Python `if` Statements

Teach Your Programs to Make Decisions

Learning Objectives

By the end of this guide, you will understand:

- What an `if` statement is
- Why programs need decision making
- How Python evaluates conditions
- The syntax of an `if` statement
- Indentation and code blocks
- Boolean conditions
- Comparison and logical operators inside `if`
- Nested conditions
- Multiple independent `if` statements
- Truthy and falsy values
- `if` with user input
- `if` with strings, numbers, and collections
- Common mistakes
- Real world decision making patterns
- How to design readable conditions
- How to test decision logic
- How `if` statements fit into larger programs

---

1. Why Do Programs Need Decisions?

A program rarely performs exactly the same action every time.

Consider a shopping application:

```
If the product is in stock → allow purchase
If the product is out of stock → show unavailable
```

A learning platform:

```
If the learner completed the lesson → unlock the next lesson
```

A banking application:

```
If balance is sufficient → allow withdrawal
```

A login system:

```
If credentials are correct → allow access
```

A Python program needs a way to express these rules.

That is what the `if` statement provides.

---

2. What Is an `if` Statement?

An `if` statement tells Python:

> Run this block of code only when a particular condition is true.

Example:

```
age = 25

if age >= 18:
    print("Access allowed")
```

Python evaluates:

```
age >= 18
```

The result is:

```
True
```

Therefore Python executes:

```
print("Access allowed")
```

---

3. The Basic Structure

The basic syntax is:

```
if condition:
    # code to run
```

For example:

```
score = 80

if score >= 50:
    print("Pass")
```

There are three important parts:

```
if
 ↓
condition
 ↓
:
 ↓
indented code block
```

---

4. Understanding the Colon `:`

Python uses a colon after the condition:

```
if age >= 18:
```

The colon tells Python:

> The block belonging to this `if` statement starts here.

Then the code inside the block must be indented.

```
if age >= 18:
    print("Adult")
```

Without the colon:

```
if age >= 18
```

Python will report a syntax error.

---

5. Indentation Is Part of Python Syntax

Python uses indentation to define code blocks.

Correct:

```
if age >= 18:
    print("Adult")
    print("Access allowed")
```

Both `print()` statements belong to the `if` block.

Incorrect:

```
if age >= 18:
print("Adult")
```

Python expects an indented block after the `if`.

---

6. How an `if` Statement Executes

Consider:

```
age = 25

if age >= 18:
    print("Adult")

print("Program finished")
```

Python processes it like this:

```
age = 25
   ↓
Evaluate age >= 18
   ↓
True
   ↓
Run print("Adult")
   ↓
Run print("Program finished")
```

Output:

```
Adult
Program finished
```

---

7. What Happens When the Condition Is False?

Consider:

```
age = 15

if age >= 18:
    print("Adult")

print("Program finished")
```

The condition:

```
age >= 18
```

becomes:

```
False
```

Therefore Python skips the indented block.

Output:

```
Program finished
```

The `if` block is not executed.

---

8. The Core Mental Model

Think of an `if` statement as a gate:

```
              Condition
                  ↓
            ┌──────────────┐
            │ Is it True? │
            └──────────────┘
               ↓       ↓
              Yes      No
               ↓        ↓
         Run block   Skip block
```

That is the fundamental behavior of `if`.

---

9. `if` With Boolean Variables

You can directly use a Boolean variable as a condition.

```
is_logged_in = True

if is_logged_in:
    print("Welcome")
```

Because:

```
is_logged_in → True
```

Python executes the block.

---

10. `if` With `False`

```
is_logged_in = False
```

```
if is_logged_in:
    print("Welcome")
```

Nothing is printed because the condition is false.

This pattern is useful when a variable already represents a yes/no state.

---

11. `if` With Comparison Operators

Comparison operators are commonly used inside `if`.

```
score = 75

if score >= 50:
    print("Pass")
```

Other examples:

```
if age == 18:
    print("Exactly 18")

if age != 18:
    print("Not 18")

if age > 18:
    print("Older than 18")

if age < 18:
    print("Younger than 18")
```

---

12. `if` With Logical Operators

You can combine multiple conditions.

```
age = 25
has_account = True

if age >= 18 and has_account:
    print("Access granted")
```

Both conditions must be true.

Another example:

```
is_admin = False
is_manager = True
```

```
if is_admin or is_manager:
    print("Staff access")
```

---

13. `if` With `not`

```
account_blocked = False

if not account_blocked:
    print("Account is active")
```

The expression:

```
not account_blocked
```

becomes:

```
True
```

because `account_blocked` is `False`.

---

14. `if` With Strings

You can compare strings inside conditions.

```
language = "Python"

if language == "Python":
    print("Python selected")
```

The condition checks whether the variable contains exactly:

```
Python
```

---

15. Case Sensitivity

Python string comparisons are case-sensitive.

```
language = "python"

if language == "Python":
    print("Selected")
```

The block does not run.

To handle different capitalization:

```
language = "PYTHON"

if language.lower() == "python":
    print("Python selected")
```

---

16. `if` With User Input

User input makes `if` statements much more useful.

```
name = input("Enter your name: ")

if name == "Hafsa":
    print("Welcome, Hafsa")
```

Python:

1. waits for input
2. stores the input
3. compares it
4. executes the block if the condition is true

---

17. Numeric User Input

Remember that `input()` returns a string.

Therefore:

```
age = input("Enter your age: ")
```

does not give you an integer.

Convert it first:

```
age = int(input("Enter your age: "))

if age >= 18:
    print("Adult")
```

The flow is:

```
User Input
    ↓
String
    ↓
int()
    ↓
```

```
Number
    ↓
Comparison
    ↓
if
```

---

18. `if` With Membership

You can use `in` inside an `if`.

```
skills = ["Python", "JavaScript", "Flutter"]

if "Python" in skills:
    print("Python is available")
```

Another example:

```
role = "admin"

if role in ["admin", "manager"]:
    print("Staff access")
```

---

19. `if` With `not in`

```
blocked_users = ["Asim"]

username = "Hafsa"

if username not in blocked_users:
    print("User can continue")
```

This is useful for:

- blocked lists
- allowed values
- duplicate checks
- filtering
- validation

---

20. `if` With `None`

A common Python pattern is checking whether a value is missing.

```
value = None

if value is None:
    print("No value provided")
```

Use:

```
is None
```

rather than:

```
== None
```

The identity check is the standard Python style for this case.

---

21. Truthy and Falsy Values

Python allows many values to be used directly as conditions.

For example:

```
name = "Hafsa"

if name:
    print("Name exists")
```

A non-empty string is truthy.

An empty string is falsy:

```
name = ""

if name:
    print("Name exists")
```

The block does not run.

---

22. Empty Collections

Empty collections are also falsy.

```
items = []

if items:
    print("Items available")
```

Nothing happens because the list is empty.

With items:

```
items = ["Python"]

if items:
    print("Items available")
```

Output:

```
Items available
```

---

23. Common Falsy Values

Some commonly falsy values are:

```
False
None
0
0.0
""
[]
{}
set()
```

Many non-empty and non-zero values are truthy.

This allows concise conditions.

---

24. `if` Does Not Need `== True`

Avoid:

```
is_active = True

if is_active == True:
    print("Active")
```

Prefer:

```
if is_active:
    print("Active")
```

The second version is cleaner and more idiomatic.

---

25. Multiple Independent `if` Statements

You can have multiple `if` statements.

```
score = 85

if score >= 50:
    print("Pass")

if score >= 80:
    print("Excellent")
```

Both conditions are evaluated independently.

Output:

```
Pass
Excellent
```

This is different from an `if` / `elif` chain.

---

26. `if` vs `elif`

An important distinction:

Multiple `if` statements

Each condition is checked independently.

```
if score >= 50:
    print("Pass")

if score >= 80:
    print("Excellent")
```

Both can execute.

`if` with `elif`

Only the first matching branch executes.

```
if score >= 80:
    print("Excellent")
elif score >= 50:
    print("Pass")
```

Understanding this distinction becomes important when you learn `if...else` and `elif`.

---

27. Nested `if` Statements

An `if` can contain another `if`.

Example:

```
age = 25
has_account = True

if age >= 18:
    if has_account:
        print("Access granted")
```

The inner condition is checked only if the outer condition is true.

The execution flow is:

```
age >= 18?
   ↓
Yes
   ↓
has_account?
   ↓
Yes
   ↓
Access granted
```

---

28. When Nested `if` Is Useful

Nested conditions can represent dependent decisions.

For example:

```
if user_exists:
    if account_active:
        print("Continue")
```

The second question only matters if the user exists.

However, deeply nested code can become difficult to read.

Sometimes this is clearer:

```
if user_exists and account_active:
    print("Continue")
```

Use the structure that best communicates the logic.

---

29. `if` With Arithmetic Expressions

Conditions can contain calculations.

```
price = 1500
quantity = 4

if price * quantity >= 5000:
    print("Free shipping")
```

Python first evaluates:

```
price * quantity
```

Then compares the result:

```
>= 5000
```

---

30. `if` With Function Results

A function result can be used directly in a condition.

```
name = input("Enter your name: ")

if len(name) >= 3:
    print("Valid name")
```

Here:

```
len(name) >= 3
```

is the condition.

---

31. `if` and Data Validation

Validation is one of the most common uses of `if`.

Example:

```
score = 85

if 0 <= score <= 100:
    print("Valid score")
```

Another:

```
username = "Hafsa"
```

```
if username and 3 <= len(username) <= 20:
    print("Valid username")
```

The program can reject invalid data before processing it.

---

32. haas.dev Example

Imagine a haas.dev resource system.

A resource should only be shown when it is published.

```
published = True

if published:
    print("Show resource")
```

Now add authentication:

```
published = True
user_logged_in = True

if published and user_logged_in:
    print("Show resource")
```

The `if` statement turns a product rule into executable logic.

---

33. E Commerce Example

Suppose a product is purchasable only when it is available.

```
stock = 8

if stock > 0:
    print("Add to cart")
```

If:

```
stock = 0
```

the block is skipped.

A simple `if` statement can therefore represent an actual application rule.

---

34. Banking Example

Suppose a user wants to withdraw money.

```
balance = 5000
withdrawal = 2000

if withdrawal <= balance:
    print("Withdrawal allowed")
```

The condition prevents the program from approving a withdrawal larger than the balance.

---

35. Permission Example

```
is_admin = True

if is_admin:
    print("Open admin dashboard")
```

In a real application, the condition would normally be based on authenticated and authorized user information rather than a hardcoded Boolean.

---

36. Feature Availability Example

Imagine an application has a feature flag.

```
new_dashboard_enabled = True

if new_dashboard_enabled:
    print("Show new dashboard")
```

This pattern is common in real software systems.

A feature can be turned on or off without rewriting the entire application flow.

---

37. `if` and Business Rules

A useful way to think about `if` statements is:

```
Business Rule
     ↓
Condition
     ↓
True / False
     ↓
Action
```

For example:

```
Order is large enough
     ↓
total >= 5000
     ↓
True
     ↓
Free shipping
```

Python:

```
if total >= 5000:
    shipping = 0
```

This is how programming turns real-world rules into executable logic.

---

38. Avoid Giant Conditions

This can become difficult to read:

```
if age >= 18 and account_active and not blocked and has_subscription and
is_admin:
    print("Access")
```

If the rule is genuinely complex, break it into meaningful pieces:

```
is_adult = age >= 18
valid_account = account_active and not blocked
has_access = is_admin or has_subscription

if is_adult and valid_account and has_access:
    print("Access")
```

Named conditions make the program easier to understand.

---

39. Keep Conditions Focused

Good:

```
if stock > 0:
    print("Available")
```

Less clear:

```
if stock > 0 and price > 0 and product_name and category and user_logged_in
and not blocked:
```

```
    ...
```

If many unrelated rules are being checked, consider separating validation and business logic.

---

40. Common Mistake: Forgetting the Colon

Incorrect:

```
if age >= 18
    print("Adult")
```

Correct:

```
if age >= 18:
    print("Adult")
```

The colon is required.

---

41. Common Mistake: Incorrect Indentation

Incorrect:

```
if age >= 18:
print("Adult")
```

Correct:

```
if age >= 18:
    print("Adult")
```

Indentation defines the block.

---

42. Common Mistake: Using `=` Instead of `==`

Incorrect:

```
if age = 18:
    print("18")
```

Correct:

```
if age == 18:
    print("18")
```

Remember:

```
=   → assignment
==  → comparison
```

---

43. Common Mistake: Comparing Input Without Conversion

Incorrect:

```
age = input("Enter age: ")

if age >= 18:
    print("Adult")
```

Correct:

```
age = int(input("Enter age: "))

if age >= 18:
    print("Adult")
```

---

44. Common Mistake: Incorrect `or` Usage

Incorrect:

```
if role == "admin" or "manager":
    print("Staff")
```

Correct:

```
if role == "admin" or role == "manager":
    print("Staff")
```

Or:

```
if role in ["admin", "manager"]:
    print("Staff")
```

---

45. Common Mistake: Forgetting Boundary Values

Suppose:

```
if score > 50:
```

A score of exactly 50 will fail.

If the requirement is:

> 50 or above

use:

```
if score >= 50:
```

Always translate requirements carefully.

---

46. Testing an `if` Statement

Do not test only one situation.

Suppose:

```
if age >= 18:
    print("Adult")
```

Test:

```
17 → condition False
18 → condition True
19 → condition True
```

The value exactly at the boundary is especially important.

---

47. Decision Table

Before implementing complicated rules, a decision table can help.

Suppose access requires:

```
age >= 18
AND
account active
```

Age Valid	Account Active	Access
False	False	No
False	True	No
True	False	No
True	True	Yes

Then implement:

```
if age >= 18 and account_active:
    print("Access granted")
```

This is a simple way to reason about conditions before writing code.

---

48. Mini Project: Age Checker

Create:

```
age = int(input("Enter your age: "))

if age >= 18:
    print("You are an adult")
```

Test it with:

```
15
18
25
```

Observe when the message appears.

---

49. Mini Project: Even Number Checker

```
number = int(input("Enter a number: "))

if number % 2 == 0:
    print("Even number")
```

The key expression is:

```
number % 2 == 0
```

The remainder is zero for even numbers.

---

50. Mini Project: Free Shipping

```
total = float(input("Enter order total: "))

if total >= 5000:
    print("Free shipping")
```

This represents a real business rule.

---

51. Mini Project: Resource Availability

```
resources_available = 12

if resources_available > 0:
    print("Resources available")
```

You could later expand this into a complete resource management system.

---

52. Mini Project: Login Check

```
username = input("Username: ")
password = input("Password: ")

if username == "Hafsa" and password == "python123":
    print("Login successful")
```

This demonstrates:

- input
- string comparison
- logical operators
- an `if` statement

In a real application, credentials should never be hardcoded like this; authentication should use secure password handling and a proper identity system.

---

53. Five Minute Challenge

Build a simple order eligibility checker.

Create:

```
order_total
stock
```

The order should be accepted only when:

```
order total is at least 1000
AND
stock is greater than 0
```

Expected logic:

```
if order_total >= 1000 and stock > 0:
    print("Order can be placed")
```

Then test:

```
order_total = 1500, stock = 5
order_total = 500, stock = 5
order_total = 1500, stock = 0
```

---

54. Practice Exercises

Easy

Exercise 1

Create:

```
age = 20
```

Print `"Adult"` if the age is at least 18.

Exercise 2

Create:

```
number = 10
```

Print `"Positive"` if the number is greater than zero.

Exercise 3

Create:

```
score = 80
```

Print `"Passed"` if the score is at least 50.

Exercise 4

Check whether:

```
name = "Hafsa"
```

is equal to `"Hafsa"`.

Exercise 5

Create:

```
items = ["Python", "JavaScript"]
```

Print `"Python found"` when Python exists in the list.

---

55. Medium Exercises

Exercise 6: Even Number

Ask the user for a number and print `"Even"` if it is divisible by 2.

Exercise 7: Age Validation

Ask for an age and print `"Valid age"` when:

```
0 <= age <= 120
```

Exercise 8: Password Validation

Ask the user for a password.

Print `"Password accepted"` if its length is at least 8 characters.

Exercise 9: Shipping Rule

Ask for an order total.

Print `"Free shipping"` when the total is at least 5000.

Exercise 10: Account Access

Create:

```
account_active
has_account
```

Print `"Access granted"` only when both are true.

---

56. Hard Exercises

Exercise 11: Product Purchase

A product can be purchased only when:

```
stock > 0
AND
price > 0
AND
user is logged in
```

Implement the condition.

Exercise 12: Resource Access

A resource can be opened when:

```
resource is published
AND
resource is either free OR user is logged in
```

Use parentheses where appropriate.

Exercise 13: Username Validation

A username is valid when:

```
username is not empty
AND
length is between 3 and 20
```

Implement the condition.

Exercise 14: Exam Eligibility

A student can take an exam when:

```
attendance >= 75
AND
fee_paid
```

Write the condition.

Exercise 15: Admin Access

A dashboard is accessible when:

```
user is admin
AND
account is active
AND
account is not blocked
```

Implement it.

---

57. Interview Questions

What is an `if` statement?

An `if` statement executes a block of code only when its condition evaluates to true.

What is the syntax of an `if` statement?

```
if condition:
    statement
```

Why is indentation important in Python?

Indentation defines code blocks. The statements inside an `if` block must be indented.

What happens when an `if` condition is false?

Python skips the corresponding indented block and continues with the code after it.

Can an `if` condition contain multiple conditions?

Yes.

For example:

```
if age >= 18 and account_active:
    ...
```

Can an `if` statement use a function call?

Yes.

```
if len(name) >= 3:
    print("Valid")
```

What is a truthy value?

A value that Python interprets as true when used in a Boolean context.

What is a falsy value?

A value that Python interprets as false in a Boolean context, such as `False`, `None`, `0`, `""`, and empty collections.

What is the difference between multiple `if` statements and an `if`/`elif` chain?

Multiple `if` statements are evaluated independently.

An `if` / `elif` chain selects the first matching branch.

Can `if` statements be nested?

Yes.

```
if condition_a:
    if condition_b:
        print("Both conditions passed")
```

---

58. `if` Statement Cheat Sheet

Basic

```
if condition:
    print("Condition is true")
```

Boolean variable

```
if is_active:
    print("Active")
```

Comparison

```
if score >= 50:
    print("Pass")
```

Multiple conditions

```
if age >= 18 and has_account:
    print("Access")
```

Either condition

```
if is_admin or is_manager:
    print("Staff")
```

Negation

```
if not blocked:
    print("Allowed")
```

Membership

```
if "Python" in skills:
    print("Found")
```

Missing value

```
if value is None:
    print("Missing")
```

Range

```
if 0 <= score <= 100:
    print("Valid")
```

---

59. The `if` Problem Solving Framework

When you need an `if` statement, follow these steps.

Step 1: Identify the decision

Ask:

> What does my program need to decide?

Example:

```
Can the user access the resource?
```

Step 2: Identify the facts

```
User is logged in
Resource is published
```

Step 3: Turn facts into conditions

```
user_logged_in
resource_published
```

Step 4: Connect them

```
if user_logged_in and resource_published:
```

Step 5: Define the action

```
if user_logged_in and resource_published:
    print("Open resource")
```

Step 6: Test both sides

Test:

```
True + True
True + False
False + True
False + False
```

This approach helps prevent logic errors.

---

60. From `if` to Real Programs

An `if` statement is one of the first major steps from syntax to programming logic.

A real application can follow:

```
User Input
     ↓
Validation
     ↓
Comparison
     ↓
if Condition
     ↓
Business Rule
     ↓
Action
```

For example:

```
order_total = 6500
stock = 8

if order_total >= 5000 and stock > 0:
    print("Order accepted")
```

The program has transformed a real-world requirement into executable logic.

---

61. Key Takeaways

- `if` statements allow programs to make decisions.
- An `if` block runs only when its condition is true.
- A colon `:` is required after the condition.
- Indentation defines the `if` block.

- Conditions commonly use comparison operators.
- Conditions can use `and`, `or`, and `not`.
- Boolean variables can be used directly as conditions.
- Python supports truthy and falsy values.
- `input()` should be converted when numerical comparisons are needed.
- `in` and `not in` can be used for membership checks.
- `is None` is the standard way to check for `None`.
- Multiple `if` statements are evaluated independently.
- Nested `if` statements allow dependent decisions.
- Parentheses can make complex conditions clearer.
- Boundary values should always be tested.
- Complex business rules are easier to maintain when broken into named conditions.
- `if` statements are the foundation of decision-making logic in Python.

Final Mental Model

Think of an `if` statement as a decision gate:

```
              FACTS
                ↓
            CONDITION
                ↓
        ┌─────────────────┐
        │ True or False   │
        └─────────────────┘
             ↓       ↓
           True     False
             ↓         ↓
       Run block    Skip block
             ↓         ↓
             └────┬────┘
                  ↓
            Continue Program
```

The most important skill is not memorizing:

```
if condition:
```

The real skill is learning to translate a real-world rule into a precise condition.

For example:

```
"Only active users who are 18 or older can access this feature."
```

becomes:

```
if age >= 18 and account_active:
    print("Access granted")
```

That is the beginning of turning **requirements into working software**.

---

haas.dev

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

Visit haas.dev for more resources and guides.