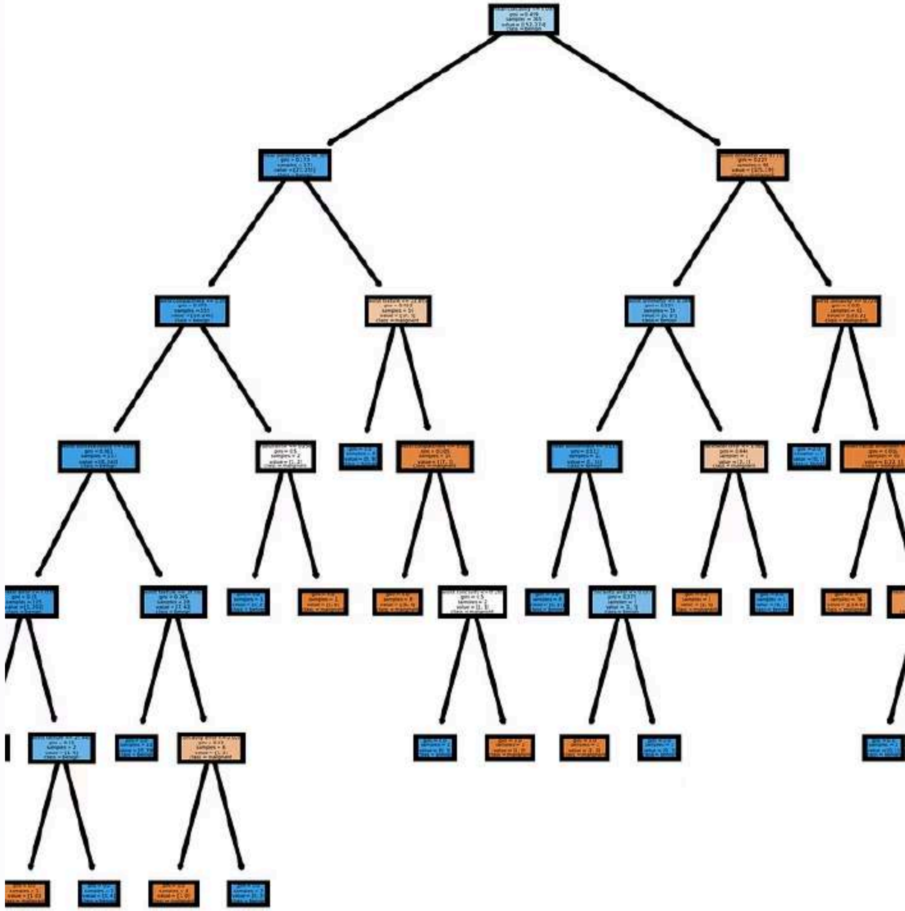
Python Decision Making – Day 2

Master the art of program control flow through conditions and Boolean logic



COURSE OVERVIEW

What We'll Learn Today

01

Boolean Logic

Understanding True and False values and comparison operators

02

Conditional Statements

Making decisions with if, else, and elif

03

Nested Conditions

Building complex decision structures

04

Real Applications

Solving practical programming challenges

CHAPTER 1

Boolean Logic Fundamentals

Understanding Boolean Values

In Python, Boolean values represent truth. There are only two possible Boolean values: `True` and `False`. These values are fundamental to decision making in programs.

Booleans are named after George Boole, a mathematician who pioneered Boolean algebra. In Python, note that True and False must be capitalized.

```
# Boolean values
is_raining = True
is_sunny = False

# Boolean in action
print(is_raining) # Output: True
print(type(is_raining)) #
```

Comparison Operators

== (Equal to)

Checks if two values are equal

```
5 == 5  # True
```

!= (Not equal)

Checks if values differ

```
5 != 3  # True
```

< > (Less/Greater)

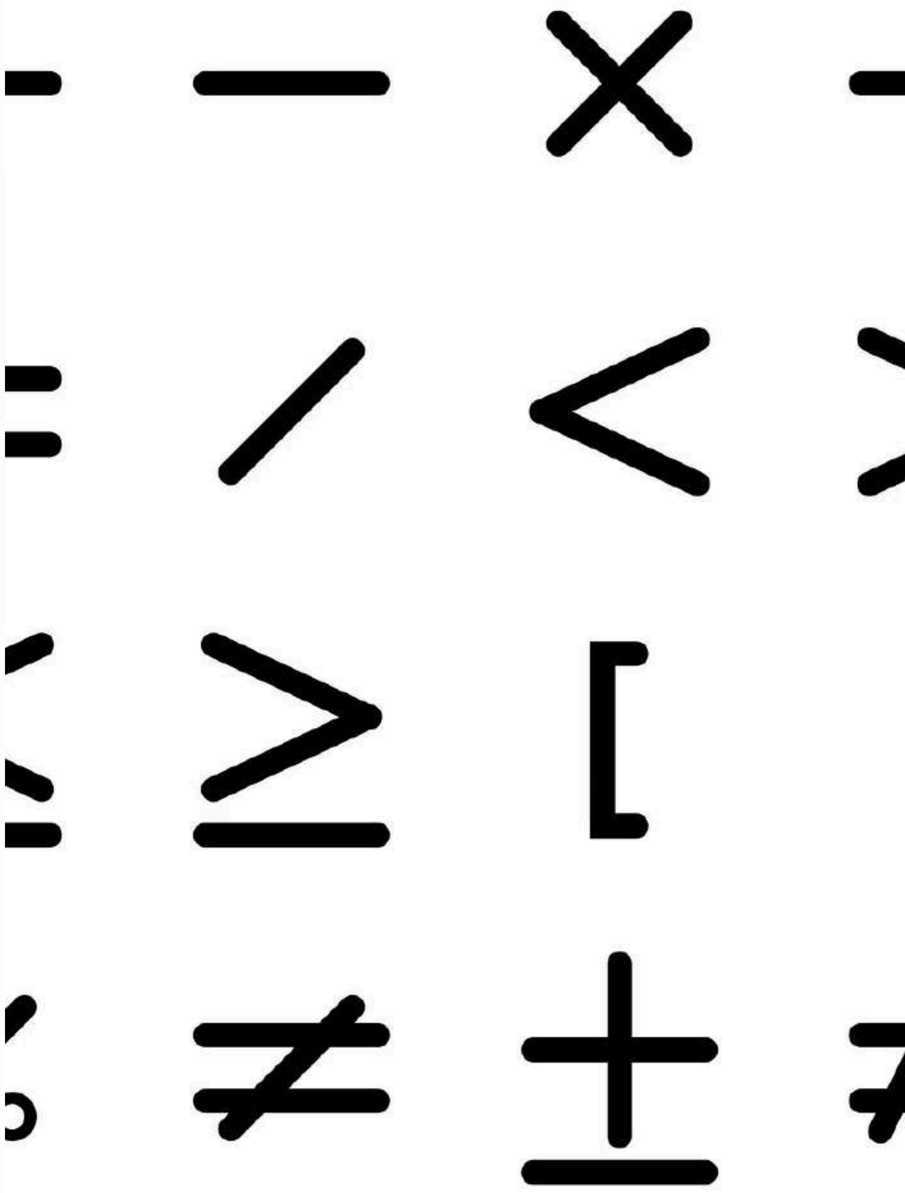
Compares numeric values

```
3 < 5  # True
```

<= >= (Less/Greater or Equal)

Includes equality check

```
5 <= 5 # True
```



Comparison Examples in Practice

Numeric Comparisons

```
age = 25
print(age > 18)     # True
print(age == 30)    # False
print(age <= 25)    # True
print(age != 20)    # True
```

String Comparisons

```
name = "Python"
print(name == "Python")   # True
print(name != "Java")     # True
print("A" < "B")          # True
print("apple" > "zebra")  # False
```

Strings are compared lexicographically (alphabetically), where uppercase letters come before lowercase in ASCII order.

Logical Operators

and

Returns True only if both conditions are True

```
True and True   # True
True and False  # False
```

or

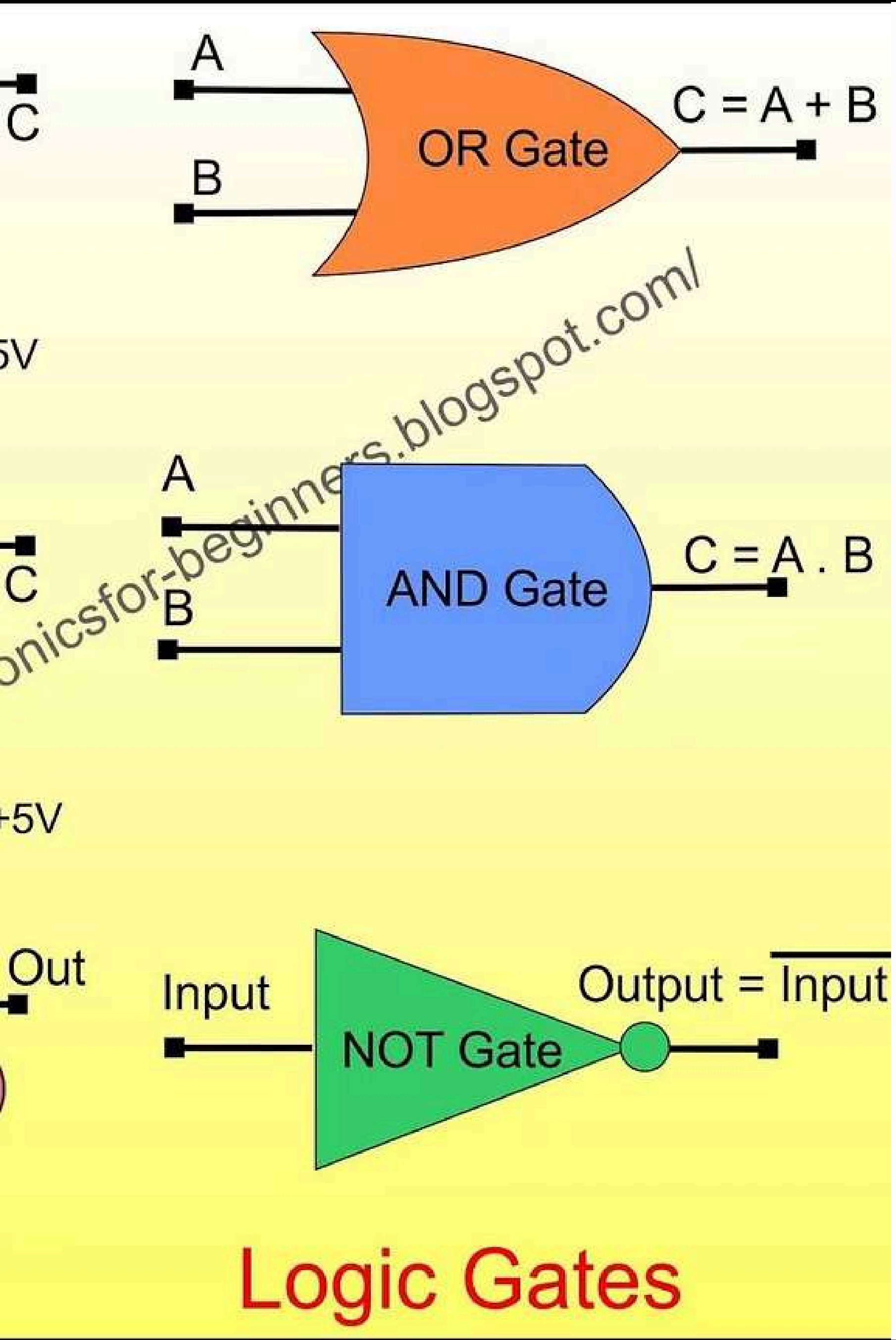
Returns True if at least one condition is True

```
True or False  # True
False or False # False
```

not

Inverts the Boolean value

```
not True # False
not False # True
```

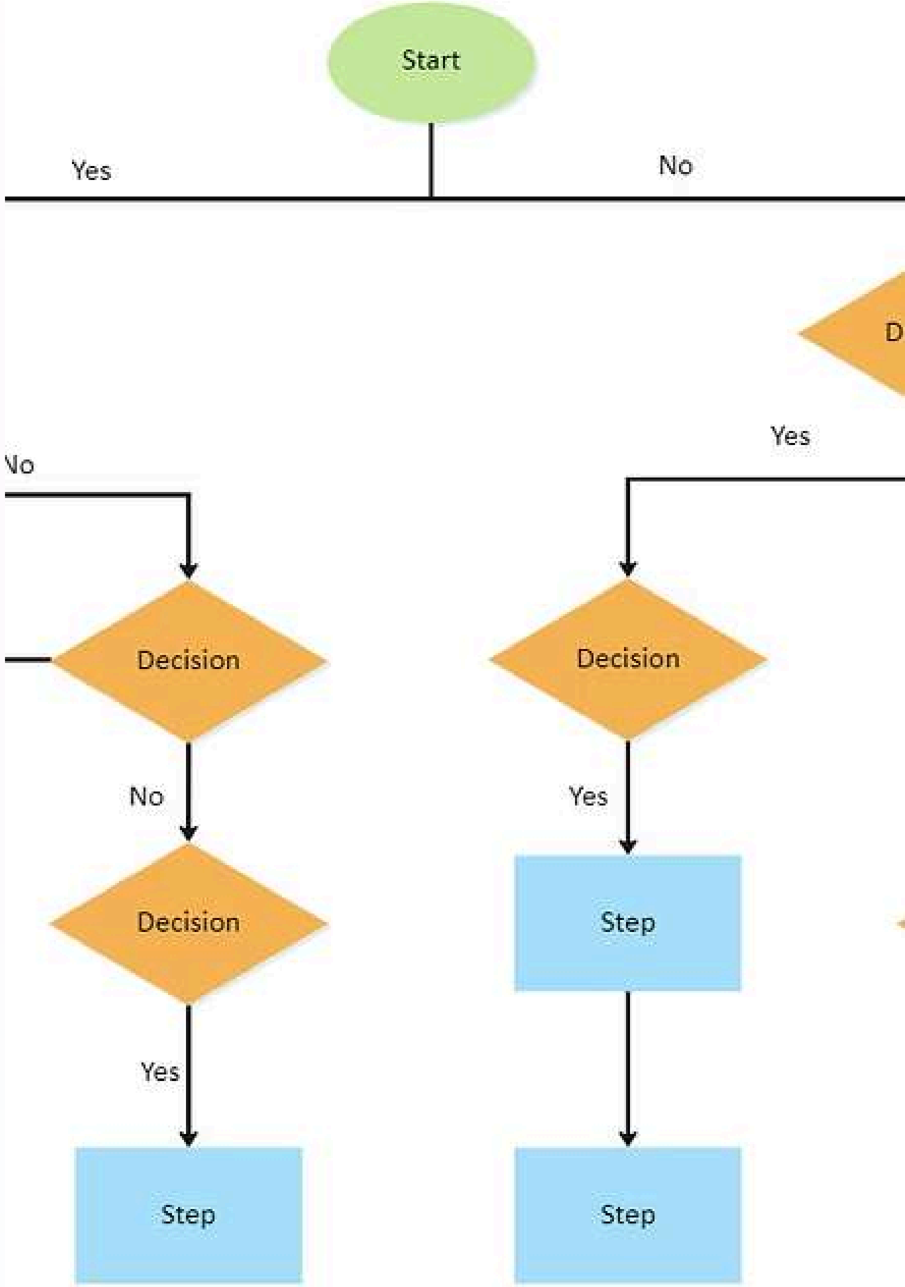


Combining Logical Operators

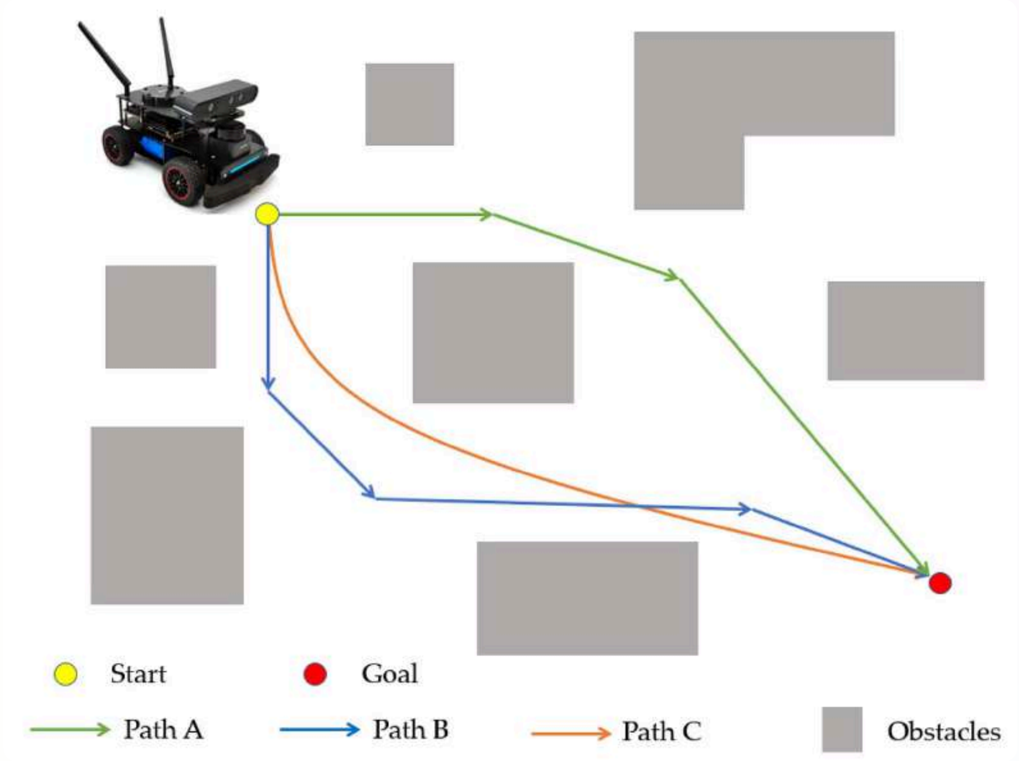
```
age = 25
has_license = True
is_insured = True

# Complex conditions using logical operators
can_drive = age >= 18 and has_license and is_insured
print(can_drive) # True

# Multiple OR conditions
is_weekend = True
is_holiday = False
can_sleep_in = is_weekend or is_holiday
print(can_sleep_in) # True

# Combining AND, OR, and NOT
has_permission = (age > 21 or has_license) and not is_insured
print(has_permission) # False
```

CHAPTER 2

Decision Making in Programs



Why Programs Need Decisions

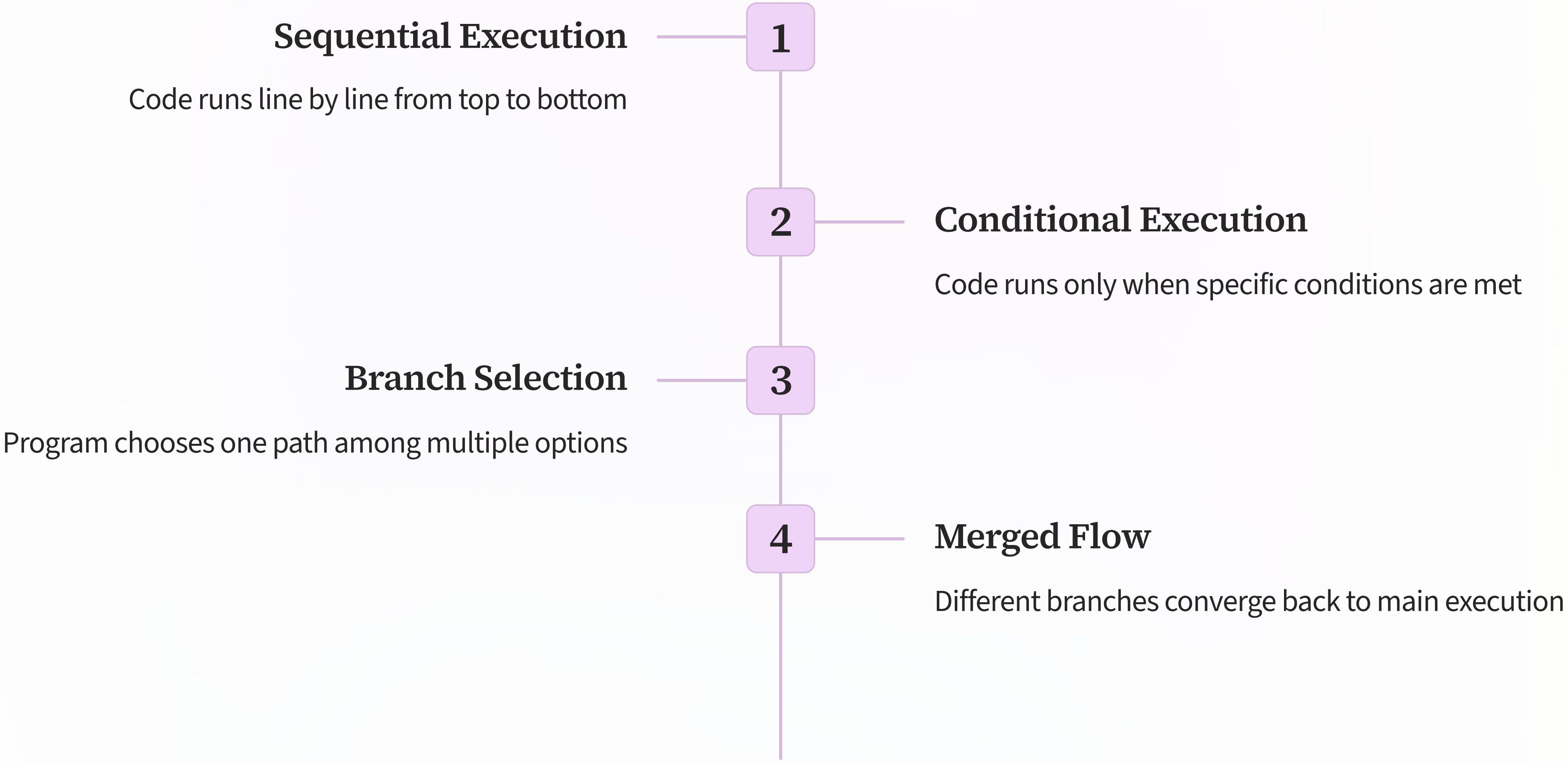


Programs need to respond differently based on varying conditions. Without decision-making capabilities, programs would execute the same instructions regardless of input or state.

Decision structures allow programs to:

- Validate user input
- Handle different scenarios
- Control program flow
- Implement business logic

Control Flow Concept



1

Sequential Execution

Code runs line by line from top to bottom

2

Conditional Execution

Code runs only when specific conditions are met

3

Branch Selection

Program chooses one path among multiple options

4

Merged Flow

Different branches converge back to main execution

CHAPTER 3

The if Statement

Basic if Statement Syntax

The if statement executes a block of code only when a condition evaluates to True. The syntax requires a colon after the condition and proper indentation for the code block.

Indentation in Python is not just style—it's syntax. The standard is 4 spaces.

```
# Basic if statement structure
if condition:
 # Code to execute if True
 statement1
 statement2

# Example
temperature = 30
if temperature > 25:
 print("It's hot outside!")
 print("Wear light clothes")
```

if Statement Examples

Age Verification

```
age = 20
if age >= 18:
    print("You can vote")
    print("Registration open")
```

Password Check

```
password = "secure123"
if len(password) >= 8:
    print("Strong password")
    print("Access granted")
```

Grade Evaluation

```
score = 85
if score >= 90:
 print("Excellent work!")
 print("Grade: A")
```

Common if Statement Patterns

```
# Pattern 1: Checking ranges
price = 150
if price > 100:
 discount = price * 0.1
 print(f"Discount applied: ${discount}")

# Pattern 2: String validation
username = "john_doe"
if len(username) > 5:
 print("Username accepted")

# Pattern 3: Boolean flag checking
is_logged_in = True
if is_logged_in:
 print("Welcome back!")

# Pattern 4: Multiple conditions
score = 85
passed = True
if score > 70 and passed:
 print("You passed the course!")
```

CHAPTER 4

if-else Statements

Adding else for Alternative Actions

The else clause provides an alternative path when the if condition is False. This creates a binary decision structure—one block executes if the condition is True, the other if it's False.

Only one block will ever execute in an if-else structure, never both.

```
# if-else syntax
if condition:
 # Executes when True
 action1()
else:
 # Executes when False
 action2()

# Example
age = 16
if age >= 18:
 print("You can drive")
else:
 print("Too young to drive")
```

if-else Practical Examples

Even or Odd Checker

```
number = 7
if number % 2 == 0:
print(f"{number} is even")
else:
print(f"{number} is odd")
```

Login Validation

```
password = "python123"
correct_password = "secure456"

if password == correct_password:
    print("Login successful")
else:
    print("Invalid password")
```

Temperature Warning

```
temperature = 15
if temperature > 20:
print("Wear light clothes")
else:
print("Wear warm clothes")
```

CHAPTER 5

if-elif-else Chains

Handling Multiple Conditions

When you need to check multiple mutually exclusive conditions, use elif (else if). Python evaluates conditions from top to bottom and executes only the first True block it encounters.

```
# if-elif-else structure
if condition1:
 # Runs if condition1 is True
 action1()
elif condition2:
 # Runs if condition2 is True
 action2()
elif condition3:
 # Runs if condition3 is True
 action3()
else:
 # Runs if all are False
 default_action()
```

Grade Calculator Example

```
score = 87

if score >= 90:
 grade = "A"
 print(f"Excellent! Your grade is {grade}")
elif score >= 80:
 grade = "B"
 print(f"Good work! Your grade is {grade}")
elif score >= 70:
 grade = "C"
 print(f"Satisfactory. Your grade is {grade}")
elif score >= 60:
 grade = "D"
 print(f"Needs improvement. Your grade is {grade}")
else:
 grade = "F"
 print(f"Failed. Your grade is {grade}")

print(f"Final grade: {grade}")
# Output: Good work! Your grade is B
# Output: Final grade: B
```

More elif Examples

1

Day of Week

```
day = 3
if day == 1:
    print("Monday")
elif day == 2:
    print("Tuesday")
elif day == 3:
    print("Wednesday")
else:
    print("Other day")
```

2

Ticket Pricing

```
age = 65
if age < 12:
    price = 5
elif age < 65:
    price = 10
else:
    price = 7
print(f"Ticket: ${price}")
```

3

BMI Category

```
bmi = 24.5
if bmi < 18.5:
 print("Underweight")
elif bmi < 25:
 print("Normal")
elif bmi < 30:
 print("Overweight")
else:
 print("Obese")
```

Important elif Rules

Order Matters

Conditions are evaluated top to bottom. Once one is True, remaining conditions are skipped. Place most specific conditions first.

Only One Block Executes

Even if multiple conditions are True, only the first True block runs. This is different from using multiple separate if statements.

else is Optional

You can use if-elif without an else clause. However, else provides a catch-all for unexpected cases.

CHAPTER 6

Nested Conditions

What Are Nested Conditions?

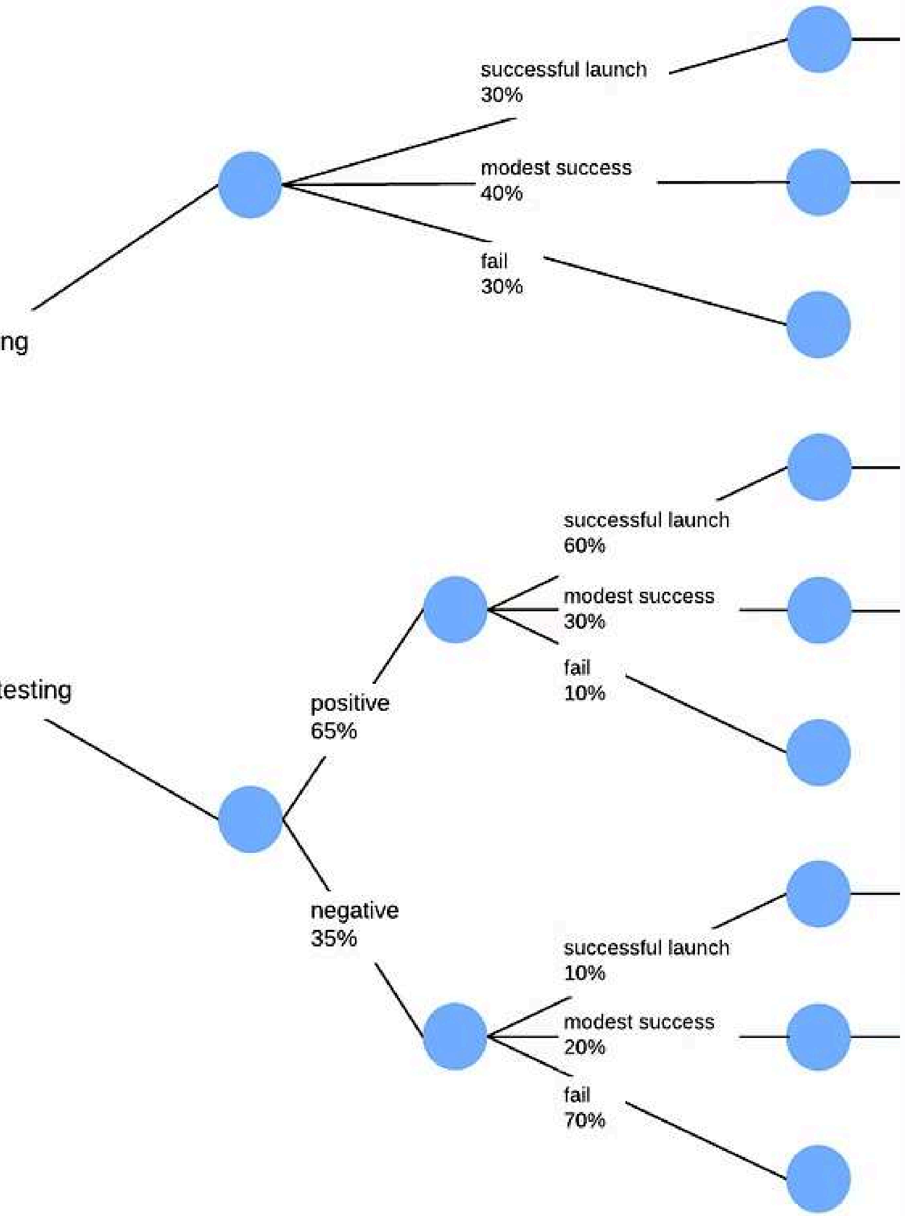
Nested conditions are if statements placed inside other if statements. They allow you to create complex decision trees where subsequent checks depend on earlier conditions being True.

Each level of nesting adds another layer of specificity to your program's logic, enabling sophisticated decision-making processes.

```
# Basic nested structure
if outer_condition:
 # Outer block
 if inner_condition:
 # Inner block
 action()
 else:
 # Inner else
 other_action()
```

Simple Nesting Example

```
age = 25
has_license = True

if age >= 18:
 print("You are old enough to drive")

 if has_license:
 print("You can drive legally")
 print("Enjoy the road!")
 else:
 print("You need to get a license first")
 print("Visit the DMV")
else:
 print("You are too young to drive")
 print("Wait until you turn 18")

# Output:
# You are old enough to drive
# You can drive legally
# Enjoy the road!
```



Multi-Level Nesting

```
score = 85
attendance = 95
assignment_complete = True

if score >= 60:
 print("You passed the exam")

 if attendance >= 80:
 print("Good attendance record")

 if assignment_complete:
 print("All requirements met")
 print("Grade: A")
 else:
 print("Complete your assignment")
 print("Grade: B")
 else:
 print("Attendance below requirement")
 print("Grade: C")
else:
 print("You failed the exam")
 print("Grade: F")
```

Practical Nested Conditions

Login System

```
username = "admin"
password = "secret"

if username == "admin":
    if password == "secret":
        print("Access granted")
    else:
        print("Wrong password")
else:
    print("User not found")
```

Discount Calculator

```
is_member = True
purchase_amount = 150

if is_member:
 if purchase_amount > 100:
 discount = 0.20
 else:
 discount = 0.10
else:
 if purchase_amount > 200:
 discount = 0.05
 else:
 discount = 0
```

Best Practices for Nesting

1

Limit Nesting Depth

Avoid nesting more than 3 levels deep. Excessive nesting makes code hard to read and maintain. Consider restructuring or using logical operators instead.

2

Use Proper Indentation

Maintain consistent 4-space indentation for each level. This visual structure is crucial for understanding the logic flow.

3

Consider Alternatives

Sometimes nested conditions can be replaced with logical operators (and, or) or elif chains for clearer code.

4

Add Comments

Document complex nested logic with comments explaining the decision criteria at each level.

CHAPTER 7

The pass Keyword

Understanding pass

The `pass` keyword is a null operation—when executed, nothing happens. It's used as a placeholder where Python syntax requires a statement but you don't want to execute any code.

Think of `pass` as a "to-do" marker or empty placeholder that prevents syntax errors while you develop your program.

```
# Error without pass
if age > 18:
 # SyntaxError: expected statement

# Correct with pass
if age > 18:
 pass # Will implement later

# pass does nothing
x = 10
pass
print(x) # Output: 10
```

When to Use pass

Future Implementation

Use pass as a placeholder when you're designing the structure but haven't written the implementation yet.

```
if user_type == "admin":
    pass  # TODO: Add admin logic
else:
    pass  # TODO: Add user logic
```

Empty Conditional Blocks

When a condition requires a block but you intentionally want no action.

```
if error_code == 0:
    pass  # No error, continue
else:
    print("Error occurred")
```

Testing and Debugging

Temporarily disable code sections during development without deleting them.

```
if debug_mode:
 pass # Skip debug output
 # print(debug_info)
```

CHAPTER 8

Real-World Decision Problems

Problem 1: ATM Withdrawal System

```
balance = 500
withdrawal_amount = 200
daily_limit = 1000
withdrawal_today = 300

print("=== ATM Withdrawal System ===")

if withdrawal_amount <= 0:
 print("Invalid amount")
elif withdrawal_amount > balance:
 print("Insufficient funds")
 print(f"Your balance: ${balance}")
elif withdrawal_today + withdrawal_amount > daily_limit:
 print("Daily limit exceeded")
 remaining = daily_limit - withdrawal_today
 print(f"You can withdraw up to ${remaining} today")
else:
 balance = balance - withdrawal_amount
 withdrawal_today = withdrawal_today + withdrawal_amount
 print("Withdrawal successful")
 print(f"New balance: ${balance}")
```

Problem 2: Movie Ticket Pricing

```
age = 14
is_student = True
is_weekend = False

base_price = 15

if age < 5:
    price = 0
    print("Free for children under 5")
elif age < 13:
    price = base_price * 0.5
    print("Child discount: 50% off")
elif age >= 65:
    price = base_price * 0.6
    print("Senior discount: 40% off")
elif is_student:
    price = base_price * 0.8
    print("Student discount: 20% off")
else:
    price = base_price

if is_weekend:
    price = price + 3
    print("Weekend surcharge: +$3")

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

This system demonstrates complex pricing logic with multiple conditions and nested calculations based on customer attributes and timing.

Problem 3: Weather Recommendation

```
temperature = 28
is_raining = False
wind_speed = 15
humidity = 70

print("=== Weather Activity Recommendation ===")

if is_raining:
 print("It's raining!")
 print("Recommendation: Stay indoors or bring umbrella")
else:
 if temperature > 30:
 print("It's very hot outside")
 if humidity > 80:
 print("High humidity - Stay hydrated!")
 print("Recommendation: Beach or pool")
 elif temperature > 20:
 print("Pleasant weather")
 if wind_speed > 20:
 print("Windy conditions")
 print("Recommendation: Kite flying or sailing")
 else:
 print("Recommendation: Outdoor sports or hiking")
 else:
 print("Cool weather")
 print("Recommendation: Walking or light exercise")
```

⚠ CHAPTER 9

Common Errors and Pitfalls

Typical Mistakes to Avoid

Indentation Errors

```
# Wrong
if x > 5:
print("Big")  # IndentationError

# Correct
if x > 5:
    print("Big")
```

Missing Colon

```
# Wrong
if x > 5  # SyntaxError
    print("Big")

# Correct
if x > 5:
    print("Big")
```

Assignment vs Comparison

```
# Wrong
if x = 5:  # SyntaxError
    print("Five")

# Correct
if x == 5:
    print("Five")
```

Incorrect Boolean Capitalization

```
# Wrong
if true: # NameError
 print("Yes")

# Correct
if True:
 print("Yes")
```

Logic Errors and Debugging Tips

→

Order of Conditions Matters

Place more specific conditions before general ones. If you check `x > 0` before `x > 10`, the second condition may never execute.

→

Empty Condition Blocks

Every if/elif/else block needs at least one statement. Use `pass` if you're not ready to implement the logic yet.

→

Using and/or Incorrectly

Remember: `and` requires both conditions True, `or` needs just one. Common mistake: `if x == 5 or 6` (wrong) vs `if x == 5 or x == 6` (correct).

→

Floating Point Comparisons

Avoid using `==` with floats due to precision issues. Instead, check if the difference is within a small threshold.

PCAP EXAM

Decision Making in PCAP Certification

The PCAP (Certified Associate in Python Programming) exam tests your understanding of conditional statements extensively. You'll need to:

- Understand Boolean logic and comparison operators
- Write correct if-elif-else structures
- Debug conditional logic errors
- Work with nested conditions
- Apply logical operators (and, or, not)

Decision-making concepts appear in approximately 20% of PCAP exam questions. Practice writing clean, logical conditional statements to succeed.

Exam Tip: Pay close attention to indentation, colons, and operator precedence in conditional statements—these are frequently tested areas.