# Python `if...else`: Making Decisions in Your Programs

**Subtitle:** Learn how Python evaluates conditions and chooses different actions, from simple decisions to practical real-world program logic.

**Website Name:** haas.dev

**Website Link:** https://dev-roast-app.vercel.app

---

## 1. The Problem: Programs Need to Make Decisions

A program is not useful if it can only execute the same instructions every time.

Real programs constantly make decisions:

- Is the user logged in?
- Is the password correct?
- Is the payment successful?
- Is the product available?
- Is the student's score high enough to pass?
- Is the user old enough to access a feature?
- Should a button be displayed?
- Should an error message be shown?

Consider a simple application:

```
age = 20
```

The program might need to decide:

> If the user is 18 or older, allow access. Otherwise, deny access.

This is where `if...else` becomes important.

Python allows us to tell the program:

```
IF something is true
    do this

ELSE
    do something different
```

That is the foundation of decision-making in Python.

---

## 2. What Is `if...else`?

`if...else` is a **conditional control structure**.

It allows Python to choose between different blocks of code based on whether a condition is `True` or `False`.

The basic structure is:

```
if condition:
    # code when condition is True
else:
    # code when condition is False
```

The condition is evaluated first.

If the condition is `True`, the `if` block executes.

If the condition is `False`, Python skips the `if` block and executes the `else` block.

---

# 3. Your First `if...else` Program

```
age = 20

if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

Output:

```
You are an adult.
```

Let's understand what happened.

### Step 1

Python stores:

```
age = 20
```

### Step 2

Python evaluates:

```
age >= 18
```

This becomes:

```
20 >= 18
```

The result is:

```
True
```

### Step 3

Because the condition is `True`, Python executes:

```
print("You are an adult.")
```

The `else` block is ignored.

---

# 4. What Happens When the Condition Is False?

Consider:

```
age = 15

if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

Python evaluates:

```
15 >= 18
```

The result is:

```
False
```

Therefore Python skips:

```
print("You are an adult.")
```

and executes:

```
print("You are a minor.")
```

Output:

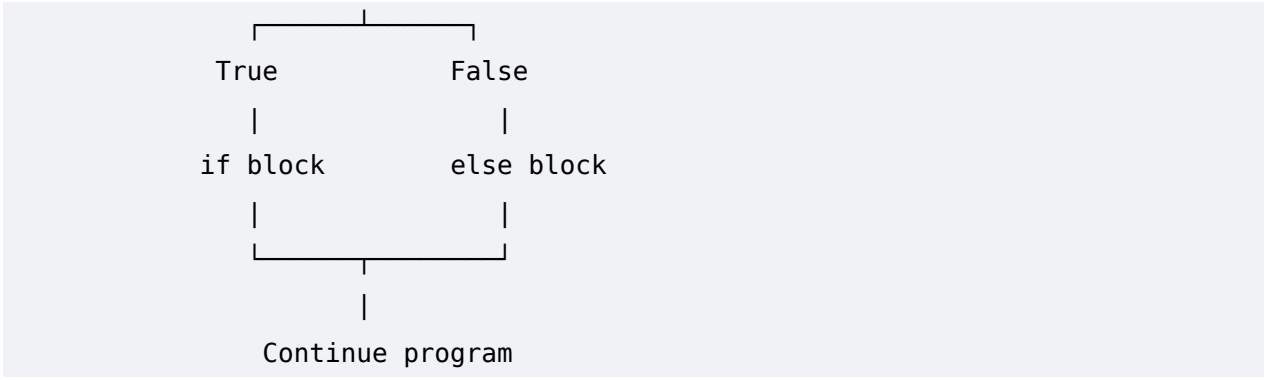
```
You are a minor.
```

---

# 5. The Mental Model

Always think about `if...else` like this:

```
                Condition
                    |
```




Only **one** of the two branches executes.

That is the most important idea to understand.

---

# 6. Conditions Produce Boolean Results

An `if` statement works with a condition.

A condition ultimately evaluates to:

```
True
```

or:

```
False
```

For example:

```
10 > 5
```

produces:

```
True
```

while:

```
10 < 5
```

produces:

```
False
```

You can even test this directly:

```
print(10 > 5)
print(10 < 5)
```

Output:

```
True
False
```

---

# 7. Comparison Operators with `if...else`

Conditions frequently use comparison operators.

| Operator | Meaning |
|---|---|
| `==` | equal to |
| `!=` | not equal to |
| `>` | greater than |
| `<` | less than |
| `>=` | greater than or equal to |
| `<=` | less than or equal to |

---

## Example: Equality

```
password = "python123"

if password == "python123":
    print("Login successful.")
else:
    print("Incorrect password.")
```

---

## Example: Greater Than

```
score = 85

if score > 50:
```

```
    print("Passed.")
else:
    print("Failed.")
```

---

### Example: Less Than

```
temperature = 15

if temperature < 20:
    print("It is cold.")
else:
    print("It is warm.")
```

---

## 8. `==` vs `=`

This is one of the most common beginner mistakes.

**Assignment:**

```
age = 20
```

This means:

> Store `20` in `age`.

**Comparison:**

```
age == 20
```

This means:

> Is `age` equal to `20`?

They have completely different purposes.

---

## 9. Python Indentation Is Part of the Syntax

Python uses indentation to identify which statements belong to the `if` or `else` block.

Correct:

```
age = 20

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

The indentation tells Python that:

```
print("Adult")
```

belongs to the `if` block.

And:

```
print("Minor")
```

belongs to the `else` block.

---

# 10. What Happens Without Indentation?

This is incorrect:

```
age = 20

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

Python will produce an indentation-related error.

Python intentionally uses indentation to make code structure visually clear.

---

# 11. The Colon `:` Matters

An `if` statement must end with a colon:

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

The same applies to `else`:

```
else:
```

Correct:

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

Incorrect:

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

---

## 12. Using User Input

Conditional statements become much more useful when combined with input.

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

if age >= 18:
    print("Access granted.")
else:
    print("Access denied.")
```

Suppose the user enters:

```
22
```

Python converts the input to an integer:

```
22
```

Then evaluates:

```
22 >= 18
```

The result is `True`.

Output:

```
Access granted.
```

---

## 13. Why `int()` Is Important Here

Remember that `input()` returns text.

For example:

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

Even if the user types:

```
20
```

the value is initially a string.

If you want to perform numerical comparisons, convert it:

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

Now Python has an integer.

---

# 14. A Simple Login Example

Conditional logic appears everywhere in authentication.

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

if username == "admin" and password == "1234":
    print("Login successful.")
else:
    print("Invalid username or password.")
```

The program checks whether the required conditions are satisfied.

If both are correct:

```
Login successful.
```

Otherwise:

```
Invalid username or password.
```

This is a simplified example, not how production authentication should store passwords.

---

# 15. Real-World Example: haas.dev Resource Access

Imagine a simplified resource system on **haas.dev**.

A user wants to access a resource.

```
is_logged_in = True

if is_logged_in:
    print("You can access the resource.")
else:
    print("Please log in first.")
```

The program doesn't need to ask the same question repeatedly.

It simply checks the current state and chooses an action.

This is the fundamental pattern behind many applications.

---

# 16. Working with Boolean Variables

Python's Boolean values can be used directly.

```
is_logged_in = True

if is_logged_in:
    print("Welcome back!")
else:
    print("Please log in.")
```

You don't need:

```
if is_logged_in == True:
```

Although that can work, the simpler version is clearer:

```
if is_logged_in:
```

---

# 17. Another Boolean Example

```
is_available = False

if is_available:
    print("Product available.")
else:
    print("Product is out of stock.")
```

Output:

```
Product is out of stock.
```

This pattern is extremely common in application development.

---

# 18. Checking Strings

You can compare strings using `if...else`.

```
role = "admin"

if role == "admin":
    print("You have administrator access.")
else:
    print("You have regular user access.")
```

Python compares the actual string values.

---

# 19. Case Sensitivity

Python string comparisons are case-sensitive.

```
role = "Admin"

if role == "admin":
    print("Admin access")
else:
    print("Regular access")
```

Output:

```
Regular access
```

Because:

```
"Admin" != "admin"
```

If you want case-insensitive comparison:

```
role = "Admin"

if role.lower() == "admin":
    print("Admin access")
else:
    print("Regular access")
```

---

# 20. Checking Whether a Number Is Even

The modulo operator can be combined with conditions.

```
number = 8

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

Why?

The remainder of an even number divided by 2 is zero.

```
8 % 2 = 0
```

Therefore:

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

is `True`.

---

## 21. Checking a Product Stock

```
stock = 12

if stock > 0:
    print("Product is available.")
else:
    print("Product is out of stock.")
```

This is a simplified version of logic that exists in e-commerce systems.

---

## 22. Checking a User's Balance

```
balance = 5000
withdrawal = 3000

if withdrawal <= balance:
    print("Withdrawal approved.")
else:
    print("Insufficient balance.")
```

The program compares two values before deciding what to do.

---

## 23. `if...else` with Arithmetic

Conditions don't have to compare variables directly.

```
price = 1000
discount = 200

if price - discount > 500:
    print("Premium purchase.")
else:
    print("Standard purchase.")
```

Python evaluates the expression first.

---

## 24. Multiple Conditions with `and`

Sometimes one condition isn't enough.

The `and` operator requires both conditions to be true.

```
age = 25
has_ticket = True

if age >= 18 and has_ticket:
    print("You can enter.")
else:
    print("Entry denied.")
```

Both must be true.

---

## 25. Multiple Conditions with `or`

`or` requires at least one condition to be true.

```
is_admin = False
is_moderator = True

if is_admin or is_moderator:
    print("You can manage content.")
else:
    print("Access denied.")
```

Because `is_moderator` is `True`, access is allowed.

---

## 26. Using `not`

`not` reverses a Boolean result.

```
is_blocked = False

if not is_blocked:
    print("User can continue.")
else:
    print("User is blocked.")
```

Since:

```
not False
```

is:

```
True
```

the first block executes.

---

## 27. `if...else` with Lists

Conditions can also work with collections.

```
cart = ["Laptop", "Mouse"]

if cart:
    print("Cart contains products.")
else:
    print("Cart is empty.")
```

An empty list is considered false-like in a condition.

---

## 28. Truthy and Falsy Values

Python does not only understand literal `True` and `False`.

Some values behave as false in conditions.

Common falsy values include:

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

Most other values are truthy.

---

### Example:

```
username = ""

if username:
    print("Username entered.")
else:
    print("Username is empty.")
```

Because the string is empty, the `else` block executes.

---

## 29. Checking for Empty Input

This is a practical pattern.

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

if name:
    print("Welcome,", name)
else:
    print("Name cannot be empty.")
```

This is more useful than simply checking whether the input equals a particular string.

---

## 30. Nested `if...else`

An `if` statement can contain another `if`.

Example:

```
age = 25
has_id = True

if age >= 18:
    if has_id:
        print("Entry allowed.")
    else:
        print("ID required.")
else:
    print("You are underage.")
```

Execution works from the outside toward the inside.

First:

```
age >= 18
```

is checked.

Only if it is true does Python check:

```
has_id
```

---

## 31. When Should You Use Nested Conditions?

Nested conditions can be useful when one decision depends on another.

For example:

```
Is the user logged in?
        |
       Yes
        |
Is the account verified?
        |
       Yes
        |
Allow access
```

However, deeply nested conditions can become difficult to maintain.

Good developers don't simply make conditions work.

They also think about:

> **How easy will this logic be to understand six months from now?**

---

# 32. `if...else` vs `if` Alone

Sometimes you don't need an `else`.

Example:

```
age = 20

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

If the condition is false, Python simply does nothing.

Use `else` when you specifically need an alternative action.

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

---

# 33. Common Beginner Mistakes

## Mistake 1: Forgetting the colon

Incorrect:

```
if age >= 18
```

Correct:

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

---

## Mistake 2: Incorrect indentation

Incorrect:

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

Correct:

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

---

## Mistake 3: Using `=` instead of `==`

Incorrect concept:

```
if age = 18:
```

Correct comparison:

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

---

## Mistake 4: Forgetting input conversion

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

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

This can cause a type-related problem because `age` is a string.

Better:

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

---

## Mistake 5: Overcomplicating simple logic

Don't write complicated conditions when a simple one is enough.

Instead of unnecessary comparisons with Boolean values:

```
if is_logged_in == True:
```

prefer:

```
if is_logged_in:
```

---

# 34. Practical Exercise 1: Age Checker

Create a program that:

1. asks for the user's age
2. converts it to an integer
3. checks whether the user is 18 or older
4. prints an appropriate message

Expected behavior:

```
Enter your age: 21
You can continue.
```

---

# 35. Practical Exercise 2: Number Checker

Create a program that checks whether a number is:

- positive
- or not positive

Starter:

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

if number > 0:
    print("Positive")
else:
    print("Zero or negative")
```

Then test it with:

```
10
0
-5
```

Observe how the same program follows different execution paths.

---

# 36. Practical Exercise 3: Login System

Create a simple login simulation.

Requirements:

```
Username: admin
Password: python123
```

If both match:

```
Login successful
```

Otherwise:

```
Invalid credentials
```

---

# 37. Practical Exercise 4: Shopping Eligibility

Create a program with:

```
balance = 5000
price = 3500
```

If the balance is enough:

```
Purchase successful
```

Otherwise:

```
Insufficient balance
```

Then modify the values and test different scenarios.

---

# 38. Practical Exercise 5: haas.dev Resource Access

Create a simplified resource-access system.

Variables:

```
is_logged_in = True
has_access = False
```

Your program should decide whether the user can access a premium resource.

Think carefully about what should happen in each state.

Test:

```
True, True
True, False
False, True
False, False
```

---

# 39. Mini Project: Simple Admission Checker

Build a small Python program that checks whether a student qualifies for admission.

Inputs:

```
Student name
Age
Test score
```

Rules:

- age must be at least 18
- score must be at least 60

Example:

```
Enter your name: Sara
Enter your age: 20
Enter your score: 75

Admission approved.
```

If the requirements aren't met:

```
Admission not approved.
```

### Starter structure:

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

if age >= 18 and score >= 60:
    print("Admission approved.")
else:
    print("Admission not approved.")
```

Now improve it by displaying **why** admission was not approved.

---

# 40. Debugging Conditional Logic

When an `if...else` program produces the wrong result, don't randomly change the code.

Trace the condition.

Suppose:

```
age = 17

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

```
else:
    print("Denied")
```

Ask:

### What is the actual value?

```
age = 17
```

### What comparison is being performed?

```
17 >= 18
```

### What is the result?

```
False
```

### Which branch should execute?

```
else
```

This approach makes debugging logical rather than guess-based.

---

# 41. Interview Questions

### Q1. What is the purpose of `if...else`?

It allows a program to choose between two execution paths based on whether a condition is true or false.

### Q2. What happens if an `if` condition is false?

Python skips the `if` block and executes the `else` block, if one exists.

### Q3. What does `==` do?

It compares two values for equality.

### Q4. What does `=` do?

It assigns a value to a variable.

### Q5. Why is indentation important in Python?

Python uses indentation to define code blocks.

### Q6. Can an `if` statement exist without `else`?

Yes.

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

### Q7. Can conditions use multiple logical operators?

Yes. Conditions can use operators such as:

```
and
or
not
```

---

# 42. 5-Minute Challenge

Write a program for a simple website account.

The program should have:

```
is_logged_in
is_verified
```

Rules:

- If the user is logged in **and** verified → show dashboard.
- Otherwise → show an appropriate message.

Then modify the program so that it handles these four states differently:

```
Not logged in
Logged in but not verified
Verified but not logged in
Logged in and verified
```

The goal is not just to make the code run.

The goal is to understand how each condition changes the program's execution path.

---

# 43. Challenge: Build a Discount Checker

Create a shopping discount system.

Requirements:

```
cart_total >= 5000 → 20% discount
cart_total >= 3000 → 10% discount
otherwise → no discount
```

For now, use only `if...else` and see whether you can represent the requirements clearly.

Then compare your solution with the next conditional concept you learn.

---

# 44. Engineering Thinking: Conditions Are Business Rules

One of the most important lessons is that `if...else` isn't just a beginner programming feature.

In real software, conditions often represent **business rules**.

For example:

```
if balance >= withdrawal:
```

represents a banking rule.

```
if stock > 0:
```

represents an inventory rule.

```
if user.is_verified:
```

represents an account rule.

```
if score >= passing_score:
```

represents an education rule.

The syntax is simple.

The difficult part is correctly translating real-world requirements into logical conditions.

---

# 45. From Requirements to Conditions

Suppose a client says:

> "Only logged-in users should be allowed to download this file."

Translate the requirement into a Boolean question:

```
Is the user logged in?
```

Then into code:

```
if is_logged_in:
    print("Download started.")
else:
    print("Please log in first.")
```

This is a fundamental developer skill:

> **Turn a human requirement into a precise condition.**

---

# 46. Multiple Real-World Decision Examples

### Website access

```
if is_logged_in:
    print("Open dashboard")
else:
    print("Open login page")
```

### Payment

```
if payment_successful:
    print("Order confirmed")
else:
    print("Payment failed")
```

### Inventory

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

### Authentication

```
if password_correct:
    print("Access granted")
else:
    print("Access denied")
```

### Validation

```
if email:
    print("Continue")
else:
    print("Email required")
```

These all follow the same fundamental pattern.

---

# 47. Key Takeaways

Remember these rules:

1. `if` checks a condition.
2. The condition evaluates to `True` or `False`.
3. If it is `True`, the `if` block executes.
4. If it is `False`, the `else` block executes.
5. Only the appropriate branch executes.

6. Python uses indentation to define blocks.
7. `:` is required after `if` and `else`.
8. `=` assigns values.
9. `==` compares values.
10. `and`, `or`, and `not` can build more complex conditions.
11. Boolean values can be used directly.
12. Real applications use conditional logic to implement business rules.

---

# 48. Cheat Sheet

### Basic structure

```
if condition:
    # True
else:
    # False
```

### Comparison

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

### Boolean

```
if is_logged_in:
    print("Dashboard")
else:
    print("Login")
```

### `and`

```
if age >= 18 and has_id:
    print("Allowed")
else:
    print("Denied")
```

### `or`

```
if is_admin or is_moderator:
    print("Access granted")
```

```
else:
    print("Access denied")
```

### `not`

```
if not is_blocked:
    print("Continue")
else:
    print("Blocked")
```

### User input

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

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

---

## Final Practice Task

Build a **Simple Student Result Checker**.

Your program should:

1. Ask for the student's name.
2. Ask for marks.
3. Determine whether the student passed.
4. Display the student's name.
5. Display either:
    - `Passed`
    - `Failed`
6. Test your program with several different marks.
7. Trace the condition manually for at least two inputs.

Then ask yourself:

> **What happens when a program needs more than two possible outcomes?**

That question leads naturally to the next stage of Python conditional logic.

Visit haas.dev for more resources and guides.

haas.dev

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