4, 5}
```

Notice that `3` appears in both sets, but it appears only once in the result.

That is because sets contain unique values.

---

4. Union Using `union()`

The same operation can be written using the method:

```
result = A.union(B)

print(result)
```

Result:

```
{1, 2, 3, 4, 5}
```

So these are equivalent:

```
A | B
```

and:

```
A.union(B)
```

---

5. Union Mental Model

Think:

```
A = {1, 2, 3}
B = {3, 4, 5}

A | B
```

You want:

```
Everything from A
+
Everything from B
-
Duplicates
```

Result:

```
{1, 2, 3, 4, 5}
```

Question Union Answers

> **What values exist in either set?**

---

6. Real World Example: Programming Languages

```
frontend = {"HTML", "CSS", "JavaScript"}
backend = {"Python", "JavaScript", "SQL"}
```

Find all technologies:

```
technologies = frontend | backend

print(technologies)
```

Result:

```
{"HTML", "CSS", "JavaScript", "Python", "SQL"}
```

`JavaScript` was present in both sets but appears only once.

---

7. Intersection

What Is Intersection?

Intersection finds values that exist in **both sets**.

Example:

```
A = {1, 2, 3}
B = {3, 4, 5}

result = A & B

print(result)
```

Result:

```
{3}
```

Only `3` exists in both sets.

---

8. Intersection Using `intersection()`

You can also write:

```
result = A.intersection(B)

print(result)
```

This produces:

```
{3}
```

So:

```
A & B
```

and:

```
A.intersection(B)
```

mean the same thing.

---

9. Intersection Mental Model

Think:

```
A = {1, 2, 3}
B = {3, 4, 5}
```

Ask:

> Which values are shared by both groups?

Answer:

```
{3}
```

Question Intersection Answers

> **What values do these sets have in common?**

---

10. Real World Example: Common Skills

```
hafsa_skills = {
    "Python",
    "JavaScript",
    "Git",
    "SQL"
}

asim_skills = {
    "Python",
    "Git",
    "Docker",
    "Linux"
}
```

Find their common skills:

```
common_skills = hafsa_skills & asim_skills

print(common_skills)
```

Result:

```
{"Python", "Git"}
```

This is much easier than manually checking every skill.

---

11. Difference

What Is Difference?

Difference finds values that exist in the **first set but not the second set**.

Example:

```
A = {1, 2, 3}
B = {3, 4, 5}

result = A - B

print(result)
```

Result:

```
{1, 2}
```

`1` and `2` exist in `A` but not in `B`.

---

12. Difference Using `difference()`

You can write:

```
result = A.difference(B)

print(result)
```

Result:

```
{1, 2}
```

So:

```
A - B
```

and:

```
A.difference(B)
```

are equivalent.

---

13. Difference Is Directional

This is very important.

These two operations are not necessarily the same:

```
A - B
```

and:

```
B - A
```

Example:

```
A = {1, 2, 3}
B = {3, 4, 5}

print(A - B)
print(B - A)
```

Output:

```
{1, 2}
{4, 5}
```

Why?

`A - B` means:

> Values in A that are not in B.

`B - A` means:

> Values in B that are not in A.

---

14. Real World Example: Python Only Students

```
python_students = {
    "Ali",
    "Hafsa",
    "Sara",
    "Usman"
}

javascript_students = {
    "Hafsa",
    "Sara",
    "Ahmed"
}
```

Find students who are learning Python but not JavaScript:

```
python_only = python_students - javascript_students

print(python_only)
```

Result:

```
{"Ali", "Usman"}
```

---

15. Reverse Difference

Find JavaScript students who are not learning Python:

```
javascript_only = javascript_students - python_students

print(javascript_only)
```

Result:

```
{"Ahmed"}
```

The direction changes the result.

---

16. Symmetric Difference

What Is Symmetric Difference?

Symmetric difference finds values that belong to **one set but not both**.

Example:

```
A = {1, 2, 3}
B = {3, 4, 5}

result = A ^ B

print(result)
```

Result:

```
{1, 2, 4, 5}
```

The common value `3` is removed.

---

17. Symmetric Difference Using a Method

You can also write:

```
result = A.symmetric_difference(B)

print(result)
```

Result:

```
{1, 2, 4, 5}
```

So:

```
A ^ B
```

and:

```
A.symmetric_difference(B)
```

are equivalent.

---

18. Symmetric Difference Mental Model

Given:

```
A = {1, 2, 3}
B = {3, 4, 5}
```

Remove the values shared by both.

What remains?

```
{1, 2, 4, 5}
```

Question Symmetric Difference Answers

> **What values belong to exactly one of these sets?**

---

19. Comparing the Four Operations

Given:

```
A = {1, 2, 3}
B = {3, 4, 5}
```

Union

```
A | B
```

Result:

```
{1, 2, 3, 4, 5}
```

Meaning:

> Everything.

Intersection

```
A & B
```

Result:

```
{3}
```

Meaning:

> Common values.

Difference

```
A - B
```

Result:

```
{1, 2}
```

Meaning:

> Only A.

Symmetric Difference

```
A ^ B
```

Result:

```
{1, 2, 4, 5}
```

Meaning:

> Only one side, not both.

---

20. Easy Memory Trick

Remember:

```
|  → OR → everything from both
&  → AND → common values
-  → remove → only from the left
^  → exclusive OR → only one side
```

This makes the operators easier to remember.

---

21. Set Operations With Three Sets

Set operations are not limited to two sets.

Suppose:

```
python = {"Ali", "Hafsa", "Sara"}
javascript = {"Hafsa", "Sara", "Usman"}
react = {"Hafsa", "Usman", "Ahmed"}
```

You can combine sets:

```
all_students = python | javascript | react
```

You can find students common to all three:

```
common_students = python & javascript & react
```

The result is:

```
{"Hafsa"}
```

because Hafsa exists in all three groups.

---

22. Union of Multiple Sets

```
frontend = {"HTML", "CSS", "JavaScript"}
backend = {"Python", "Django", "SQL"}
mobile = {"Flutter", "Dart", "JavaScript"}

all_technologies = frontend | backend | mobile

print(all_technologies)
```

The result contains every unique technology across the three groups.

---

23. Intersection of Multiple Sets

```
group_a = {"Python", "Git", "SQL", "Docker"}
group_b = {"Python", "Git", "Docker"}
group_c = {"Python", "Git"}

common = group_a & group_b & group_c

print(common)
```

Result:

```
{"Python", "Git"}
```

These are the values shared by all three groups.

---

24. Set Difference With Multiple Sets

You can also subtract multiple sets.

```
all_skills = {
    "Python",
    "JavaScript",
    "Git",
    "SQL",
    "Docker"
}

known_skills = {
    "Python",
    "Git"
}

not_known = all_skills - known_skills

print(not_known)
```

Result:

```
{"JavaScript", "SQL", "Docker"}
```

---

25. Subsets

Set operations are not only about combining sets.

You can also determine whether one set is completely contained inside another.

Example:

```
all_languages = {
    "Python",
    "JavaScript",
    "Java",
    "C++"
}

backend_languages = {
```

```
    "Python",
    "Java"
}
```

Check:

```
print(backend_languages.issubset(all_languages))
```

Output:

```
True
```

This means:

> Every item in `backend_languages` exists in `all_languages`.

---

26. Subset Operator

You can also use:

```
backend_languages <= all_languages
```

Output:

```
True
```

So these are equivalent:

```
A.issubset(B)
```

and:

```
A <= B
```

---

27. Proper Subset

You can use `<` when you want to check whether one set is a **proper subset**.

```
A = {1, 2}
B = {1, 2, 3}

print(A < B)
```

Output:

```
True
```

`A` is smaller than `B` and every item in `A` is inside `B`.

But:

```
A = {1, 2}
B = {1, 2}

print(A < B)
```

Output:

```
False
```

They contain the same values, so `A` is not a proper subset.

---

28. Superset

A superset contains every item from another set.

```
all_languages = {
    "Python",
    "JavaScript",
    "Java"
}

backend = {
    "Python",
    "Java"
}

print(all_languages.issuperset(backend))
```

Output:

```
True
```

You can also use:

```
print(all_languages >= backend)
```

---

29. Proper Superset

Use `>` to check for a proper superset.

```
A = {1, 2, 3}
B = {1, 2}
```

```
print(A > B)
```

Output:

```
True
```

`A` contains everything in `B` and has additional values.

---

30. Disjoint Sets

Two sets are **disjoint** when they have no common values.

Example:

```
frontend = {"HTML", "CSS", "JavaScript"}
databases = {"MySQL", "PostgreSQL", "MongoDB"}

print(frontend.isdisjoint(databases))
```

Output:

```
True
```

There are no common values.

---

31. Non Disjoint Sets

If two sets have at least one common value, they are not disjoint.

```
frontend = {"HTML", "CSS", "JavaScript"}
web = {"JavaScript", "Python"}

print(frontend.isdisjoint(web))
```

Output:

```
False
```

Because `"JavaScript"` exists in both.

---

32. Updating Sets With Operations

There is an important difference between:

```
A | B
```

and:

```
A |= B
```

The first creates a new result:

```
result = A | B
```

The second updates `A` itself:

```
A |= B
```

Example:

```
A = {1, 2}
B = {2, 3}

A |= B

print(A)
```

Result:

```
{1, 2, 3}
```

Similarly:

```
A &= B
```

updates `A` with the intersection.

```
A -= B
```

updates `A` with the difference.

```
A ^= B
```

updates `A` with the symmetric difference.

---

33. Non Mutating vs Updating Operations

Compare:

```
A = {1, 2}
B = {2, 3}

result = A | B

print(A)
print(result)
```

`A` remains:

```
{1, 2}
```

while `result` becomes:

```
{1, 2, 3}
```

But:

```
A |= B
```

changes `A`.

Now:

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

This distinction matters when writing larger programs.

---

34. Method Versions With Multiple Sets

Some methods can work with multiple sets.

For example:

```
A.union(B, C)
```

combines all three.

Similarly:

```
A.intersection(B, C)
```

finds values common to all.

You can also use operators:

```
A | B | C
```

and:

```
A & B & C
```

Both styles are useful.

---

35. Real World Example: Course Enrollment

Imagine a learning platform has three courses:

```
python_course = {
    "Ali",
    "Hafsa",
    "Sara",
    "Usman"
}

javascript_course = {
    "Hafsa",
    "Sara",
    "Ahmed"
}

react_course = {
    "Hafsa",
    "Ahmed",
    "Bilal"
}
```

Students enrolled in at least one course

```
all_students = (
    python_course
    | javascript_course
    | react_course
)
```

Students enrolled in both Python and JavaScript

```
python_and_js = python_course & javascript_course
```

Result:

```
{"Hafsa", "Sara"}
```

Students enrolled in Python but not JavaScript

```
python_only = python_course - javascript_course
```

Result:

```
{"Ali", "Usman"}
```

Students enrolled in exactly one of Python and JavaScript

```
exactly_one = python_course ^ javascript_course
```

Result:

```
{"Ali", "Usman", "Ahmed"}
```

---

36. Real World Example: Developer Skills

```
frontend_skills = {
    "HTML",
    "CSS",
    "JavaScript",
    "React"
}

backend_skills = {
    "Python",
    "Django",
    "SQL",
    "JavaScript"
}
```

All skills

```
all_skills = frontend_skills | backend_skills
```

Common skills

```
common_skills = frontend_skills & backend_skills
```

Result:

```
{"JavaScript"}
```

Frontend only

```
frontend_only = frontend_skills - backend_skills
```

Backend only

```
backend_only = backend_skills - frontend_skills
```

This type of comparison is useful in skill analysis and recommendation systems.

---

37. Real World Example: haas.dev Resources

Imagine haas.dev has resources tagged by technology:

```
python_resources = {
    "Python Basics",
    "Python Loops",
    "Python Lists"
}

web_resources = {
    "HTML Basics",
    "Python Basics",
    "JavaScript Basics"
}
```

Find resources that belong to both categories:

```
shared = python_resources & web_resources

print(shared)
```

Result:

```
{"Python Basics"}
```

Find resources unique to Python:

```
python_only = python_resources - web_resources
```

Find all resources:

```
all_resources = python_resources | web_resources
```

Set operations allow the application to perform these comparisons without manually checking every resource.

---

38. Real World Example: User Permissions

Suppose:

```
user_permissions = {
    "read",
    "write",
    "comment"
```

```
}

required_permissions = {
    "read",
    "write"
}
```

Check whether the user has everything required:

```
if required_permissions.issubset(user_permissions):
    print("Access granted")
else:
    print("Access denied")
```

You can also find exactly what is missing:

```
missing = required_permissions - user_permissions

print(missing)
```

If the user has only:

```
user_permissions = {"read"}
```

then:

```
missing = {"write"}
```

This is a practical application of set difference.

---

39. Real World Example: Duplicate Data

Suppose:

```
usernames = [
    "hafsa",
    "asim",
    "hafsa",
    "sara",
    "asim"
]
```

Convert to a set:

```
unique_usernames = set(usernames)
```

Now:

```
{"hafsa", "asim", "sara"}
```

To check whether duplicates existed:

```
if len(usernames) != len(set(usernames)):
    print("Duplicate usernames found")
```

This is a useful validation pattern.

---

40. Set Operations vs Loops

Without set operations, you might manually write:

```
common = []

for item in A:
    if item in B:
        common.append(item)
```

With sets:

```
common = A & B
```

The second version expresses the intention much more clearly.

The goal is not to avoid loops completely.

The goal is to use the data structure that naturally matches the problem.

---

41. Common Mistakes

Mistake 1: Forgetting Difference Direction

These are different:

```
A - B
```

and:

```
B - A
```

Always ask:

> Which set do I want values **from**?

Mistake 2: Confusing Union and Intersection

Union:

```
A | B
```

means:

> Everything from both.

Intersection:

```
A & B
```

means:

> Only what they share.

Mistake 3: Confusing Difference and Symmetric Difference

Difference:

```
A - B
```

means:

> Only A.

Symmetric difference:

```
A ^ B
```

means:

> Only one side, excluding common values.

Mistake 4: Expecting Operations to Always Modify the Original Set

This:

```
A | B
```

does not modify `A`.

You need:

```
A |= B
```

if you want to update `A`.

Mistake 5: Treating Sets Like Lists

This does not work:

```
A[0]
```

Set operations are based on **membership and relationships**, not positions.

---

42. Problem Solving Framework

When you have two groups of data, ask these questions:

1. Do I need everything?

Use:

```
A | B
```

2. Do I need common values?

Use:

```
A & B
```

3. Do I need values only in A?

Use:

```
A - B
```

4. Do I need values only in B?

Use:

```
B - A
```

5. Do I need values that belong to exactly one group?

Use:

```
A ^ B
```

6. Do I need to know whether A is completely contained in B?

Use:

```
A.issubset(B)
```

7. Do I need to know whether A contains B?

Use:

```
A.issuperset(B)
```

8. Do I need to know whether they have nothing in common?

Use:

```
A.isdisjoint(B)
```

---

43. Mini Project: Student Course Analyzer

Build a program that analyzes course enrollment.

```
python_students = {
    "Ali",
    "Hafsa",
    "Sara",
    "Usman"
}

javascript_students = {
    "Hafsa",
    "Sara",
    "Ahmed"
}

react_students = {
    "Hafsa",
    "Ahmed",
    "Bilal"
}
```

Step 1: Find all students

```
all_students = (
    python_students
    | javascript_students
    | react_students
)

print(all_students)
```

Step 2: Find Python and JavaScript students

```
python_and_js = python_students & javascript_students

print(python_and_js)
```

Step 3: Find Python only

```
python_only = python_students - javascript_students

print(python_only)
```

Step 4: Find students in Python or JavaScript, but not both

```
one_of_two = python_students ^ javascript_students

print(one_of_two)
```

Step 5: Check whether every Python student is also enrolled in React

```
if python_students.issubset(react_students):
    print("Everyone is enrolled in React")
else:
    print("Some students are not enrolled in React")
```

This project combines the major set operations into one realistic problem.

---

44. 5 Minute Challenge

Create these sets:

```
python = {
    "Ali",
    "Hafsa",
    "Sara",
    "Usman"
}

javascript = {
    "Hafsa",
    "Sara",
    "Ahmed"
}

react = {
    "Hafsa",
    "Ahmed",
    "Bilal"
}
```

Your program should calculate:

1. All unique students.
2. Students learning both Python and JavaScript.
3. Students learning Python but not JavaScript.
4. Students learning JavaScript but not Python.
5. Students learning exactly one of Python and JavaScript.
6. Students learning all three technologies.
7. Students learning Python or React.
8. Whether all Python students are also React students.
9. Whether Python and React have no students in common.
10. The students who are in React but not Python or JavaScript.

Try solving each problem using set operations rather than manual loops.

---

45. Practice Exercises

Exercise 1

Given:

```
A = {1, 2, 3}
B = {3, 4, 5}
```

Find:

- union
- intersection
- `A - B`
- `B - A`
- symmetric difference

Exercise 2

Create two sets of programming languages and find all languages known by either group.

Exercise 3

Create two sets of programming languages and find languages known by both groups.

Exercise 4

Create two sets of students and find students who belong only to the first group.

Exercise 5

Create two sets of students and find students who belong to exactly one group.

Exercise 6

Create a set of required skills and a set of user skills.

Check whether the user has all required skills.

Exercise 7

Find the missing skills using:

```
required - user
```

Exercise 8

Create three sets of course students.

Find students enrolled in all three courses.

Exercise 9

Create two sets of website features and find the features shared by both projects.

Exercise 10

Create a list containing duplicate categories.

Convert it to a set and determine whether duplicates existed.

---

46. Mini Quiz

Question 1

Given:

```
A = {1, 2, 3}
B = {3, 4, 5}
```

What does:

```
A | B
```

return?

A.

```
{3}
```

B.

```
{1, 2}
```

C.

```
{1, 2, 3, 4, 5}
```

D.

```
{4, 5}
```

Question 2

What does `&` represent in set operations?

A. Union

B. Intersection

C. Difference

D. Symmetric difference

Question 3

What does:

```
A - B
```

mean?

A. Values common to A and B

B. All values from A and B

C. Values in A but not B

D. Values in exactly one of A and B

Question 4

Which operation finds values belonging to exactly one of two sets?

A. `|`

B. `&`

C. `-`

D. `^`

Question 5

What does:

```
A.issubset(B)
```

check?

Answers

1. C
2. B
3. C
4. D
5. It checks whether every value in `A` also exists in `B`.

---

47. Interview Questions

1. What are set operations in Python?
2. What is union?
3. What is intersection?
4. What is the difference between union and intersection?
5. What does the `|` operator do?
6. What does the `&` operator do?
7. What does the `-` operator do?
8. Why is set difference directional?
9. What does the `^` operator do?
10. What is symmetric difference?
11. What is the difference between difference and symmetric difference?
12. How do you find common values between two sets?
13. How do you find values unique to one set?
14. How do you find all unique values across two sets?
15. How do you find values shared by three sets?
16. What is a subset?
17. What is a superset?
18. What does `issubset()` do?
19. What does `issuperset()` do?
20. What does `isdisjoint()` check?
21. What is the difference between `A | B` and `A |= B`?
22. Do set operations always modify the original set?
23. How can set operations help remove duplicate data?
24. How can sets be used to compare user permissions?

25. How can sets be used to find common skills between developers?

---

48. Set Operations Cheat Sheet

```
A | B
```

Union

Everything from both sets.

```
A.union(B)
```

```
A & B
```

Intersection

Values common to both sets.

```
A.intersection(B)
```

```
A - B
```

Difference

Values in A but not B.

```
A.difference(B)
```

```
A ^ B
```

Symmetric Difference

Values in exactly one of the two sets.

```
A.symmetric_difference(B)
```

```
A.issubset(B)
```

Checks whether A is contained inside B.

```
A.issuperset(B)
```

Checks whether A contains B.

```
A.isdisjoint(B)
```

Checks whether A and B have no common values.

Operator Cheat Sheet

```
A | B  → everything
A & B  → common values
A - B  → only A
B - A  → only B
A ^ B  → only one side
A <= B → A is a subset of B
A < B  → A is a proper subset of B
A >= B → A is a superset of B
A > B  → A is a proper superset of B
```

---

49. Key Takeaways

- Set operations allow you to compare and combine groups of unique values.
- `|` performs union.
- `&` performs intersection.
- `-` performs difference.
- `^` performs symmetric difference.
- Difference depends on the direction of the operation.
- Union gives everything from both sets.
- Intersection gives only common values.
- Difference gives values from the left set that are absent from the right.
- Symmetric difference gives values belonging to exactly one set.
- `issubset()` checks whether one set is contained in another.
- `issuperset()` checks whether one set contains another.
- `isdisjoint()` checks whether two sets have no values in common.
- Operators such as `|`, `&`, `-`, and `^` provide concise ways to express set relationships.
- Set operations can replace complicated manual comparison logic in many real-world problems.
- They are especially useful for users, skills, permissions, tags, categories, courses, and data analysis.

Final Mental Model

When you have two sets, think in terms of questions:

```
A = {1, 2, 3}
B = {3, 4, 5}
```

Need everything?

```
A | B
```

→ `{1, 2, 3, 4, 5}`

Need what they share?

```
A & B
```

→ `{3}`

Need only A?

```
A - B
```

→ `{1, 2}`

Need only B?

```
B - A
```

→ `{4, 5}`

Need values from exactly one side?

```
A ^ B
```

→ `{1, 2, 4, 5}`

The core idea is:

> **Set operations turn questions about groups of data into simple Python expressions.**

Visit haas.dev for more resources and guides.

https://dev-roast-app.vercel.app/resources