Introduction to Python Programming

Chapter 5: If Statements and Conditions

Introduction

Now we're getting to the exciting part! If statements allow your programs to make decisions and respond differently based on conditions. This is where programs start to feel "smart" and dynamic.

---

1. Understanding Code Blocks

Before we dive into if statements, we need to understand **code blocks** - arguably one of the most important concepts in Python.

What is a Code Block?

A **code block** is a chunk of code that belongs together. In Python, code blocks are created using **indentation** (white space at the beginning of lines).

```
if age > 17:
    print("You are an adult")          # This line is IN the code block
    print("You can vote")              # This line is IN the code block
print("Thank you")                     # This line is OUTSIDE the code block
```

The Anatomy of an If Statement

```
if condition:                 # If line ends with a colon (:)
    # Code block starts       # Everything indented is INSIDE the block
    statement1
    statement2
    # Code block ends
statement3                    # Not indented = OUTSIDE the block
```

How it works:

1. Python checks if the condition is True
2. If True, it runs ALL the indented code (the code block)
3. After the code block, it continues with the next un-indented line
4. If False, it skips the entire code block and goes straight to the next un-indented line

Indentation Creates the Code Block

```python
temperature = 75

if temperature > 70:
    print("It's warm outside")       # IN the code block
    print("Wear light clothing")     # IN the code block
print("Have a nice day")             # OUTSIDE - always runs
```

If temperature is 75:

```
It's warm outside
Wear light clothing
Have a nice day
```

If temperature is 60:

```
Have a nice day
```

Code Blocks Are Everywhere!

Important: Code blocks aren't just for if statements! You'll see them with:

- **If statements** (conditions)
- **While loops** (repetition)
- **For loops** (iteration)
- **Functions** (later in the course)
- **Classes** (later in the course)

Understanding code blocks now will help you throughout your entire programming journey.

White Space Is CRITICAL

The **exact number of spaces matters**. Python needs consistency.

```python
# This works - consistent indentation
if age > 17:
    print("Adult")
    print("Can vote")

# This BREAKS - inconsistent indentation
if age > 17:
    print("Adult")
      print("Can vote")    # ERROR! Different indentation
```

Real-world problem: If you copy code from someone who uses 3 spaces and you use 4 spaces, mixing them will cause errors!

VS Code helps: It usually converts tabs to 4 spaces automatically, keeping things consistent.

Nested Code Blocks

Code blocks can contain other code blocks! Each level adds more indentation:

```
if temperature > 70:                   # Level 0 (no indent)
    print("It's warm")                 # Level 1 (one indent)
    if temperature > 90:               # Level 1
        print("It's very hot!")        # Level 2 (two indents)
        print("Stay hydrated")         # Level 2
    print("Enjoy the weather")         # Level 1
print("Goodbye")                       # Level 0
```

The pattern: Each nested if statement creates a deeper code block with one more level of indentation.

We'll explore nested if statements more later in this chapter.

2. Comparison Operators

To make decisions, we need to compare values. Python provides several comparison operators:

Operator	Meaning	Example	Result
==	Equal to	5 == 5	True
!=	Not equal to	5 != 3	True
>	Greater than	7 > 3	True
<	Less than	3 < 7	True
>=	Greater or equal	5 >= 5	True
<=	Less or equal	4 <= 4	True

Examples

```
age = 25

# Equal to
if age == 25:
    print("You are 25")
```

```python
# Not equal to
if age != 30:
    print("You are not 30")

# Greater than
if age > 18:
    print("You are older than 18")

# Less than or equal
if age <= 30:
    print("You are 30 or younger")
```

Common Mistake: = vs ==

```python
# WRONG - This is assignment, not comparison
if age = 25:     # ERROR!
    print("Hello")

# CORRECT - This is comparison
if age == 25:
    print("Hello")
```

Remember:

- Single equals = assigns a value
- Double equals == compares values

3. Basic If Statements

The simplest if statement checks one condition:

```python
age = 20

if age >= 18:
    print("You are an adult")
    print("You can vote")
print("Thank you")  # Always runs
```

How it works:

1. Is age >= 18 true? YES (20 >= 18)
2. Run the code block (both print statements)
3. Continue to next line (outside code block)

If the condition is False:

```
age = 16

if age >= 18:
    print("You are an adult")      # SKIPPED
    print("You can vote")           # SKIPPED
print("Thank you")                  # RUNS (not in code block)
```

Output: `Thank you`

---

4. If-Else Statements

Often you want to do one thing if a condition is True, and something else if it's False. Use `else`:

```
age = 16

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

How it works:

- If condition is True → run the if block
- If condition is False → run the else block
- One or the other will ALWAYS run (never both)

Another Example

```
temperature = 65

if temperature > 70:
    print("It's warm")
    print("Wear shorts")
else:
    print("It's cool")
    print("Wear a jacket")
```

If temperature is 80: Runs the if block
If temperature is 65: Runs the else block

---

5. If-Elif-Else Statements

What if you have more than two options? Use `elif` (else if):

```python
age = 70

if age >= 65:
    print("You are a senior")
elif age >= 18:
    print("You are an adult")
else:
    print("You are a child")
```

How it works:

1. Check first condition (age >= 65) → TRUE → run that block and STOP
2. (If first was false) Check elif condition (age >= 18)
3. (If both were false) Run else block

Important: Only ONE block runs! Once a condition is True, the rest are skipped.

Comparing Multiple Separate If Statements

Consider this approach:

```python
age = 25

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

if age >= 65:
    print("You are a senior")

if age < 18:
    print("You are a child")
```

Problem: Python checks EVERY if statement independently. If age is 25:

- First if: True → prints "You are an adult"
- Second if: False → skips
- Third if: False → skips

Now with elif:

```python
age = 25

if age >= 65:
    print("You are a senior")
elif age >= 18:
    print("You are an adult")
else:
    print("You are a child")
```

Better: Once it finds age >= 18 is True, it stops checking. More efficient and clearer logic!

Order Matters with Elif!

```python
# WRONG - Order matters!
age = 25

if age >= 18:
    print("Adult")        # This runs at 25
elif age >= 65:
    print("Senior")       # Never reached! 65 is also >= 18
```

```python
# CORRECT - Check more specific conditions first
age = 70

if age >= 65:
    print("Senior")       # Checks this first
elif age >= 18:
    print("Adult")
else:
    print("Child")
```

Example: Three Age Categories

```python
age = 25

if age >= 65:
    print("You are a senior")
    print("Senior discount applies")
elif age >= 18:
    print("You are an adult")
    print("Standard pricing")
else:
    print("You are a child")
    print("Child discount applies")
```

This clearly handles three distinct categories with no overlap.

---

6. Compound Conditions (Logical Operators)

Sometimes you need to check multiple conditions at once. Python provides three logical operators:

The AND Operator

Both conditions must be True for the result to be True.

```
temperature = 80
has_money = True

if temperature > 70 and has_money:
    print("You can buy ice cream!")
```

Truth Table for AND:

Condition 1	Condition 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

Real-world example:
"If it's warm outside AND you have money, you can have ice cream"

- Need BOTH to be true

```
temperature = 80
has_money = False

if temperature > 70 and has_money:
    print("Buy ice cream")    # DOESN'T RUN - need both true
else:
    print("No ice cream today")
```

The OR Operator

At least ONE condition must be True for the result to be True.

```
weather = "rainy"
temperature = 95

if weather == "rainy" or temperature > 90:
    print("Bring an umbrella")
```

Truth Table for OR:

Condition 1	Condition 2	Result
True	True	True
True	False	True
False	True	True
False	False	False

Real-world example:
"If it's sunny and hot OR it's raining, bring your umbrella"

- Can use umbrella for shade (hot) OR for rain
- Notice in English we might say "bring it for sun AND rain" but logically it's OR - you bring it if EITHER condition is true

```
weather = "sunny"
temperature = 95

# Bring umbrella if hot OR rainy
if temperature > 90 or weather == "rainy":
    print("Bring umbrella")    # RUNS - one condition is true
```

The NOT Operator

Reverses the condition - makes True into False and False into True.

```
is_raining = False

if not is_raining:
    print("You don't need an umbrella")
```

Truth Table for NOT:

Condition	Result
True	False
False	True

Combining Logical Operators

You can combine multiple operators:

```
age = 25
has_license = True
has_car = False

if age >= 18 and has_license and has_car:
    print("You can drive your own car")
elif age >= 18 and has_license:
    print("You can drive, but need to borrow a car")
else:
    print("You cannot drive")
```

Order of Evaluation

When combining operators, Python evaluates in this order:

1. Comparisons (==, >, <, etc.)
2. `not`
3. `and`
4. `or`

Use parentheses to be explicit:

```
# Without parentheses - might be confusing
if age >= 18 and has_license or has_permit:
    print("Can drive")

# With parentheses - crystal clear
if (age >= 18 and has_license) or has_permit:
    print("Can drive")
```

7. Nested If Statements

If statements can be inside other if statements! This is called **nesting**.

Basic Nested If

```python
age = 25
has_license = True

if age >= 18:
    print("You are an adult")
    if has_license:
        print("You can drive")
    else:
        print("You need a license to drive")
else:
    print("You are too young to drive")
```

How it works:

1. First checks age >= 18
2. If True, enters that code block
3. Inside, checks has_license
4. Each level of nesting adds one indentation

More Complex Nesting

```python
temperature = 85
has_money = True
store_open = True

if temperature > 70:
    print("It's warm outside")
    if has_money:
        print("You have money")
        if store_open:
            print("You can buy ice cream!")
        else:
            print("Store is closed, sorry")
    else:
        print("You need money for ice cream")
else:
    print("Too cold for ice cream")
```

When to Use Nesting

Nested if statements are powerful but can get complicated quickly. Use them when:

- You need to check a second condition ONLY if the first is true

- The logic naturally has levels of decision-making

Too much nesting makes code hard to read. Later in the course, we'll learn techniques to simplify complex logic.

The Power of Combination

This is where programming becomes powerful! By combining simple if statements in different ways, you can create complex decision-making logic.

Think of it like building with blocks:

- Each if statement is simple to understand
- But combining them creates sophisticated programs
- The **algorithm** (steps to solve a problem) is built by arranging these pieces

Getting familiar with nested if statements is critical - they appear everywhere in programming. Start simple, practice often, and gradually work up to more complex combinations.

---

8. Practical Examples

Example 1: Grade Calculator

```python
score = 85

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "F"

print(f"Your grade is: {grade}")
```

Example 2: Login System

```
username = input("Enter username: ")
password = input("Enter password: ")

if username == "admin" and password == "secret123":
    print("Login successful!")
    print("Welcome to the system")
else:
    print("Invalid username or password")
```

Example 3: Ticket Pricing

```
age = int(input("Enter your age: "))
is_student = input("Are you a student? (yes/no): ")

if age < 12:
    price = 5.00
    print("Child ticket")
elif age >= 65:
    price = 7.00
    print("Senior ticket")
elif is_student == "yes":
    price = 8.00
    print("Student ticket")
else:
    price = 12.00
    print("Adult ticket")

print(f"Ticket price: ${price}")
```

Key Takeaways

✓ **Code blocks are created by indentation** - consistent white space is critical

✓ **Code blocks appear everywhere** - if statements, loops, functions, classes

✓ **Comparison operators:** ==, !=, >, <, >=, <=

✓ **If-else gives two options**, if-elif-else gives multiple options

✓ **Only ONE block runs** in an if-elif-else chain

✓ **Order matters** with elif - check specific conditions first

✓ **Logical operators:** `and` (both must be true), `or` (at least one true), `not` (reverses)

✓ **Nested if statements** create complex decision logic from simple pieces

✓ **The power is in combination** - simple pieces create sophisticated programs

Reflection Questions

1. What creates a code block in Python?
2. What's the difference between = and ==?
3. In an if-elif-else structure, how many blocks can run?
4. What does the `and` operator require to return True?
5. What does the `or` operator require to return True?
6. Why does order matter when using elif statements?
7. When should you use nested if statements?

Practice Exercises

1. **Age Checker:** Write a program that asks for age and prints whether someone is a child (<13), teenager (13-19), adult (20-64), or senior (65+)
2. **Password Strength:** Ask for a password and check if it's longer than 8 characters AND contains a number. Print "Strong" or "Weak"
3. **Temperature Advisor:** Ask for temperature and whether it's raining. Give clothing advice based on conditions:
    - Hot (>75) and not raining: "Wear shorts"
    - Hot and raining: "Wear light rain jacket"
    - Cold (<=75) and raining: "Wear warm rain jacket"
    - Cold and not raining: "Wear a sweater"
4. **Movie Ticket:** Ask for age and whether they're a student. Calculate ticket price:
    - Under 12: $8
    - 12-64 and student: $10
    - 12-64 and not student: $15
    - 65+: $10

Appendix: Match Statements (Optional)

Python 3.10 introduced **match statements** as an alternative to if-elif-else chains. While we won't use these much in this course, you should know they exist.

Basic Match Statement

```python
day = "Monday"

match day:
    case "Monday":
        print("Start of work week")
    case "Friday":
        print("End of work week")
    case "Saturday" | "Sunday":
        print("Weekend!")
    case _:
        print("Midweek day")
```

This is equivalent to:

```python
if day == "Monday":
    print("Start of work week")
elif day == "Friday":
    print("End of work week")
elif day == "Saturday" or day == "Sunday":
    print("Weekend!")
else:
    print("Midweek day")
```

Why Match Exists

Match statements can be cleaner when you're checking one variable against many specific values. They're also more powerful for pattern matching (advanced topic).

Why We're Not Using Them Much

1. If-elif-else is more flexible
2. Match only works with Python 3.10+
3. Most Python code uses if-elif-else
4. If statements can handle more complex conditions

You're welcome to use match statements if you prefer, but all examples in this course will use if-elif-else since it's more universal and flexible.