Python `elif` and Multiple Conditions

Learn how to make Python choose between multiple possible outcomes instead of stopping at the first `if`.

haas.dev

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

---

1. Why Do We Need `elif`?

In real programs, a decision is rarely just:

> If this is true, do something. Otherwise, do something else.

Often, a program has **three, four, five, or even more possible outcomes**.

For example:

A student receives a different grade depending on their marks:

- 90 or above → A
- 80–89 → B
- 70–79 → C
- 60–69 → D
- Below 60 → Fail

You could try to solve this using many separate `if` statements, but that can make the program unnecessarily complicated.

Python provides `elif` for exactly this situation.

```
if condition1:
    # first option
elif condition2:
    # second option
elif condition3:
    # third option
else:
    # fallback option
```

`elif` means:

> **"If the previous condition was false, check this condition."**

---

2. What Does `elif` Mean?

`elif` is short for:

```
else if
```

It allows Python to check another condition **only when the conditions before it were false**.

For example:

```
age = 20

if age < 13:
    print("Child")
elif age < 18:
    print("Teenager")
else:
    print("Adult")
```

Python evaluates the conditions from top to bottom.

For `age = 20`:

```
age < 13
False

age < 18
False

else
True
```

Output:

```
Adult
```

---

3. The Basic Structure

The general structure is:

```
if condition:
    statement
elif condition:
    statement
elif condition:
    statement
else:
    statement
```

There are four important parts:

`if`

The first condition Python checks.

`elif`

An additional condition checked only if previous conditions were false.

`else`

The fallback option when none of the previous conditions is true.

Indentation

Indentation tells Python which statements belong to each condition.

---

4. How Python Executes `if` + `elif` + `else`

Consider:

```
temperature = 30

if temperature < 10:
    print("Cold")
elif temperature < 25:
    print("Cool")
elif temperature < 35:
    print("Warm")
else:
    print("Hot")
```

Python does not execute every block.

It checks them sequentially.

Step 1

```
temperature < 10
```

Result:

```
False
```

Python moves to the next condition.

Step 2

```
temperature < 25
```

Result:

```
False
```

Python continues.

Step 3

```
temperature < 35
```

Result:

```
True
```

Python executes:

```
print("Warm")
```

Then it stops checking the remaining conditions.

Output:

```
Warm
```

This is one of the most important things to understand about `elif`:

> **Once Python finds a true condition in an `if` / `elif` chain, it executes that block and skips the remaining conditions.**

---

5. `if` vs `elif`

Understanding this difference prevents many beginner mistakes.

Multiple Independent `if` Statements

```
age = 20

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

if age >= 13:
    print("Teenager")
```

Both conditions are checked independently.

Output:

```
Adult
Teenager
```

That is probably not what we wanted.

Using `elif`

```
age = 20

if age >= 18:
    print("Adult")
elif age >= 13:
    print("Teenager")
```

Python checks the first condition.

It is true, so it prints:

```
Adult
```

It does not check the `elif`.

This makes `elif` useful when **only one outcome should be selected**.

---

6. Think of `elif` as a Decision Chain

Imagine a person standing in front of several doors.

Python checks:

```
Is condition 1 true?
        |
      No
        ↓
Is condition 2 true?
        |
      No
        ↓
Is condition 3 true?
        |
      Yes
        ↓
Choose this option
```

After selecting one option, Python leaves the chain.

This mental model is extremely useful.

---

7. A Simple Example

```
number = 7

if number > 10:
```

```
    print("Greater than 10")
elif number == 10:
    print("Equal to 10")
else:
    print("Less than 10")
```

Output:

```
Less than 10
```

Python checks:

```
7 > 10 → False
7 == 10 → False
else → True
```

---

8. Multiple `elif` Conditions

You can have more than one `elif`.

```
marks = 75

if marks >= 90:
    print("A")
elif marks >= 80:
    print("B")
elif marks >= 70:
    print("C")
elif marks >= 60:
    print("D")
else:
    print("F")
```

Output:

```
C
```

The structure is:

```
90+       → A
80–89     → B
70–79     → C
60–69     → D
Below 60  → F
```

---

9. Why Order Matters

This is one of the most important concepts in conditional programming.

Look at this code:

```
marks = 95

if marks >= 60:
    print("D")
elif marks >= 70:
    print("C")
elif marks >= 80:
    print("B")
elif marks >= 90:
    print("A")
```

What happens?

The first condition is:

```
marks >= 60
```

For 95, this is true.

Python immediately prints:

```
D
```

It never reaches the `A` condition.

The problem is not Python.

The problem is the **order of the conditions**.

Correct:

```
if marks >= 90:
    print("A")
elif marks >= 80:
    print("B")
elif marks >= 70:
    print("C")
elif marks >= 60:
    print("D")
else:
    print("F")
```

General rule

When using ranges with `elif`, think carefully about which condition should be checked first.

A useful pattern is:

```
Most specific/highest threshold
            ↓
Next threshold
            ↓
Next threshold
            ↓
Fallback
```

---

10. Another Way to Think About Conditions

Suppose you are checking age categories.

```
age = 16

if age < 5:
    print("Toddler")
elif age < 13:
    print("Child")
elif age < 18:
    print("Teenager")
else:
    print("Adult")
```

Notice something interesting.

We don't need to write:

```
elif age >= 13 and age < 18:
```

because the earlier conditions have already been eliminated.

When Python reaches:

```
elif age < 18:
```

we already know:

```
age >= 13
```

because:

```
age < 13
```

was false.

This can make conditional code much cleaner.

---

11. Using Comparison Operators

`elif` can work with all common comparison operators.

Greater than

```
if score > 90:
    print("Excellent")
```

Less than

```
elif score < 50:
    print("Needs improvement")
```

Equal to

```
elif score == 75:
    print("Exactly 75")
```

Greater than or equal to

```
elif score >= 80:
    print("Very good")
```

Less than or equal to

```
elif score <= 40:
    print("Fail")
```

---

12. Using `elif` With Strings

Conditions don't have to involve numbers.

You can compare strings too.

```
day = "Monday"

if day == "Saturday":
    print("Weekend")
elif day == "Sunday":
    print("Weekend")
else:
    print("Working day")
```

Output:

```
Working day
```

---

13. Multiple Conditions With `and`

Sometimes one decision depends on multiple conditions being true.

For example:

```
age = 22
has_id = True

if age >= 18 and has_id:
    print("Entry allowed")
else:
    print("Entry denied")
```

The `and` operator means:

> Both conditions must be true.

---

14. `elif` With `and`

You can combine `elif` with `and`.

```
age = 25
has_ticket = True

if age < 18:
    print("Not allowed")
elif age >= 18 and has_ticket:
    print("Entry allowed")
else:
    print("Ticket required")
```

The second condition requires both:

```
age >= 18
```

and:

```
has_ticket
```

to be true.

---

15. Multiple Conditions With `or`

`or` means at least one condition must be true.

```
day = "Saturday"

if day == "Saturday" or day == "Sunday":
    print("Weekend")
else:
    print("Weekday")
```

Output:

```
Weekend
```

You can also use it inside an `elif`:

```
day = "Friday"

if day == "Monday":
    print("Start of week")
elif day == "Friday" or day == "Saturday":
    print("Near or on weekend")
else:
    print("Regular day")
```

---

16. Combining `and`, `or`, and `elif`

Real programs often require more complex decisions.

For example:

```
age = 20
has_ticket = True
is_vip = False

if age < 18:
    print("Not allowed")
elif has_ticket and is_vip:
    print("VIP entry")
elif has_ticket:
    print("Regular entry")
else:
    print("No ticket")
```

Python evaluates the chain from top to bottom.

The first matching branch wins.

---

17. Parentheses Make Complex Conditions Easier to Understand

Consider:

```
if age >= 18 and has_ticket or is_vip:
    print("Allowed")
```

This can be difficult to read.

A clearer version is:

```
if (age >= 18 and has_ticket) or is_vip:
    print("Allowed")
```

Parentheses make your intended logic easier to see.

When conditions become complicated, prioritize **readability**, not cleverness.

---

18. Boolean Expressions Become Conditions

Remember that a condition ultimately produces a Boolean:

```
True
```

or:

```
False
```

For example:

```
age >= 18
```

is an expression.

If:

```
age = 20
```

then:

```
age >= 18
```

becomes:

```
True
```

Python then executes the corresponding branch.

---

19. Using User Input

`elif` becomes much more useful when combined with `input()`.

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

if marks >= 90:
    print("Grade A")
elif marks >= 80:
    print("Grade B")
elif marks >= 70:
    print("Grade C")
elif marks >= 60:
    print("Grade D")
else:
    print("Fail")
```

If the user enters:

```
85
```

Output:

```
Grade B
```

The complete flow is:

```
User enters data
       ↓
input() receives text
       ↓
int() converts it to a number
       ↓
Python checks conditions
       ↓
First true branch executes
       ↓
Program continues
```

---

20. Why `int()` Matters Here

Remember that `input()` returns a string.

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

Even if the user enters:

```
90
```

Python receives:

```
"90"
```

not:

```
90
```

For numeric comparisons, convert it:

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

Now Python can correctly evaluate:

```
marks >= 90
```

---

21. A Real Grade Calculator

Let's build a small program.

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

if marks < 0 or marks > 100:
    print("Invalid marks")
elif marks >= 90:
    print("Grade: A")
elif marks >= 80:
    print("Grade: B")
elif marks >= 70:
    print("Grade: C")
elif marks >= 60:
    print("Grade: D")
else:
    print("Grade: F")
```

This demonstrates an important engineering idea:

> **Validate invalid input before processing normal cases.**

The program first asks:

```
Is the input impossible?
```

If yes, it stops the normal grading logic.

Otherwise, it evaluates the valid range.

---

22. `elif` and Business Rules

Conditional logic is not just a beginner programming exercise.

Businesses use rules like:

```
If customer is new → offer welcome discount

Else if customer is premium → offer premium discount

Else if customer has a coupon → apply coupon

Else → regular price
```

Python could represent this as:

```
if is_new_customer:
    discount = 20
elif is_premium_customer:
    discount = 15
elif has_coupon:
    discount = 10
else:
    discount = 0
```

This is business logic.

The programmer is translating a real-world rule into executable instructions.

---

23. haas.dev Example

Imagine haas.dev has different access levels.

```
user_type = "student"

if user_type == "admin":
    print("Full platform access")
```

```
elif user_type == "student":
    print("Student resources available")
elif user_type == "visitor":
    print("Public resources available")
else:
    print("Unknown user type")
```

The program decides what the user can access based on their role.

The same idea appears in:

- websites
- mobile applications
- dashboards
- payment systems
- learning platforms
- admin panels
- APIs
- games

---

24. Another haas.dev Example: Resource Availability

Imagine a resource has different states.

```
status = "published"

if status == "draft":
    print("Resource is not public yet")
elif status == "published":
    print("Resource is available")
elif status == "archived":
    print("Resource has been archived")
else:
    print("Unknown status")
```

This is a common software pattern.

Instead of thinking only in terms of:

```
if / elif / else
```

start thinking:

> **What states can this system have, and what should happen for each state?**

That is an important step toward software engineering.

---

25. `elif` Is Not a Loop

A common beginner misunderstanding is thinking that `elif` repeatedly checks conditions.

It doesn't.

An `if` / `elif` / `else` structure is a **decision structure**.

It generally runs through the chain once.

Example:

```
number = 10

if number > 5:
    print("Greater than 5")
elif number > 8:
    print("Greater than 8")
```

Output:

```
Greater than 5
```

Even though `10 > 8` is also true, Python never reaches that condition.

---

26. When Multiple `if` Statements Are Correct

`elif` should not replace every `if`.

Sometimes several actions should happen independently.

Example:

```
age = 25
has_account = True
is_verified = True

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

if has_account:
    print("Account exists")

if is_verified:
    print("Verified user")
```

All three statements can be printed.

These are independent facts.

Using `elif` here would be incorrect because the conditions are not mutually exclusive.

---

27. Independent Decisions vs One Decision

This distinction is extremely important.

Use separate `if` statements when:

Multiple conditions may be true at the same time.

```
if has_email:
    print("Email available")

if has_phone:
    print("Phone available")
```

Use `if` + `elif` when:

You want one outcome from several alternatives.

```
if payment == "cash":
    print("Cash payment")
elif payment == "card":
    print("Card payment")
elif payment == "bank":
    print("Bank transfer")
```

Ask yourself:

> **Can more than one branch logically happen?**

If yes, separate `if` statements may be appropriate.

If no, an `if` / `elif` chain is often appropriate.

---

28. Nested `if` and `elif`

An `elif` can also exist inside another `if`.

```
age = 20
has_account = True

if has_account:
    if age < 18:
        print("Minor account")
    elif age < 60:
```

```
        print("Adult account")
    else:
        print("Senior account")
else:
    print("Create an account first")
```

Here we have two levels of decisions.

Outer decision:

```
Does the account exist?
```

Inner decision:

```
What age category is the user?
```

Nested conditions are useful, but too much nesting can make code difficult to understand.

---

29. Avoid Unnecessary Nesting

Instead of:

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

sometimes you can write:

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

This is often easier to read.

The goal is not to avoid nesting completely.

The goal is to use the simplest structure that clearly represents the business rule.

---

30. Common Mistake: Forgetting the Colon

Incorrect:

```
if marks >= 90
    print("A")
```

Correct:

```
if marks >= 90:
    print("A")
```

The colon tells Python that a block of code follows.

The same applies to `elif`:

```
elif marks >= 80:
    print("B")
```

---

31. Common Mistake: Wrong Indentation

Incorrect:

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

Correct:

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

Python uses indentation as part of its syntax.

Indentation isn't merely visual formatting.

It communicates program structure.

---

32. Common Mistake: Writing `else if`

Python does not use:

```
else if
```

Instead, Python uses:

```
elif
```

Correct:

```
if score >= 90:
    print("A")
elif score >= 80:
    print("B")
```

---

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

Incorrect:

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

Correct:

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

Remember:

```
=   assignment
==  comparison
```

---

34. Common Mistake: Putting Conditions in the Wrong Order

Incorrect:

```
score = 95

if score >= 50:
    print("Pass")
elif score >= 90:
    print("Excellent")
```

Output:

```
Pass
```

The `>= 90` condition is unreachable for values that already satisfy `>= 50`.

Better:

```
if score >= 90:
    print("Excellent")
elif score >= 50:
    print("Pass")
else:
    print("Fail")
```

---

35. Common Mistake: Overcomplicated Conditions

This:

```
if age >= 18 and age < 60:
    print("Adult")
```

is perfectly valid.

But if you're using an ordered `elif` chain:

```
if age < 13:
    print("Child")
elif age < 18:
    print("Teenager")
elif age < 60:
    print("Adult")
else:
    print("Senior")
```

you don't need to repeat lower boundaries.

The previous conditions already establish them.

---

36. Common Mistake: Forgetting the Invalid Case

Suppose you build a grading system:

```
if marks >= 90:
    print("A")
elif marks >= 80:
    print("B")
elif marks >= 70:
    print("C")
else:
    print("Below C")
```

What happens if the user enters:

```
150
```

The program says:

```
A
```

Technically, the condition is true.

But the input itself is invalid.

A better system validates the input first:

```
if marks < 0 or marks > 100:
    print("Invalid marks")
elif marks >= 90:
    print("A")
elif marks >= 80:
```

```
    print("B")
elif marks >= 70:
    print("C")
else:
    print("Below C")
```

This introduces an important programming principle:

> **Don't assume input is valid.**

---

37. `elif` With Boolean Variables

Suppose:

```
is_admin = False
is_teacher = True
is_student = False
```

You can write:

```
if is_admin:
    print("Admin dashboard")
elif is_teacher:
    print("Teacher dashboard")
elif is_student:
    print("Student dashboard")
else:
    print("Guest dashboard")
```

Python doesn't require:

```
if is_admin == True:
```

You can simply write:

```
if is_admin:
```

---

38. Conditions Can Contain Function Calls

Later, as you learn functions, you'll see patterns such as:

```
if is_logged_in():
    print("Dashboard")
elif is_registered():
    print("Login required")
```

```
else:
    print("Register first")
```

The condition can come from a function that returns a Boolean.

This is one reason understanding `if` / `elif` early is important.

---

39. Multiple Conditions and Readability

There is a difference between code that works and code that is easy to understand.

Harder to read:

```
if age >= 18 and has_ticket and not is_banned and account_active:
    print("Allowed")
```

This may still be valid.

But if the business rule is important, you could make the logic clearer:

```
is_eligible = age >= 18
has_access = has_ticket and account_active
is_blocked = is_banned

if is_blocked:
    print("Access denied")
elif is_eligible and has_access:
    print("Access allowed")
else:
    print("Access requirements not met")
```

Good code makes decisions understandable.

---

40. Decision Tables

Before writing complicated conditions, you can create a decision table.

Suppose an online store gives discounts:

Customer	**Order Amount**	**Result**
New	Any	10%
Premium	100+	20%

Premium	Below 100	15%
Regular	Any	0%

You can translate this into code.

```
if is_new:
    discount = 10
elif is_premium and order_amount >= 100:
    discount = 20
elif is_premium:
    discount = 15
else:
    discount = 0
```

Notice how the table helps determine the order.

---

41. Real-World Example: Delivery Charges

Imagine a delivery system:

```
Order >= 5000 → Free delivery
Order >= 3000 → 100 delivery fee
Order >= 1000 → 200 delivery fee
Below 1000 → 300 delivery fee
```

Python:

```
order_amount = 3500

if order_amount >= 5000:
    delivery_fee = 0
elif order_amount >= 3000:
    delivery_fee = 100
elif order_amount >= 1000:
    delivery_fee = 200
else:
    delivery_fee = 300

print(delivery_fee)
```

Output:

```
100
```

This is a real business rule represented as code.

---

42. Real-World Example: Login System

A simplified login decision could be:

```
is_logged_in = False
account_exists = True

if is_logged_in:
    print("Open dashboard")
elif account_exists:
    print("Show login page")
else:
    print("Show registration page")
```

This represents a user journey.

The same decision-making concept appears in web applications.

---

43. Real-World Example: Mobile App Status

Suppose an app receives a network status:

```
status = "offline"

if status == "online":
    print("Load data")
elif status == "offline":
    print("Show offline message")
elif status == "connecting":
    print("Show loading indicator")
else:
    print("Unknown network status")
```

This kind of state-based logic appears frequently in application development.

---

44. Real-World Example: Resource Access

Imagine a learning platform has subscription levels:

```
plan = "premium"
```

```
if plan == "free":
    print("Access free resources")
elif plan == "student":
    print("Access student resources")
elif plan == "premium":
    print("Access premium resources")
else:
    print("Unknown subscription")
```

The same basic structure can eventually become much more sophisticated when connected to databases, APIs, authentication systems, and application interfaces.

---

45. A Complete Beginner Project: Grade Calculator

Let's build a small project from scratch.

Requirements

The program should:

1. Ask for marks.
2. Reject marks below 0.
3. Reject marks above 100.
4. Assign a grade.
5. Display the result.

Step 1: Get input

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

Step 2: Validate

```
if marks < 0 or marks > 100:
    print("Invalid marks")
```

Step 3: Add grade conditions

```
elif marks >= 90:
    print("Grade A")
elif marks >= 80:
    print("Grade B")
elif marks >= 70:
    print("Grade C")
elif marks >= 60:
    print("Grade D")
```

```
else:
    print("Grade F")
```

Complete program

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

if marks < 0 or marks > 100:
    print("Invalid marks")
elif marks >= 90:
    print("Grade A")
elif marks >= 80:
    print("Grade B")
elif marks >= 70:
    print("Grade C")
elif marks >= 60:
    print("Grade D")
else:
    print("Grade F")
```

---

46. Improve the Grade Calculator

Now make it more useful.

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

if marks < 0 or marks > 100:
    print("Invalid marks")
elif marks >= 90:
    print("Grade A - Excellent")
elif marks >= 80:
    print("Grade B - Very Good")
elif marks >= 70:
    print("Grade C - Good")
elif marks >= 60:
    print("Grade D - Needs Improvement")
else:
    print("Grade F - Fail")
```

Now the program communicates more information.

---

47. Another Mini Project: Simple Electricity Bill

Imagine:

```
0–100 units   → 10 per unit
101–200       → 15 per unit
201–300       → 20 per unit
301+          → 25 per unit
```

You could begin:

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

if units <= 100:
    print("Rate: 10")
elif units <= 200:
    print("Rate: 15")
elif units <= 300:
    print("Rate: 20")
else:
    print("Rate: 25")
```

The important lesson isn't the electricity bill itself.

It's learning how to convert a range-based requirement into an ordered decision structure.

---

48. `elif` and Program Design

When writing conditions, don't immediately start typing.

First ask:

What are the possible states?

Example:

```
Order:
- pending
- confirmed
- shipped
- delivered
- cancelled
```

Then ask:

What should happen in each state?

Finally, convert those rules into code.

```
if status == "pending":
    print("Waiting for confirmation")
elif status == "confirmed":
    print("Order confirmed")
elif status == "shipped":
    print("Order is on the way")
elif status == "delivered":
    print("Order delivered")
elif status == "cancelled":
    print("Order cancelled")
else:
    print("Unknown status")
```

This approach scales much better than randomly adding conditions.

---

49. Conditions Should Represent Requirements

Suppose someone says:

> "Premium users with an active subscription can access this resource."

Translate the sentence into logic.

There are three concepts:

```
Premium user
AND
Active subscription
```

Python:

```
if is_premium and subscription_active:
    print("Access granted")
else:
    print("Access denied")
```

The programming skill is not memorizing `elif`.

The deeper skill is:

> **Turning human requirements into logical conditions.**

---

50. From Requirements to Code

Consider this requirement:

> "If the user's marks are 90 or above, show A. If they are between 80 and 89, show B. If they are between 70 and 79, show C. Otherwise show F."

Break it down:

```
90+ → A
80+ → B
70+ → C
Otherwise → F
```

Then write:

```
if marks >= 90:
    grade = "A"
elif marks >= 80:
    grade = "B"
elif marks >= 70:
    grade = "C"
else:
    grade = "F"
```

The code is simply the final form of the requirement.

---

51. `elif` Is About Choosing a Path

Think of your program as a road.

At a decision point:

```
                ┌── Condition A → Path A
Input → Decision
                ├── Condition B → Path B
                ├── Condition C → Path C
                └── Otherwise → Path D
```

Only one path is selected from the chain.

This is why `elif` is so useful for:

- grades
- menus
- user roles
- subscription plans
- order states
- payment methods

- age categories
- pricing tiers
- application states
- validation results

---

52. Mini Quiz

Question 1

What will this print?

```
x = 15

if x < 10:
    print("A")
elif x < 20:
    print("B")
else:
    print("C")
```

Answer:

```
B
```

Question 2

What is wrong with this?

```
if marks >= 50:
    print("Pass")
elif marks >= 90:
    print("Excellent")
```

The `marks >= 90` condition will never be reached for values 90 or above because those values already satisfy `marks >= 50`.

A better order is:

```
if marks >= 90:
    print("Excellent")
elif marks >= 50:
    print("Pass")
```

Question 3

When should you use separate `if` statements instead of `elif`?

When multiple conditions can independently be true and multiple actions may need to happen.

---

53. Practice Exercises

Exercise 1: Age Category

Ask the user for age.

Print:

```
Child
Teenager
Adult
Senior
```

Use `elif`.

Exercise 2: Number Category

Ask for a number.

Print:

```
Positive
Negative
Zero
```

Exercise 3: Temperature

Ask for temperature.

Create categories such as:

```
Cold
Cool
Warm
Hot
```

Exercise 4: Login Status

Create:

```
is_logged_in
has_account
```

Decide whether the user should see:

```
Dashboard
Login page
Registration page
```

Exercise 5: Shopping Discount

Create rules:

```
5000+ → 20%
3000+ → 15%
1000+ → 10%
Below 1000 → 0%
```

Write the conditions in the correct order.

---

54. Five-Minute Challenge

Build a **Simple Ticket Pricing System**.

Ask the user for their age.

Rules:

```
Below 5       → Free
5–12          → Child ticket
13–17         → Teen ticket
18–59         → Adult ticket
60+           → Senior ticket
```

Your program should:

1. Ask for age.
2. Use `if`, `elif`, and `else`.
3. Print the correct category.
4. Handle invalid negative ages.

Example:

```
Enter your age: 16

Teen ticket
```

---

55. Debugging Challenge

Find the problem:

```
marks = 85

if marks >= 60:
    print("D")
elif marks >= 70:
    print("C")
elif marks >= 80:
    print("B")
elif marks >= 90:
    print("A")
else:
    print("F")
```

Think before changing the code.

Ask:

> Which condition becomes true first for `85`?

Answer:

```
marks >= 60
```

So Python prints:

```
D
```

The fix is to reorder the conditions from the highest threshold to the lowest.

---

56. Interview Questions

1. What is `elif` in Python?

`elif` allows a program to test another condition when the previous `if` or `elif` conditions were false.

2. Can a Python program have multiple `elif` blocks?

Yes.

```
if condition1:
    ...
elif condition2:
    ...
elif condition3:
    ...
```

3. Is `else` required?

No.

You can have:

```
if condition:
    print("Yes")
```

or:

```
if condition:
    print("A")
elif another_condition:
    print("B")
```

without an `else`.

4. Can there be multiple `else` blocks?

Not within the same `if` chain.

One `if` / `elif` chain can have at most one `else`.

5. Does Python check every `elif`?

No.

Once Python finds the first true condition, it executes that branch and skips the remaining branches.

6. What is the difference between `if` and `elif`?

Separate `if` statements are independent. `elif` belongs to the same decision chain and is checked only if previous conditions were false.

---

57. Cheat Sheet

```
if condition:
    # first choice
elif condition:
    # second choice
elif condition:
    # third choice
else:
    # fallback
```

Comparison

```
==
!=
>
<
>=
<=
```

Logical operators

```
and
or
not
```

Example

```
score = 85

if score >= 90:
    print("A")
elif score >= 80:
    print("B")
elif score >= 70:
    print("C")
else:
    print("F")
```

Remember

```
if      → first condition
elif    → another condition
else    → fallback
```

---

58. Key Takeaways

After learning `elif` and multiple conditions, you should understand:

- `elif` means "else if".
- It allows Python to handle multiple possible outcomes.
- Conditions are checked from top to bottom.
- The first true branch is executed.
- Remaining `elif` conditions are skipped.
- `else` handles the case where no previous condition is true.

- Condition order matters.
- Separate `if` statements are appropriate when multiple conditions can independently be true.
- `and` requires multiple conditions to be true.
- `or` requires at least one condition to be true.
- Parentheses can make complex logic easier to understand.
- Input should be validated before normal processing.
- Good conditional code represents clear business rules.
- The deeper skill is translating real-world requirements into logical decisions.

---

59. Final Practice Project

Build a **Student Performance Analyzer**.

The program should ask for:

```
Student name
Marks
Attendance percentage
```

Then apply rules.

For example:

```
Invalid marks → Invalid input

90+            → Excellent
80–89          → Very Good
70–79          → Good
60–69          → Average
Below 60       → Needs Improvement
```

Then add an attendance rule:

```
Attendance below 75% → Attendance warning
```

Your program should demonstrate that some decisions are alternatives while others are independent.

For example:

```
if marks < 0 or marks > 100:
    print("Invalid marks")
elif marks >= 90:
```

```
    print("Excellent")
elif marks >= 80:
    print("Very Good")
elif marks >= 70:
    print("Good")
elif marks >= 60:
    print("Average")
else:
    print("Needs Improvement")

if attendance < 75:
    print("Attendance warning")
```

Notice the design.

The grade is **one choice from several alternatives**, so it uses `if` + `elif` + `else`.

Attendance is an **independent condition**, so it uses another `if`.

That distinction is more important than memorizing the syntax.

---

60. The Engineering Mindset

Beginners often think:

> "I need to learn how to write `elif`."

A developer eventually starts thinking:

> "What decisions does this program need to make?"

That shift matters.

Before writing:

```
if
elif
else
```

identify:

1. What information do I have?
2. What possible states exist?
3. Which states are mutually exclusive?
4. Can multiple conditions be true at the same time?
5. What should happen for each state?
6. What should happen when the input is invalid?

7. Which condition should be checked first?

Then write the code.

The syntax is small.

The ability to model decisions correctly is the real programming skill.

---

3 Questions Before You Touch the Keyboard

1. What are all the possible outcomes?
2. Are these outcomes alternatives or independent conditions?
3. What should happen when none of the expected conditions is true?

Visit haas.dev for more resources and guides.

haas.dev

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