Python Programming

CT108-3-1-PYP

0005 - Control Structures (Conditional Statement)

Topic Learning Outcomes

At the end of this topic, you should be able to:

- Understand the concept and usage of selection statements.
- Analyze the problem, decide and evaluate conditions.

Contents & Structure

- One Way Decision (If)
- Two-Way Decision (If Else)
- Multiway Decision (If Elif)
- Nested Decision

Boolean Expression

1. Boolean Expression

A **Boolean expression** is a logical statement that can have one of two values: **True** or **False**. It expresses a condition that can either be satisfied (True) or not satisfied (False). Boolean expressions are used in decision-making processes in programming, where different actions are taken based on whether a certain condition is met or not.

Boolean expressions often involve:

- **Comparison operators**: ==, !=, >, <, >=, <=
- **Logical operators**: and, or, not

Example 1:

- **Expression**: Is 10 greater than 5?
- 10 > 5 ⟶ **True**

Example 2:

- **Expression**: Is 3 equal to 4?
- 3 == 4 ⟶ **False**

Boolean expressions are crucial in if statements, loops, and other control structures that rely on decision-making.

2. Boolean Value

A **Boolean value** is the result of evaluating a Boolean expression, and it is either:

- **True** (Yes)
- **False** (No)

In programming, Boolean values are represented as True and False. These are used in conditions to control the flow of the program, and they also form the foundation of logic in algorithms.

Example:

```
x = 5
y = 10

# Boolean expression: Is x less than y?
result = x < y  # True
print(result)  # Outputs: True
```

3. Comparison Operators

These are used to compare two values, resulting in a Boolean value (True/False).

- == (Equal to): Checks if two values are equal.

```
5 == 5  # True
5 == 6  # False
```

- != (Not equal to): Checks if two values are not equal.

```
5 != 6  # True
5 != 5  # False
```

- > (Greater than): Checks if the left value is greater than the right value.

```
7 > 3  # True
3 > 7  # False
```

- < (Less than): Checks if the left value is less than the right value.

```
3 < 7  # True
7 < 3  # False
```

- >= (Greater than or equal to): Checks if the left value is greater than or equal to the right value.

```
5 >= 5  # True
6 >= 5  # True
```

- <= (Less than or equal to): Checks if the left value is less than or equal to the right value.

```
5 <= 5  # True
4 <= 5  # True
```

4. Logical Operators

- These operators are used to combine multiple Boolean expressions or invert them.
- **and**: Returns True if both conditions are True.

```
(5 > 3) and (7 > 4)  # True, because both conditions are True
(5 > 3) and (7 < 4)  # False, because the second condition is False
```

- **or**: Returns True if at least one of the conditions is True.

```
(5 > 3) or (7 < 4)  # True, because the first condition is True
(5 < 3) or (7 < 4)  # False, because both conditions are False
```

- **not**: Inverts the Boolean value.

```
not (5 > 3)  # False, because (5 > 3) is True, and `not` inverts it
```

5. Example Scenarios

Scenario 1: Decision Making

```
age = 20

# Boolean expression: Is age greater than or equal to 18?
if age >= 18:
    print("Eligible to vote")  # This will be executed because the condition is True
else:
    print("Not eligible to vote")
```

Scenario 2: Complex Condition

```
temperature = 25
weather = "sunny"

# Boolean expression with 'and'
if temperature > 20 and weather == "sunny":
    print("Go for a walk")  # This will be executed because both conditions are True
else:
    print("Stay indoors")
```

6. Boolean Value in Real-Life Questions

- **Do you want coffee?**
  - Answer: Yes or No
  - Boolean value: True (Yes) or False (No)

- **Is x greater than 10?**
  - Answer: Yes (True) if x > 10, No (False) otherwise.

- **Does 5 divide 153?**
  - Answer: No (153 is not divisible by 5)
  - Boolean value: False

In summary, Boolean expressions help to represent logic in code, while Boolean values help us understand whether a certain condition holds true or false.

How to Get Boolean Results

In programming, Boolean results (True or False) are obtained in two main ways:

- **Using Comparison Operators**: Compare two values to determine a relationship (e.g., equality, inequality).
- **Using Boolean (Logical) Operators**: Combine one or more Boolean expressions to produce a final Boolean result.

Let's go over both of these approaches in detail, with examples.

1. Comparison Operators

Comparison operators are used to compare two values. The result of any comparison is always a Boolean value (True or False).

1A. List of Comparison Operators

Operator	Description	Example	Result
==	Equal to	5 == 5	True
!=	Not equal to	5 != 3	True
>	Greater than	7 > 3	True
>=	Greater than or equal to	7 >= 7	True
<	Less than	4 < 6	True
<=	Less than or equal to	6 <= 6	True

1B. Explanation with Examples

Equal to (==):

- This operator checks if two values are the same.

```
5 == 5  # True
5 == 3  # False
```

- **Not equal to (!=)**:
  - This operator checks if two values are different.

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

- **Greater than (>)**:
  - This operator checks if the value on the left is greater than the value on the right.

```
7 > 3  # True
3 > 7  # False
```

- **Greater than or equal to (>=)**:
  - This operator checks if the value on the left is greater than or equal to the value on the right.

```
7 >= 5  # True
6 >= 5  # True
```

- **Less than (<)**:
  - This operator checks if the value on the left is less than the value on the right.

```
4 < 6  # True
6 < 4  # False
```

- **Less than or equal to (<=)**:
  - This operator checks if the value on the left is less than or equal to the value on the right.

```
6 <= 6  # True
5 <= 6  # True
```

- **Example in a Real Scenario**:

```
age = 18

# Check if a person is eligible to vote
is_eligible = age >= 18
print(is_eligible)  # True (because age is 18, which is equal to 18)
```

2. Boolean (Logical) Operators

Boolean operators allow you to combine one or more Boolean expressions. The result is also a Boolean value (True or False).

2A. List of Boolean Operators

Operator	Description	Example	Result
and	Returns True if **both** conditions are True	(5 > 3) and (7 > 4)	True
or	Returns True if **at least one** condition is True	(5 > 3) or (7 < 4)	True
not	Inverts the Boolean value (if True, returns False)	not (5 > 3)	False

2B. Explanation with Examples

- **and** Operator:
  - Both conditions must be True for the result to be True.

```
(5 > 3) and (7 > 4)  # True (because both conditions are True)
(5 > 3) and (7 < 4)  # False (because the second condition is False)
```

- **or** Operator:
  - At least one condition must be True for the result to be

```
(5 > 3) or (7 < 4)  # True (because the first condition is True)
(5 < 3) or (7 < 4)  # False (because both conditions are False)
```

- **not** Operator:
  - This operator inverts the Boolean value.

```
not (5 > 3)  # False (because (5 > 3) is True, and `not` inverts it)
not (5 < 3)  # True (because (5 < 3) is False, and `not` inverts it)
```

2C. Example in a Real Scenario

```
temperature = 25
weather = "sunny"

# Check if it's a good day for a walk
is_good_day_for_walk = (temperature > 20) and (weather == "sunny")
print(is_good_day_for_walk)  # True (because both conditions are True)

# Check if it's either hot or rainy
is_hot_or_rainy = (temperature > 30) or (weather == "rainy")
print(is_hot_or_rainy)  # False (because neither condition is True)
```

3. Combining Boolean Operators and Comparison Operators

You can combine comparison operators and Boolean operators in a single expression to make more complex conditions.

Example:

```
age = 25
is_student = True

# Check if the person is either over 18 or is a student
can_get_discount = (age > 18) or is_student
print(can_get_discount)  # True (because the first condition is True)

# Check if the person is over 18 and is a student
can_get_extra_discount = (age > 18) and is_student
print(can_get_extra_discount)  # True (because both conditions are True)
```

Explanation:

- **First Expression (or)**: The result is True if either the age is over 18 or the person is a student.
- **Second Expression (and)**: The result is True only if both conditions (age > 18 and is a student) are True.

4. Example: Using if Statements with Boolean and Comparison Operators

```
score = 85
attendance = 90

# Determine if a student passes
if (score >= 60) and (attendance >= 75):
   print("The student passes the course.")
else:
   print("The student fails.")
```

Explanation:

- The student will pass only if their score is 60 or higher **and** their attendance is 75% or higher.

In summary, Boolean results can be obtained through comparison operators (to compare values) or Boolean operators (to combine conditions). This allows for decision-making in programs, creating logic based on True or False values.

5. Quick Review of Boolean Logic

Let's break down the basic Boolean logic with examples using the and, or, and not operators.

5A. and Operator

The **and** operator returns True only if **both** conditions are True. Otherwise, it returns False.

Expression	Result
True and True	True
True and False	False
False and True	False
False and False	False

Explanation:

- **True and True**: Both sides are True, so the result is True.
- **True and False**: One side is False, so the result is False.
- **False and True**: One side is False, so the result is False.
- **False and False**: Both sides are False, so the result is False.

Example:

```
# Using 'and' in a program
age = 25
has_ticket = True

# Check if the person can enter the event
can_enter = (age > 18) and has_ticket
print(can_enter)  # True, because both conditions are True
```

5B. or Operator

The **or** operator returns True if **at least one** of the conditions is True. It only returns False if **both** conditions are False.

Expression	Result
True or True	True
True or False	True
False or True	True
False or False	False

Explanation:

- **True or True**: Both sides are True, so the result is True.
- **True or False**: At least one side is True, so the result is True.
- **False or True**: At least one side is True, so the result is True.
- **False or False**: Both sides are False, so the result is False.

Example:

```
# Using 'or' in a program
has_student_id = False
has_discount_coupon = True

# Check if the person can get a discount
eligible_for_discount = has_student_id or has_discount_coupon
print(eligible_for_discount)  # True, because one condition is True (discount coupon)
```

5C. not Operator

The **not** operator is a **negation** operator. It inverts the Boolean value:

- **not True** becomes **False**.
- **not False** becomes **True**.

Expression	Result
not True	False
not False	True

Explanation:

- **not True**: The negation of True is False.
- **not False**: The negation of False is True.

Example:

```
# Using 'not' in a program
is_raining = True

# Check if the weather is not rainy
is_not_raining = not is_raining
print(is_not_raining)  # False, because 'is_raining' is True
```

6. Combining and, or, and not

You can combine these Boolean operators to form more complex conditions.

Example 1:

```
# Example combining 'and', 'or', and 'not'
is_weekend = True
has_chores = False

# Check if you can relax
can_relax = is_weekend and not has_chores
print(can_relax)  # True, because it's the weekend and there are no chores
```

Example 2:

```
# More complex condition
is_sunny = True
is_weekend = False
has_free_time = True

# Check if you can go hiking (either it's sunny and the weekend, or you have free time)
can_go_hiking = (is_sunny and is_weekend) or has_free_time
print(can_go_hiking)  # True, because you have free time, even though it's not the weekend
```

Summary of Quick Review

- **and**: Only returns True when **both** conditions are True. Example: True and False ⟶ False
- **or**: Returns True if **at least one** of the conditions is True. Example: True or False ⟶ True
- **not**: Inverts the Boolean value. Example: not True ⟶ False

These logical operators allow for complex decision-making processes in programming.

A Note on Indentation

In Python, **indentation** is crucial for defining the scope of code blocks. Indentation determines which lines of code belong to a particular block, such as inside an if statement, a loop, or a function. Unlike some other programming languages that use braces {} or begin...end keywords, Python uses indentation to define these blocks.

1. Key Rules of Indentation

- **Increase indentation** after an if, for, or while statement (we'll cover these statements more later). This shows that the lines after these statements belong to the same block of code.
- **Maintain the same indentation** for every line within a block. This helps Python and the reader understand which lines are affected by a statement like if or for.
- **Decrease indentation** back to the level of the starting statement to indicate the end of a block.
- **Blank lines** do not affect indentation. You can add them for readability.
- **Comments** that appear on a line by themselves are also ignored by Python regarding indentation.

1A. Example

```
if x > 5:
    print("x is greater than 5")
    print("This is still part of the if block")
print("This line is outside the if block")
```

Explanation:

- The two lines after if x > 5: are indented, meaning they are part of the block that runs if the condition x > 5 is true.
- The last print statement is not indented, meaning it is outside of the if block and will run regardless of the condition.

1B. Further Example

Let’s look at indentation within a loop and conditional block:

```
x = 10
for i in range(x):  # Loop starts
    if i % 2 == 0:  # Check if i is even
        print(i, "is even")  # Indented, part of the if block
    else:
        print(i, "is odd")   # Indented, part of the else block
print("Done!")  # Not indented, outside of the for loop
```

Explanation:

- The for loop iterates through numbers from 0 to x-1.
- The if and else statements check if i is even or odd.
- The print("Done!") statement is outside of the loop because it is not indented and will run after the loop completes.

2. Indentation is Very Important in Python

In Python, **indentation is not optional**—it is the **only** way to define code blocks. Every indented section is treated as a block of code that belongs to a control structure, such as if, while, for, functions, etc.

2A. Key Points

- **Blocks can have as many statements as you like**: Once you increase the indentation, you can add multiple statements inside the block as long as they maintain the same level of indentation.
- Python **forces you to write clean code**: By enforcing indentation, Python encourages good programming habits that make your code easier to read, understand, and debug.
- Without proper indentation, you'll get **IndentationErrors**, which are syntax errors indicating that Python cannot properly parse the block structure.

2B. Example of Proper Indentation

```
def greet(name):
    if name:
        print(f"Hello, {name}!")
        print("Welcome to Python programming.")
    else:
        print("Name cannot be empty!")
```

Explanation:

- The function greet takes a name as a parameter.
- If a valid name is passed, two indented print statements execute. Both belong to the same if block.
- The else statement is at the same indentation level as the if, indicating that it's the alternative block.

2C. Example of Indentation Error

```
def greet(name):
    if name:
        print(f"Hello, {name}!")
    print("Welcome to Python programming.")
    else:
        print("Name cannot be empty!")
```

Error:

- In this code, the else statement is indented incorrectly. It should align with the if statement. Python will raise an IndentationError because the structure is incorrect.

3. Further Example: Complex Indentation

```
def process_number(x):
    if x > 0:
        print("Positive number")
        if x > 100:
            print("Large number")
        else:
            print("Small number")
    else:
        print("Non-positive number")
```

3A. Explanation

- if x > 0: starts a block for positive numbers.
- Inside this block, there's another if checking if the number is large or small. Both conditions and their associated statements are properly indented.
- The else for the outer if checks for non-positive numbers and is aligned with the first if.

Summary

- **Indentation** defines the scope of control structures (if, for, while, etc.) in Python. It is mandatory.
- **Proper indentation** ensures your code is readable, easier to debug, and prevents syntax errors.
- **Blank lines and comments** don’t affect indentation and are ignored in this regard.

Let's dive deeper into **indentation** with more specific cases like **functions**, **exception handling**, and other control structures.

4-1. Indentation in Functions

In Python, when you define a function using the def keyword, everything within the function body must be indented. The function block can contain control structures, and each of them needs to follow the indentation rules.

4A. Example: Indentation in Functions

```
def calculate_square(x):
    result = x * x  # This line is inside the function
    return result   # This is still inside the function

print(calculate_square(5))  # This is outside the function
```

Explanation:

- The statements result = x * x and return result are part of the function because they are indented.
- The print statement is outside the function and has no indentation.

If you forget to indent inside the function, Python will raise an IndentationError.

4B. Nested Functions and Indentation

Functions can also contain other functions or control structures, in which case, nested indentation is needed.

```
def outer_function():
    print("This is the outer function.")

    def inner_function():  # inner function definition
        print("This is the inner function.")

    inner_function()  # calling the inner function

outer_function()
```

Explanation:

- inner_function is defined inside outer_function and is indented to indicate its scope.
- When inner_function() is called, the indented print statement inside it runs.

4-2. Indentation in Exception Handling

Python uses try, except, else, and finally blocks to handle exceptions. Each block is followed by indented code to handle specific conditions.

4A. Example: Indentation in Exception Handling

```
def divide_numbers(a, b):
    try:
        result = a / b
    except ZeroDivisionError:  # Handles division by zero
        print("Error: Division by zero is not allowed.")
    else:  # Runs if no exception occurs
        print("The result is:", result)
    finally:  # Runs no matter what
        print("Operation complete.")

divide_numbers(10, 2)
divide_numbers(10, 0)
```

Explanation:

- The try block contains the code that might raise an exception.
- The except block handles the ZeroDivisionError.
- The else block runs only if no exception is raised.
- The finally block always runs, regardless of whether an exception occurs.
- Each block has indented code under it, indicating that these statements belong to their respective blocks.

4B. Example with Multiple Exceptions

```
def handle_exceptions(x):
    try:
        result = 10 / x
    except ZeroDivisionError:
        print("Cannot divide by zero!")
    except TypeError:
        print("Invalid type, expected a number!")
    finally:
        print("Completed the exception handling.")

handle_exceptions(0)
handle_exceptions("a")
```

Explanation:

- The try block attempts to divide 10 by x.
- Depending on the type of exception (ZeroDivisionError or TypeError), the corresponding except block runs.

4-3. Indentation in Loops

Loops (for and while) are another case where indentation is essential to show which statements belong inside the loop.

4A. Example: Indentation in Loops

```
for i in range(3):
    print("Loop iteration:", i)
    for j in range(2):
        print("  Inner loop iteration:", j)
    print("Exiting inner loop.")
print("Outside both loops.")
```

Explanation:

- The for loop iterates three times (i = 0, 1, 2).
- Inside this loop, another loop runs for j = 0, 1.
- Each inner loop iteration is indented further than the outer loop.
- The last print("Outside both loops.") is at the outer level, so it runs after the outer loop completes.

4-4. Indentation In Conditional Statements (If-Elif-Else)

We have already seen simple examples of if statements, but Python supports multiple conditions with elif (else if), and all blocks must follow indentation rules.

4A. Example: Indentation in Conditional Statements

```
def categorize_number(x):
    if x > 0:
        print("Positive number")
    elif x == 0:
        print("Zero")
    else:
        print("Negative number")

categorize_number(10)
categorize_number(0)
categorize_number(-5)
```

Explanation:

- The if block checks if x is positive.
- If not, the elif block checks if x is zero.
- If neither condition is true, the else block handles the negative numbers.
- Each block has indented code inside it.

4B. Example: Multiple if Conditions

```
def check_conditions(x, y):
    if x > 0:
        print("x is positive")
        if y > 0:
            print("y is also positive")
        else:
            print("y is not positive")
    else:
        print("x is not positive")

check_conditions(10, -5)
```

Explanation:

- The function contains nested if statements. The outer if checks whether x is positive. If true, it enters another if block to check whether y is positive.
- Proper indentation helps clarify which condition and block a statement belongs to.

4-5. Indentation in Classes and Methods

When defining a class, the methods inside it need to be indented. Each method body is a block that belongs to the class, and the code inside each method follows indentation rules like functions.

4A. Example: Indentation in Classes and Methods

```
class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b

calc = Calculator()
print(calc.add(5, 3))
print(calc.subtract(5, 3))
```

Explanation:

- The class Calculator contains two methods (add and subtract).
- Each method is indented inside the class definition, and each method body is indented to show the code belongs inside the method.

4B. Example: Class with Conditional Logic

```
class NumberCheck:
    def check_number(self, num):
        if num > 0:
            print("Positive")
        elif num == 0:
            print("Zero")
        else:
            print("Negative")

checker = NumberCheck()
checker.check_number(10)
checker.check_number(0)
checker.check_number(-10)
```

Explanation:

- The check_number method has conditional logic (if-elif-else) with proper indentation for each block.
- The method is indented inside the class.

Conclusion

- **Functions**: Indentation helps group related code under function definitions.
- **Exception Handling**: try, except, else, and finally blocks require proper indentation to define which code belongs to each.
- **Loops and Conditionals**: for, while, if, elif, and else all use indentation to define their scope.
- **Classes and Methods**: Class methods must be indented within the class, and method bodies need proper indentation for internal control structures.

5. Exercise

Let’s move forward with some exercises and project applications to reinforce your understanding of indentation in Python. I'll guide you through different scenarios that you can try out in your development environment. Each exercise will test different concepts such as functions, loops, exception handling, and class structures.

Exercise 1: Fix the Indentation

In the following code, there are indentation errors. Correct the indentation to ensure the program works correctly.

```
def check_temperature(temp):
if temp > 30:
print("It's hot!")
elif temp >= 20:
print("It's warm.")
else:
print("It's cold.")

check_temperature(25)
```

Steps:

- Identify the missing indentations.
- Fix the code so it runs without errors.

Corrected Code:

```
def check_temperature(temp):
    if temp > 30:
        print("It's hot!")
    elif temp >= 20:
        print("It's warm.")
    else:
        print("It's cold.")

check_temperature(25)
```

Explanation:

- The if, elif, and else blocks are now indented inside the function.
- Each condition (if temp > 30, elif temp >= 20, etc.) is followed by an indented print() statement.

Exercise 2: Nested Loops and Conditionals

Write a Python program that uses nested loops to print a pattern and includes conditionals inside the loop. You should use proper indentation to structure the code.

Goal:

- Print the following pattern using nested loops and conditionals:

```
*
**
***
****
*****
```

Hints:

- Use a for loop to handle the rows.
- Inside the loop, use another for loop to print the stars (*).
- Use conditionals to add logic (e.g., print a message when a row has an even number of stars).

Solution:

```
def print_pattern():
    for i in range(1, 6):  # Outer loop for rows (1 to 5)
        for j in range(i):  # Inner loop to print stars
            print('*', end='')  # Print star, stay on the same line
        print()  # Move to the next line after each row

print_pattern()
```

Explanation:

- The outer loop runs for 5 iterations (1 to 5), controlling the number of rows.
- The inner loop prints i stars in each row.
- The print() at the end moves to the next line after each row is printed.

Exercise 3: Indentation in Exception Handling

Write a function divide_numbers that takes two arguments a and b. If b is zero, catch the ZeroDivisionError and print an error message. Regardless of whether an error occurs, print "Operation complete" at the end.

Expected Output Example:

```
divide_numbers(10, 2)  # Should print "Result: 5.0" and "Operation complete"
divide_numbers(10, 0)  # Should print "Cannot divide by zero!" and "Operation complete"
```

Goal:

- Write a function divide_numbers that handles ZeroDivisionError and prints a message.

Solution:

```
def divide_numbers(a, b):
    try:
        result = a / b
        print("Result:", result)
    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
    finally:
        print("Operation complete.")

divide_numbers(10, 2)  # Should print "Result: 5.0" and "Operation complete"
divide_numbers(10, 0)  # Should print "Error: Cannot divide by zero!" and "Operation complete"
```

Explanation:

- The try block attempts to divide a by b.
- If b is zero, the except ZeroDivisionError block runs, catching the error and printing an error message.
- The finally block always runs, whether or not an exception occurs, to print "Operation complete."

Exercise 4: Class and Methods with Conditionals

Create a class BankAccount with the following features:

- A constructor that initializes the account balance.
- A method deposit(amount) that adds to the balance.
- A method withdraw(amount) that subtracts from the balance, but only if the withdrawal amount is less than or equal to the current balance.
- Ensure proper indentation is used, especially inside the methods with conditional statements.

Example Usage:

```
account = BankAccount(100)
account.deposit(50)  # Balance should now be 150
account.withdraw(70)  # Balance should now be 80
account.withdraw(200)  # Should print an error message: "Insufficient funds"
```

Goal:

- Create a BankAccount class with deposit, withdraw, and proper condition handling.

Solution:

```
class BankAccount:
    def __init__(self, initial_balance):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance is {self.balance}.")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew {amount}. New balance is {self.balance}.")
        else:
            print("Error: Insufficient funds.")

# Example usage:
account = BankAccount(100)
account.deposit(50)  # Should print "Deposited 50. New balance is 150."
account.withdraw(70)  # Should print "Withdrew 70. New balance is 80."
account.withdraw(200) # Should print "Error: Insufficient funds."
```

Explanation:

- The BankAccount class has a constructor to initialize the balance.
- The deposit method adds money to the balance, while the withdraw method checks if the amount can be withdrawn based on the current balance. If not, it prints an error message.
- Proper indentation ensures that the condition (if amount <= self.balance) is correctly placed inside the withdraw method.

Exercise 5: Functions with Nested Conditionals and Loops

Write a function grade_students that takes a list of student grades and prints a message for each student based on their score:

- If the grade is 90 or above, print "Excellent!"
- If the grade is between 80 and 89, print "Good job!"
- If the grade is between 70 and 79, print "You passed!"
- If the grade is below 70, print "You need to improve."

Ensure proper indentation for the nested conditional logic.

Example Input:

```
grades = [95, 82, 67, 74]
grade_students(grades)
```

Expected Output:

```
Excellent!
Good job!
You need to improve.
You passed!
```

Goal: Write a function grade_students that categorizes student grades based on their score.

Solution:

```
def grade_students(grades):
    for grade in grades:
        if grade >= 90:
            print("Excellent!")
        elif grade >= 80:
            print("Good job!")
        elif grade >= 70:
            print("You passed!")
        else:
            print("You need to improve.")

# Example input:
grades = [95, 82, 67, 74]
grade_students(grades)
```

Output:

```
Excellent!
Good job!
You need to improve.
You passed!
```

Explanation:

- The for loop iterates over the list of grades, and nested if-elif-else conditions are used to classify each grade.
- Proper indentation ensures that the correct message is printed based on the grade.

Conditions in Python

In Python, conditional structures allow a program to execute specific pieces of code based on whether a certain condition is true or false. These conditions are expressions that evaluate to either True or False. These conditional statements are useful for decision-making and controlling the flow of the program.

1. Key Conditional Structures

- **if statement**
- **else statement**
- **elif (else if) statement**

Let's break down each of these structures and how they work in conjunction with conditions.

2. The Condition Expression

A condition is an expression that can be evaluated to either True or False. A simple condition could involve comparison operators such as >, <, ==, !=, etc.

For example:

```
x = 10
condition = x < 0
```

In this case:

- The condition x < 0 evaluates to either True or False, depending on the value of x.
- If x is less than 0, the condition is True; otherwise, it is False.

Example:

```
x = -5
if x < 0:
  print("x is negative")
```

- If x is -5, the condition x < 0 is True, and the program prints "x is negative".
- If x is 5, the condition x < 0 is False, and the program skips the print statement.

3. if Statement

The if statement is used to check a condition. If the condition evaluates to True, the block of code under the if statement is executed.

Syntax:

```
if condition:
    # Code block executed if condition is True
```

Example:

```
temperature = 30
if temperature > 25:
    print("It's a hot day!")
```

- Here, the condition temperature > 25 is checked.
- If the temperature is higher than 25, the message "It's a hot day!" will be printed.

4. else Statement

The else statement provides an alternative block of code that will run if the condition in the if statement evaluates to False.

Syntax:

```
if condition:
    # Code block executed if condition is True
else:
    # Code block executed if condition is False
```

Example:

```
age = 16
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

- If age is 16, the condition age >= 18 is False, so the program prints "You are a minor.".
- If age were 20, the condition would be True, and it would print "You are an adult.".

5. elif (Else If) Statement

The elif statement is used when you want to check multiple conditions. It stands for "else if," and allows you to test another condition if the previous one was False.

Syntax:

```
if condition1:
    # Code block executed if condition1 is True
elif condition2:
    # Code block executed if condition2 is True
else:
    # Code block executed if both condition1 and condition2 are False
```

Example

```
score = 75
if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: F")
```

- If score is 75, the first two conditions (score >= 90 and score >= 80) are False.
- The third condition score >= 70 is True, so it prints "Grade: C".
- If none of the conditions are True, the else block is executed, printing "Grade: F".

6. Combining Conditions with Logical Operators

You can combine multiple conditions using logical operators like and, or, and not.

Example with and:

```
x = 5
y = 10
if x > 0 and y > 0:
   print("Both x and y are positive.")
```

- Both conditions x > 0 and y > 0 must be True for the block to execute.

Example with or:

```
x = -5
y = 10
if x > 0 or y > 0:
   print("At least one variable is positive.")
```

- If either x > 0 or y > 0 is True, the block will execute.

Example with not:

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

- The condition not is_raining is True if is_raining is False.

7. Nested Conditionals

You can nest if statements inside other if statements to create more complex decision-making structures.

Example:

```
x = 5
if x > 0:
   if x < 10:
      print("x is a positive single-digit number.")
```

- The outer if checks if x is positive.
- The inner if checks if x is less than 10.

Conclusion

Python's conditional structures provide powerful control over the flow of a program based on true or false conditions. You can:

- Check a single condition with if.
- Provide alternatives using else.
- Test multiple conditions with elif.
- Combine conditions using logical operators like and, or, and not.

8. Further Examples

8A. Age-based Conditions

- Depending on the age, this will print "Minor", "Adult", or "Senior".

```
age = 25
if age < 18:
   print("Minor")
elif age < 65:
   print("Adult")
else:
   print("Senior")
```

8B. Temperature Check

- Different temperature ranges lead to different outputs.

```
temp = 10
if temp > 30:
   print("It's very hot!")
elif temp > 20:
   print("It's warm.")
elif temp > 10:
   print("It's cool.")
else:
   print("It's cold.")
```

Conditional Statements (Decision Making) In Python

In Python, **decision-making** allows a program to evaluate specific conditions and decide the flow of execution based on whether those conditions are **True** or **False**. Conditional statements are a fundamental concept in programming, allowing a program to behave differently under different circumstances.

Python provides several types of **conditional (decision-making) statements**, including:

1. **if statement**
2. **if...else statement**
3. **if...elif...else statement**
4. **Nested if...else statement**

Let's explore each one in detail with examples.

1. if Statement

The if statement is the simplest form of decision-making structure. It executes a block of code only if the condition provided evaluates to True. If the condition is False, the block of code is skipped.

1A. Syntax

```
if condition:
    # Block of code executed if the condition is True
```

1B. Example

```
number = 10
if number > 5:
    print("The number is greater than 5.")
```

- **Explanation**: The condition number > 5 is evaluated.
    - If number is greater than 5, the message "The number is greater than 5." is printed.
    - If the condition was False (for example, if number = 3), the code block would not execute.

1C. Further Example

```
is_raining = True
if is_raining:
    print("Take an umbrella.")
```

- **Explanation**: If the variable is_raining is True, the program prints "Take an umbrella.". If is_raining is False, nothing is printed.

2. if...else Statement

The if...else statement allows you to provide an alternative block of code that will execute if the condition in the if statement evaluates to False.

2A. Syntax

```
if condition:
    # Block of code executed if the condition is True
else:
    # Block of code executed if the condition is False
```

2B. Example

```
age = 16
if age >= 18:
    print("You are eligible to vote.")
else:
    print("You are too young to vote.")
```

- **Explanation**: The condition number > 5 is evaluated.
    - If age >= 18, the first block prints "You are eligible to vote.".
    - Otherwise, the else block executes, printing "You are too young to vote.".

2C. Further Example

```
temperature = 30
if temperature > 25:
    print("It's hot today.")
else:
    print("The weather is cool.")
```

- **Explanation**: If temperature is greater than 25, it prints "It's hot today.". If not, it prints "The weather is cool.".

3. if...elif...else Statement

The if...elif...else statement is used to test multiple conditions. It allows you to chain several conditions together. When one of the conditions evaluates to True, its corresponding block of code is executed, and the rest of the conditions are skipped.

3A. Syntax

```
if condition1:
    # Block of code executed if condition1 is True
elif condition2:
    # Block of code executed if condition2 is True
else:
    # Block of code executed if neither condition1 nor condition2 is True
```

3B. Example

```
marks = 85
if marks >= 90:
    print("Grade: A")
elif marks >= 80:
    print("Grade: B")
elif marks >= 70:
    print("Grade: C")
else:
    print("Grade: F")
```

- **Explanation**:
    - If marks >= 90, the first block prints "Grade: A".
    - If marks is between 80 and 89, the second block prints "Grade: B".
    - If marks is between 70 and 79, it prints "Grade: C".
    - If none of the above conditions are true, the else block prints "Grade: F".

3C. Further Example

```
time = 14
if time < 12:
   print("Good morning!")
elif time < 18:
   print("Good afternoon!")
else:
   print("Good evening!")
```

- **Explanation**: Based on the time of day, the program greets the user accordingly:
    - If time is less than 12, it prints "Good morning!".
    - If time is between 12 and 18, it prints "Good afternoon!".
    - Otherwise, it prints "Good evening!".

4. Nested if...else Statement

A nested if...else statement is when you place one if statement inside another if or else block. This allows for more complex decision-making, where additional conditions are checked inside another condition.

4A. Syntax

```
if condition1:
    # Block of code executed if condition1 is True
    if condition2:
        # Block of code executed if condition2 is also True
    else:
        # Block of code executed if condition1 is True but condition2 is False
else:
    # Block of code executed if condition1 is False
```

4B. Example

```
num = 10
if num > 0:
    print("The number is positive.")
    if num % 2 == 0:
        print("The number is even.")
    else:
        print("The number is odd.")
else:
    print("The number is negative.")
```

- **Explanation**:
    - The first if checks if num is greater than 0.
    - If num > 0 (which is True for 10), it prints "The number is positive.".
    - Then, inside the first if block, it checks if num is even (num % 2 == 0).
    - Since 10 is even, it prints "The number is even.". Otherwise, it would print "The number is odd.".
    - If the number had been negative, the outer else would print "The number is negative.".

4C. Further Example

```
x = 15
if x > 0:
    if x < 10:
        print("x is a positive single-digit number.")
    else:
        print("x is a positive number with more than one digit.")
else:
    print("x is negative or zero.")
```

- **Explanation**:
  - The first if checks if x > 0.
  - If x is greater than 0, the nested if checks if x < 10.
  - If x is less than 10, it prints "x is a positive single-digit number.".
  - If x is not less than 10, it prints "x is a positive number with more than one digit.".
  - If x were negative or zero, it would print "x is negative or zero.".

Summary

In Python, decision-making allows you to execute certain blocks of code based on whether conditions are True or False. Python provides several types of conditional statements:

- **if statement**: Executes a block of code if the condition is True.
  - Example: Checking if a number is positive.
- **if...else statement**: Executes one block of code if the condition is True and another block if the condition is False.
  - Example: Checking if a person is old enough to vote.
- **if...elif...else statement**: Used to test multiple conditions, where each condition is evaluated in sequence until one is True.
  - Example: Assigning letter grades based on a score.
- **Nested if...else statement**: Allows you to nest multiple if statements for complex decision-making.
  - Example: Checking if a number is positive, then checking if it's even or odd.

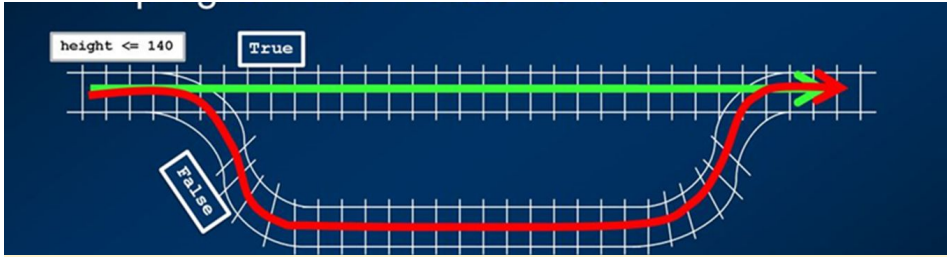
These conditional structures allow you to control the flow of your program based on the logic and rules you want to implement.

If Statement - One Way Decision

The **if statement** in Python is used to make one-way decisions based on a condition. The condition evaluates to either True or False, and if it is True, the block of code inside the if statement is executed. Otherwise, it is skipped.

1. Breakdown Of The if Statement

- **The Condition:** The condition is a Boolean expression (one that evaluates to True or False). If the condition is True, Python executes the statements in the block of code that follows the if statement.
- **The Block of Code:** The code inside the block will only run if the condition evaluates to True. This block is indented to show that it belongs to the if statement.
- **Flow:**
  - The condition is checked.
  - If it's True, the indented block of code inside the if statement is executed.
  - If it's False, Python moves on without executing the block.



2. Example: Problem 1 - MRT (Metro Rail Transport)

Let's say we need to determine if a child qualifies for a free MRT ride based on their height. Children whose height is **not larger than or equal to 140cm** can enjoy a free ride.

2A. Problem Statement

If a child's height is less than 140 cm, they can ride for free.

2B. Solution Using if Statement

```
# Get the child's height
height = float(input("Enter the child's height in cm: "))

# Check if the child qualifies for a free ride
if height < 140:
    print("The child can enjoy a free ride.")
```

Explanation

- **Input**: The input() function is used to collect the height of the child from the user, which is then converted to a float for comparison.
- **Condition**: if height < 140: checks if the child's height is less than 140 cm. If it is True, the code inside the if block is executed, which prints "The child can enjoy a free ride."
- **Flow**:
  - If the height entered is less than 140, the output will be: *"The child can enjoy a free ride."*
  - If the height is equal to or greater than 140, nothing happens, since we only provided an if condition and no else.

Example Run 1:

- Input: 135
- Output: The child can enjoy a free ride.

Example Run 2:

- Input: 145
- Output: *(No output because the child does not qualify for a free ride and we haven't defined any action for this case.)*

2C. Adding an else for Clarity

We can extend the example with an else statement to handle the case when the child doesn't qualify for a free ride.

```
# Get the child's height
height = float(input("Enter the child's height in cm: "))

# Check if the child qualifies for a free ride
if height < 140:
   print("The child can enjoy a free ride.")
else:
   print("The child does not qualify for a free ride.")
```

Flow with else:

- If the condition height < 140 is True, the message about the free ride is printed.
- If the condition is False, the else block is executed, and the message saying the child doesn't qualify for a free ride is displayed.

Example Run 3:

- Input: 145
- Output: The child does not qualify for a free ride.

This is a basic introduction to the **if** statement (one-way decision) in Python, and how it can be used to control the flow of a program based on a condition.

If-Else Statement (Two-Way Decision)

The **if-else statement** in Python is a **two-way decision** structure, which allows your program to choose between two alternative paths. Based on whether the condition evaluates to True or False, the program will either execute one block of code or another.

1. Breakdown Of The If-Else Statement

- **Condition:** The condition is a Boolean expression that will evaluate to either True or False. This condition helps the program decide which block of code to execute.
- **if Block (True Path):** If the condition evaluates to True, the block of code immediately following the if statement is executed.
- **else Block (False Path):** If the condition evaluates to False, the block of code following the else statement is executed.
- **Flow:**
    - The condition is checked.
    - If True, execute the code under the if block.
    - If False, execute the code under the else block.

This structure is known as **alternative execution** because there are two possible outcomes and only one is chosen based on the result of the condition.

1A. Syntax

```
if condition:
    # Body of if
else:
    # Body of else
```

2. Example: Problem 2 - MRT Ride (Extended)

Let's enhance the previous problem about children riding the MRT. We'll check if a child qualifies for a free ride based on height. If their height is less than 140 cm, they can ride for free. If their height is 140 cm or taller, they need to pay.

2A. Problem Statement

If a child's height is less than 140 cm, they ride for free. Otherwise, they need to pay for the ride.

2B. Solution Using if-else

```
# Get the child's height
height = float(input("Enter the child's height in cm: "))

# Check if the child qualifies for a free ride
if height < 140:
    print("The child can enjoy a free ride.")
else:
    print("The child must pay for the ride.")
```

Explanation:

- **Input**: The program asks the user to input the child's height and converts it to a float.
- **Condition**: The condition if height < 140: checks whether the child's height is less than 140 cm.
    - If True, the message "The child can enjoy a free ride" is printed.
    - If False (meaning the height is 140 cm or more), the else block runs, printing "The child must pay for the ride."
- **Flow**:
    - **True path**: If the height is less than 140, the if block is executed.
    - **False path**: If the height is 140 or greater, the else block is executed.

Example Run 1:

- Input: 135
- Output: The child can enjoy a free ride.

Example Run 2:

- Input: 150
- Output: The child must pay for the ride.

3. Further Example: Even or Odd Number Checker

This example checks whether a given number is even or odd.

3A. Problem Statement

Write a program that checks whether a number entered by the user is even or odd.

3B. Solution

```
# Get the number from the user
number = int(input("Enter a number: "))

# Check if the number is even or odd
if number % 2 == 0:
    print(f"{number} is an even number.")
else:
    print(f"{number} is an odd number.")
```

Explanation:

- **Input**: The program asks the user to enter a number, which is then converted to an integer.
- **Condition**: The condition `if number % 2 == 0:` checks if the remainder when the number is divided by 2 is zero (which indicates the number is even).
    - If True, it prints that the number is even.
    - If False, it prints that the number is odd.
- **Flow**:
    - **True path**: If the number is divisible by 2 with no remainder (even), the `if` block is executed.
    - **False path**: If there is a remainder (odd), the `else` block is executed.

Example Run 1:

- Input: 6
- Output: 6 is an even number.

Example Run 2:

- Input: 7
- Output: 7 is an odd number.

Flow of Execution:

- The condition is evaluated: `if number % 2 == 0`.
- If the number is divisible by 2 (i.e., True), the `if` block executes.
- If not (i.e., False), the `else` block executes.

Summary of if-else (Two-Way Decision)

- The **if-else statement** allows the program to make a choice based on a condition.
- If the condition evaluates to True, the program executes the code in the if block.
- If the condition evaluates to False, the program executes the code in the else block.

This is how Python allows alternative execution based on conditions to handle different outcomes depending on the data provided.

Else In Python

In Python, the else statement is used to define an alternative block of code that will be executed if the condition in the corresponding if statement evaluates to False. This creates a **two-way decision** in your code, meaning the program will either execute the code inside the if block (if the condition is True) or execute the code inside the else block (if the condition is False).

1. Key Points About The else Statement

- **Position**: The else clause must **immediately follow** the if block. There should be no code in between the if and else blocks.

- **Indentation**: The else statement must be indented to the **same level** as its corresponding if statement. If the indentation is incorrect (e.g., the else is indented differently than the if), the Python interpreter will raise an **IndentationError**.

- **Colon :**: Both the if and else clauses are followed by a colon (:), which indicates that the following indented block is part of that clause.

- **Mutual Exclusivity**: Only one of the two blocks of code (either the if or else block) will be executed, depending on whether the condition is True or False. If the condition is True, the if block executes, and the else block is skipped. If the condition is False, the else block executes.

1A. Syntax Of else In Python

```
if condition:
    # Block of code if condition is True
else:
    # Block of code if condition is False
```

2. Example 1: Determining If A Number Is Positive Or Negative

Let's write a simple program that checks if a number is positive or negative.

2A. Problem Statement

Write a program that takes a number as input and prints whether it is positive or negative. If the number is zero, it should be considered positive.

2B. Solution Using if-else

```
# Get the number from the user
number = float(input("Enter a number: "))

# Check if the number is positive or negative
if number >= 0:
   print("The number is positive.")
else:
   print("The number is negative.")
```

Explanation:

- **Input**: The program asks the user to input a number, which is converted to a floating-point number using float().
- **Condition**: The condition if number >= 0: checks whether the number is greater than or equal to 0.
    - If the condition is True (meaning the number is positive or zero), the if block is executed, and "The number is positive" is printed.
    - If the condition is False (meaning the number is negative), the else block is executed, and "The number is negative" is printed.
- **Flow**:
    - **True path**: If the number is greater than or equal to zero, the if block is executed.
    - **False path**: If the number is less than zero, the else block is executed.

Example Run 1:

- Input: 5
- Output: The number is positive.

Example Run 2:

- Input: -3
- Output: The number is negative.

Example Run 3:

- Input: 0
- Output: The number is positive.

2C. Indentation and Errors

As mentioned, the else block must be properly aligned with its corresponding if block. Here's an example that would cause an **IndentationError** due to incorrect indentation:

```
if number >= 0:
   print("The number is positive.")
 else:  # WRONG! This line is incorrectly indented
   print("The number is negative.")
```

If you run this code, Python will generate an error similar to:

```
IndentationError: unindent does not match any outer indentation level
```

The correct version should look like this:

```
if number >= 0:
   print("The number is positive.")
else:  # This line is now correctly indented
   print("The number is negative.")
```

3. Example 2: Checking If Someone Passed Or Failed

Let's create a program that determines if a student passed or failed based on their score.

3A. Problem Statement

Write a program that takes a student's score as input and checks whether they passed (a score of 50 or more) or failed (less than 50).

3B. Solution Using if-else

```
# Get the student's score
score = int(input("Enter the student's score: "))

# Check if the student passed or failed
if score >= 50:
   print("The student passed.")
else:
   print("The student failed.")
```

Explanation:

- **Input**: The program asks the user to enter the student's score, which is converted to an integer using int().
- **Condition**: The condition if score >= 50: checks whether the score is 50 or more.
    - If True (the score is 50 or more), the if block is executed, and "The student passed" is printed.
    - If False (the score is less than 50), the else block is executed, and "The student failed" is printed.
- **Flow**:
    - **True path**: If the score is greater than or equal to 50, the if block is executed.
    - **False path**: If the score is less than 50, the else block is executed.

Example Run 1:

- Input: 65
- Output: The student passed.

Example Run 2:

- Input: 45
- Output: The student failed.

4. Example 3: Is It Hot or Cold?

Let's build a program that checks if the temperature outside is hot or cold.

4A. Problem Statement

Write a program that takes the temperature as input and prints whether it is hot or cold based on a threshold of 30°C.

4B. Solution Using if-else

```
# Get the temperature
temperature = float(input("Enter the temperature in Celsius: "))

# Check if it is hot or cold
if temperature >= 30:
   print("It is hot outside.")
else:
   print("It is cold outside.")
```

Explanation:

- **Input**: The program asks the user to input the temperature in Celsius, which is converted to a floating-point number.
- **Condition**: The condition if temperature >= 30: checks whether the temperature is 30°C or more.
    - If True, the if block executes, and "It is hot outside" is printed.
    - If False, the else block executes, and "It is cold outside" is printed.

Example Run 1:

- Input: 32
- Output: It is hot outside.

Example Run 2:

- Input: 22
- Output: It is cold outside.

Summary Of else Statement

- The else block allows you to provide an **alternative path** when the condition in the if statement evaluates to False.
- The else block must **immediately follow** the if block and must have the same level of indentation.
- The colon (:) following both if and else indicates that the block of code beneath them is part of the condition or alternative action.

If-Else Statement (Two-Way Decision) Exercise

1. If-Else Statement (Two-Way Decision) in Python

In an **if-else structure**, there are always **two possible paths** of execution:

- One block executes if the condition evaluates to True.
- The other block executes if the condition evaluates to False. The key point is that **exactly one** of the two blocks will execute, ensuring that the program always makes a decision and follows one path.

Let's go through the examples you provided before solving the exercise.

1A. Example 1: Checking If A Number Is Negative Or Positive

```
if x < 0:
   print("Negative")
else:
   print("Positive")
```

Explanation:

- **Condition**: The condition x < 0 checks if x is less than 0.
  - If True, it prints "Negative".
  - If False, it prints "Positive".
- **Flow**:
  - **True path**: If x is less than 0, the code inside the if block is executed.
  - **False path**: If x is greater than or equal to 0, the code inside the else block is executed.

Example Run:

- Input: x = -5
- Output: Negative
- Input: x = 3
- Output: Positive

1B. Example 2: Checking If A Student Passed Or Failed

```
if mark >= 50:
   print("Pass")
else:
   print("Fail")
```

Explanation:

- **Condition**: The condition mark >= 50 checks if the mark is 50 or more.
  - If True, it prints "Pass".
  - If False, it prints "Fail".
- **Flow**:
  - **True path**: If the mark is 50 or more, the code in the if block is executed.
  - **False path**: If the mark is less than 50, the code in the else block is executed.

Example Run:

- Input: mark = 75
- Output: Pass
- Input: mark = 45
- Output: Fail

2. Exercise: Program 1 – Checking If A Number Is Odd Or Even

In this exercise, we will write a Python program to check whether a number is **odd** or **even**. The number will be entered by the user, and we'll use the **if-else structure** to determine whether the number is divisible by 2 (even) or not (odd).

2A. Problem Statement

Write a Python program that takes an integer input from the user and checks if the number is **odd** or **even** using an if-else statement.

2B. Solution

```
# Get a number from the user
number = int(input("Enter a number: "))

# Check if the number is even or odd
if number % 2 == 0:
   print(f"{number} is an even number.")
else:
   print(f"{number} is an odd number.")
```

2C. Explanation

- **Input**:
  - The program prompts the user to enter a number using the input() function. The int() function is used to convert the input to an integer.

- **Condition**:
  - The condition if number % 2 == 0: checks if the remainder when the number is divided by 2 is zero (i.e., if the number is evenly divisible by 2).
  - **True Path**: If the condition evaluates to True (meaning the number is even), the program prints that the number is even.
  - **False Path**: If the condition evaluates to False (meaning the number is not evenly divisible by 2, or it is odd), the program prints that the number is odd.

- **Flow**:
  - **True path**: If the number is divisible by 2 (even), the code inside the if block executes.
  - **False path**: If the number is not divisible by 2 (odd), the code inside the else block executes.

2D. Program Flow

- The user is prompted to enter a number.
- The number is checked using the modulus operator % to determine if it is divisible by 2:
  - If number % 2 == 0 is True, it means the number is even.
  - Otherwise, it is odd.
- Based on the result of the condition, either the **even** or **odd** message is displayed.

2E. Example Runs

Example Run 1:

- Input: 4
- Output: 4 is an even number.

Example Run 2:

- Input: 7
- Output: 7 is an odd number.

Example Run 3:

- Input: 0
- Output: 0 is an even number.

3. Further Explanation Of The Modulus Operator %

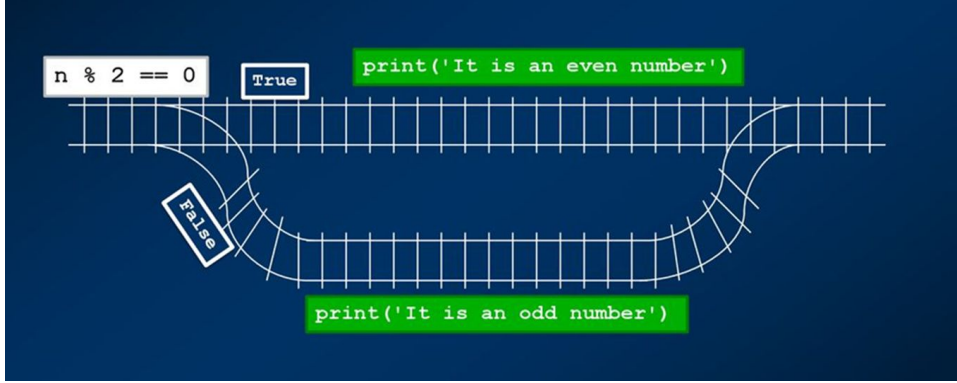
The modulus operator (%) returns the remainder when dividing two numbers. When we divide any **even number** by 2, the remainder is 0. For example:

- 4 % 2 = 0 (4 is even)
- 10 % 2 = 0 (10 is even)

For **odd numbers**, the remainder when divided by 2 is always 1. For example:

- 3 % 2 = 1 (3 is odd)
- 7 % 2 = 1 (7 is odd)

This is why we check the condition number % 2 == 0 to see if a number is even.



Summary Of If-Else In This Example

- The program checks the condition if number % 2 == 0:
  - If True, the number is even, and the if block executes.
  - If False, the number is odd, and the else block executes.

Nested “if-else” Statements In Python

Overview:

- A *nested* "if-else" statement refers to an "if" statement inside another "if-else" block. This allows for checking multiple conditions sequentially, leading to a more refined decision-making process in the code. The *outer* condition is checked first, and then based on its result, the *inner* conditions are evaluated.

- This helps handle complex logic where more than one condition needs to be checked in stages.

1. Structure of Nested "if-else"

In nested "if-else", an "if-elif-else" block can be placed inside another "if-elif-else" block.

1A. Key Points

- Each inner "if-else" can handle a condition dependent on the outer condition being true or false.
- You can nest as many levels of "if-else" statements as needed, although readability can decrease with many levels.

1B. Example for Nested "if-else"

```
if condition1:
    # Outer if block
    if condition2:
        # Inner if block
        print("Condition1 and Condition2 are true.")
    else:
        # Inner else block
        print("Condition1 is true, but Condition2 is false.")
else:
    # Outer else block
    if condition3:
        # Inner if block within the outer else
        print("Condition1 is false, but Condition3 is true.")
    else:
        # Inner else block within the outer else
        print("Both Condition1 and Condition3 are false.")
```

2. Syntax For Nested "If-Else"

- **Non-Linear Nested "if-else":** In this type, the inner condition blocks depend on the result of the outer condition.

```
if outer_condition:
    if inner_condition_1:
        # Do something if both outer and inner_condition_1 are true
    else:
        # Do something if outer is true, but inner_condition_1 is false
else:
    if inner_condition_2:
        # Do something if outer is false but inner_condition_2 is true
    else:
        # Do something if both outer and inner_condition_2 are false
```

- **Linear Nested "if-else":** This is a sequential series of nested conditions, one inside the other, without separate branches.

```
if condition1:
    if condition2:
        if condition3:
            print("All conditions are true.")
        else:
            print("Condition1 and Condition2 are true, but Condition3 is false.")
    else:
        print("Condition1 is true, but Condition2 is false.")
else:
    print("Condition1 is false.")
```

3. Detailed Explanation of Each Type

3A. Non-Linear Nested "if-else"

In a non-linear structure, different paths are taken based on conditions. This leads to a tree-like structure where each block depends on the result of the preceding condition.

Example:

```
age = 25
if age > 18:
    if age > 21:
        print("You are eligible to drive and drink.")
    else:
        print("You are eligible to drive but not drink.")
else:
    if age == 18:
        print("You just became eligible to vote!")
    else:
        print("You are underage.")
```

Explanation:

- The outer if checks whether age > 18.
- The inner if checks if the age is also greater than 21.
- If the outer condition is false, the program checks if the person just turned 18 or is underage.

3B. Linear Nested "if-else"

A linear structure is more straightforward and often used when each condition depends on the previous one being true.

Example:

```
score = 85
if score > 60:
    if score > 70:
        if score > 80:
            print("Grade: A")
        else:
            print("Grade: B")
    else:
        print("Grade: C")
else:
    print("Grade: F")
```

Explanation:

- The outer if checks if the score is greater than 60.
- The next level checks if the score is greater than 70.
- The final level checks if the score exceeds 80.

4. Example with Multiple "if-elif-else" Constructs

```
# Nested "if-elif-else" inside another "if-elif-else"
time = 13  # 24-hour format

if time < 12:
   print("Good Morning!")
else:
   if time < 17:
      print("Good Afternoon!")
   elif time < 20:
      print("Good Evening!")
   else:
      print("Good Night!")
```

In this case:

- The outer if-else block checks whether the time is before or after 12 (noon).
- The inner block then checks if it's afternoon, evening, or night.

Conclusion:

- **Non-Linear Nested "if-else"** allows branching in different directions based on the conditions.
- **Linear Nested "if-else"** proceeds sequentially, testing one condition after another.
- Nesting "if-else" statements helps handle complex decision-making scenarios by allowing multiple conditions to be evaluated efficiently. However, deeply nested code can become harder to read and maintain. It’s best to refactor complex logic with functions or alternative structures like "elif" when possible.

5. Non-Linear Nested "if-else" Example 1

In this example, we'll determine the grade based on both marks and attendance.

```
marks = 85
attendance = 80  # Percentage

if marks >= 50:
    if attendance >= 75:
        if marks >= 90:
            print("Excellent! You passed with an A+ grade.")
        elif marks >= 80:
            print("Well done! You passed with an A grade.")
        else:
            print("Good! You passed with a B grade.")
    else:
        if attendance >= 60:
            print("You passed, but your attendance is low.")
        else:
            print("You passed, but your attendance is too low for any honors.")
else:
    if marks >= 40:
        print("You failed, but you can retake the exam.")
    else:
        print("Unfortunately, you failed badly and will need to repeat the course.")
```

5A. Explanation

- **Outer if** checks if the marks >= 50 (passing mark).
    - **First inner if** checks if attendance is 75% or higher.
        - Based on the marks, we assign the grade.
    - **Else** block handles cases where attendance is lower than 75%.

- **Else** block (for failing marks) further checks if the student is eligible for a retake based on the mark being greater than 40.

6. Non-Linear Nested "if-else" Example 2

In this example, we check the weather conditions and suggest clothing accordingly, factoring in both temperature and wind speed.

```
temperature = 15  # degrees Celsius
wind_speed = 20   # km/h

if temperature > 20:
    if wind_speed < 10:
        print("It's a warm and calm day. Wear light clothes.")
    else:
        print("It's warm but windy. Consider wearing a light jacket.")
else:
    if wind_speed < 10:
        if temperature > 10:
            print("It's cool but calm. A sweater should be enough.")
        else:
            print("It's cold but calm. Wear a coat.")
    else:
        if temperature > 10:
            print("It's cool and windy. Wear a jacket.")
        else:
            print("It's cold and windy. Bundle up with a warm coat.")
```

6A. Explanation

- **Outer** if checks if the temperature is above 20°C.
- **First inner** if checks wind speed.
- **Else** block checks for cooler temperatures and adjusts the recommendation based on wind speed and temperature.

7. Linear Nested "if-else" Example 1

This example categorizes a person based on their age and experience level.

```
age = 32
experience = 6  # Years of experience

if age > 18:
    if experience >= 10:
        if age >= 40:
            print("You are a senior professional.")
        else:
            print("You are a mid-level professional with good experience.")
    else:
        if experience >= 5:
            print("You are a mid-level professional.")
        else:
            print("You are a junior professional.")
else:
    print("You are too young to be a professional.")
```

7A. Explanation

- **Outer** if checks if the person is an adult (age > 18).
- **Next inner** if checks if they have more than 10 years of experience.
    - Another level of checks categorizes them as "senior" or "mid-level" based on age.
- **Else** block handles those with less experience and further classifies them as junior or mid-level.

8. Linear Nested "if-else" Example 2

This example classifies a vehicle based on weight, engine power, and emissions.

```
weight = 2500  # in kilograms
engine_power = 180  # in horsepower
emissions = 100  # in grams of CO2 per kilometer

if weight > 2000:
    if engine_power > 150:
        if emissions < 120:
            print("This is an eco-friendly heavy vehicle.")
        else:
            print("This is a powerful heavy vehicle but not eco-friendly.")
    else:
        if emissions < 120:
            print("This is an eco-friendly heavy vehicle with moderate power.")
        else:
                print("This is a heavy vehicle with moderate power and not eco-friendly.")
else:
    print("This is a light vehicle.")
```

8A. Explanation

- **Outer** if checks if the vehicle is heavy (weight > 2000 kg).
- **Next inner** if checks engine power.
    - Further classification is done based on emissions.
- **Else** block handles light vehicles directly.

9. Comparison Between Non-Linear and Linear Structures

9A. Non-Linear Example Thought Process

In the **Non-Linear** nested structures, decisions diverge based on multiple independent conditions. For instance, in the weather example, the wind speed and temperature lead to separate branches within both the warmer and cooler categories. It's like taking different paths based on each condition's outcome.

9B. Linear Example Thought Process

In the **Linear** structure, decisions are evaluated one after the other in sequence, with each step dependent on the result of the previous one. For instance, in the vehicle example, the checks flow from heavy to powerful to eco-friendly in a step-by-step manner.

Summary

- **Non-Linear Nested if-else**: Conditions diverge, leading to different paths.
- **Linear Nested if-else**: Each condition builds on the previous one, leading to sequential decisions.

Both structures are useful for different scenarios, but deeply nested conditions in either case can become harder to manage. Using proper formatting and comments can help maintain readability.

If-Elif-Else Statement (Multi-Way Decision) In Python

In Python, the if-elif-else statement allows for multiple possible conditions to be checked one by one. This kind of structure is called a "multi-way decision." It works by first checking the condition in the if statement, then successively checking conditions in the elif statements if the first condition was false, and finally, if no previous condition was true, executing the code block in the else statement (if it is provided).

1. Key Components

- **if**: Used to check the first condition. If this condition is True, the block of code under this statement is executed.
- **elif**: Stands for "else if." It allows you to check multiple conditions if the initial if condition is false.
- **else**: Executes a block of code when none of the above conditions are true. Unlike if or elif, the else statement does not check any condition.

1A. Detailed Breakdown

- **if-elif-else statement (Multi-way Decision)**: The if-elif-else construct is useful when there are multiple conditions and only one of the blocks of code should be executed based on which condition is True.
- If the condition in the if statement is False, Python moves to the next elif condition and so on. If none of the conditions are True, the block under the else statement is executed.

1B. Syntax

```
if condition1:
    # Code block 1: Executes if condition1 is true
elif condition2:
    # Code block 2: Executes if condition1 is false and condition2 is true
elif condition3:
    # Code block 3: Executes if both condition1 and condition2 are false, and condition3 is true
else:
    # Code block 4: Executes if all previous conditions are false
```

2. Example 1: Multi-Way Decision - Positive, Negative, Or Zero Number Check

This program checks if a number entered by the user is positive, negative, or zero.

```
# Taking user input
number = float(input("Enter a number: "))

if number > 0:
   print("The number is positive.")
elif number < 0:
   print("The number is negative.")
else:
   print("The number is zero.")
```

- If the user inputs a number greater than 0, the program will output "The number is positive."
- If the user inputs a number less than 0, the program will output "The number is negative."
- If the user inputs 0, the program will output "The number is zero."

2A. Explanation

- The if checks if the number is greater than 0.
- The elif checks if the number is less than 0 if the if condition is false.
- If both if and elif are false, the else block is executed, meaning the number must be 0.

3. Example 2: Multi-way Decision – No else Clause

You don't always need to include an else clause. If no else is provided, nothing will happen if none of the conditions are true.

```
# Taking user input
age = int(input("Enter your age: "))

if age < 18:
   print("You are a minor.")
elif 18 <= age <= 65:
   print("You are an adult.")
elif age > 65:
   print("You are a senior citizen.")
```

3A. Explanation

- Here, three conditions are checked:
  - If age is less than 18, it prints "You are a minor."
  - If age is between 18 and 65, it prints "You are an adult."
  - If age is greater than 65, it prints "You are a senior citizen."

- There is no else statement. If none of the if or elif conditions are satisfied, nothing is printed.

4. Key Differences Between if-elif-else And if-else

- In if-else, there is only **one conditional check**. If the condition is true, the code inside the if block executes; otherwise, the code in the else block runs.
- In if-elif-else, **multiple conditional checks** happen. Each elif allows you to evaluate additional conditions when the previous conditions are false.

4A. Example With Only if-else

- This program checks if the number is positive. If not, it outputs that the number is "Not a positive number." There is no consideration for other possibilities like zero or negative numbers.

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

if number > 0:
   print("Positive number.")
else:
   print("Not a positive number.")
```

- **Explanation**:
  - The program checks if number > 0. If this is True, it prints "Positive number."
  - If the number is not positive (i.e., it's 0 or negative), the program will output "Not a positive number."

- **Limitations of if-else:**
  - This if-else structure doesn't consider possibilities such as the number being 0, so it lumps zero with negative numbers. To handle multiple conditions (positive, negative, or zero), an if-elif-else structure is more appropriate.

4B. Example With if-elif-else: Handling Positive, Negative, And Zero

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

if number > 0:
   print("Positive number.")
elif number == 0:
   print("The number is zero.")
else:
   print("Negative number.")
```

- **Explanation**:
  - The first if checks whether the number is greater than 0. If true, the program prints "Positive number."
  - If the number is not positive, the elif checks if the number is equal to 0. If so, it prints "The number is zero."
  - Finally, if both the previous conditions are False, the else block is executed, indicating that the number must be negative.

- **Output (Based On User Input)**
  - If the user enters 5, the output will be:

```
Positive number.
```

  - If the user enters 0, the output will be:

```
The number is zero.
```

  - If the user enters -3, the output will be:

```
Negative number.
```

4C. if-elif Chain Without else

Sometimes, you might want to evaluate multiple conditions but don't need a catch-all else case. In that scenario, the else block can be omitted.

- **Example Without else**

```
temperature = int(input("Enter the temperature: "))

if temperature < 0:
   print("It's freezing cold.")
elif temperature < 15:
   print("It's cold.")
elif temperature < 30:
   print("It's warm.")
```

- **Explanation**:
  - The program checks the temperature in stages. If the temperature is below 0, it prints "It's freezing cold."
  - If the temperature is between 0 and 15, it prints "It's cold."
  - If the temperature is between 15 and 30, it prints "It's warm."
  - There is **no else** block here, so if none of these conditions are met (i.e., the temperature is 30 or above), the program simply does nothing beyond printing "All Done".

- **Output Examples:**
  - Input: -5

```
It's freezing cold.
```

  - Input: 10

```
It's cold.
```

  - Input: 25

```
It's warm.
```

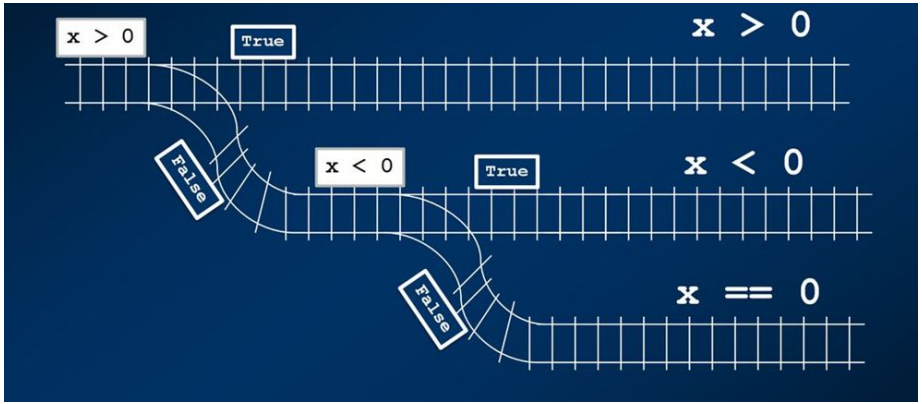
If the input is 35, there will be no output because none of the conditions (if or elif) are met and there is no else block to handle this case.

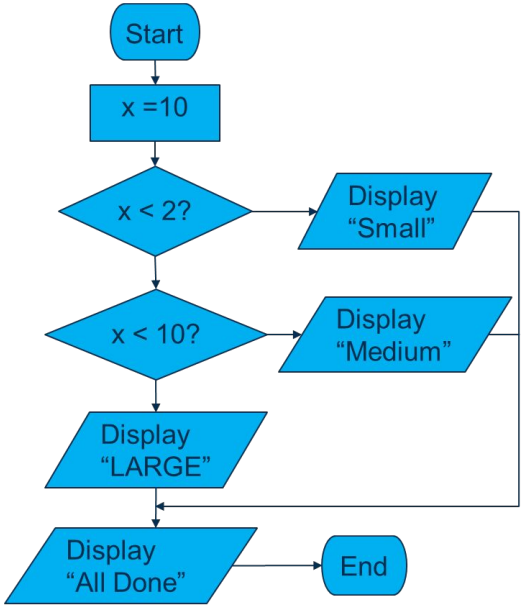
Conclusion

- **if-else**: Use this when you only need to check a single condition and provide an alternative action if that condition is not met.
- **if-elif-else**: Use this when you need to evaluate multiple conditions in sequence. The elif statements act as additional condition checks, while the else serves as a fallback if none of the conditions are True.
- **No else**: You can omit the else block when you don't need a fallback option for cases where none of the conditions are satisfied.
- **if statement**: Checks the first condition.
- **elif statements**: Check additional conditions if the previous ones are False.
- **else statement**: Executes only if none of the previous conditions are True. It acts as a default case.

By understanding how to effectively use if-elif-else, you can make your programs more flexible and handle more complex decision-making logic.

5. Detailed Comparison: With else vs Without else





5A. With else

- The else clause serves as a catch-all for any case where none of the conditions in the if or elif blocks are True.
- If none of the conditions match, the else block is always executed.

- **Example**:

```
x = 10
if x < 2:
   print('Small')
elif x < 10:
   print('Medium')
else:
   print('LARGE')
print('All Done')
```

- **Output**

```
LARGE
All Done
```

Here, since none of the conditions in if or elif are true, the else block runs, printing "LARGE".

5B. Without else

- If there is no else clause, the program does nothing if none of the if or elif conditions are True. The rest of the code continues to execute after the conditional block.
- This is useful when you don't need any action to be taken for certain cases.

- **Example**:

```
x = 5
if x < 2:
   print('Small')
elif x < 10:
   print('Medium')

print('All Done')
```

- **Output**

```
Medium
All Done
```

Here, the program prints "Medium" because x = 5 satisfies the condition x < 10. No else block is needed to handle other cases.

Examining The if-elif-else Structure In Python

The if-elif-else structure allows a program to evaluate multiple conditions one by one, in a sequence. This is particularly useful when we want to test a series of conditions and execute different code blocks depending on which condition is True. Python evaluates the conditions in order, from the first if to the last elif, and finally the else (if it's included). Once a True condition is found, the corresponding code block is executed, and the rest of the conditions are ignored.

1. Key Components

- **if clause**: Starts the conditional evaluation.
- **elif clauses**: Each elif stands for "else if" and checks another condition when the previous one is false. You can have **as many elif clauses as you need**.
- **else clause**: A catch-all for when none of the previous conditions are true. This part is optional.

1A. Important Notes

- Only one block of code inside the if, elif, or else statement will be executed.
- As soon as a True condition is found, the rest of the conditions are skipped.

1B. Understanding The Structure

In the **if-elif-else** structure:

- You begin with an if clause to check the first condition.
- If the condition is false, you proceed to evaluate the next conditions using elif.
- The else clause at the end is a fallback for when none of the if or elif conditions are true.

1C. Example Syntax

```
if condition1:
    # Execute this block if condition1 is true
elif condition2:
    # Execute this block if condition1 is false and condition2 is true
elif condition3:
    # Execute this block if both condition1 and condition2 are false, and condition3 is true
else:
    # Execute this block if none of the above conditions are true
```

1D. Example 1: Basic if-elif-else Structure

```
x = 7

if x < 2:
   print("x is less than 2")
elif x < 10:
   print("x is between 2 and 9")
else:
   print("x is 10 or greater")
```

Explanation:

- The first if statement checks if x < 2. In this case, x is 7, so this condition is false.
- The program then moves to the elif statement and checks if x < 10. Since x = 7, this condition is true, so it prints "x is between 2 and 9."
- The program doesn't check the else block because it already found a True condition in the elif statement.

1E. Example 2: Grading System Using if-elif-else

```
score = 85

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

Explanation:

- The program starts by checking if score >= 90. Since score = 85, this condition is false.
- It then checks elif score >= 80. Since 85 >= 80, this condition is true, and it prints "Grade: B."
- The program doesn’t evaluate the remaining elif statements because one of the conditions has already been satisfied.

1F. Example 3: Age Group Classification Using Multiple Conditions

```
age = 25

if age < 13:
   print("Child")
elif age < 18:
   print("Teenager")
elif age < 65:
   print("Adult")
else:
   print("Senior")
```

Explanation:

- The program checks the age in stages.
    - if age < 13: Checks if the person is a child. This is false since age = 25.
    - elif age < 18: Checks if the person is a teenager. This is also false.
    - elif age < 65: This condition evaluates to true because 25 is less than 65, so it prints "Adult."
- The else block would only execute if none of the conditions were true, which happens if the age is 65 or older (e.g., it would print "Senior").

1G. Example 4: Temperature Check Using Multiple Elif Clauses

```
temperature = 35

if temperature < 0:
   print("It's freezing cold!")
elif temperature < 15:
   print("It's cold.")
elif temperature < 30:
   print("It's warm.")
elif temperature < 40:
   print("It's hot!")
else:
   print("It's extremely hot!")
```

Explanation:

- The temperature is checked step by step:
    - The if checks if the temperature is less than 0 (freezing), but 35 is greater than 0.
    - The elif statements evaluate whether it's cold, warm, or hot. Since 35 is less than 40, it prints "It's hot!" and the remaining conditions are skipped.

2. Notes On if-elif-else Structure

- **Order of Conditions Matters**: Conditions are evaluated in the order they are written. If a condition earlier in the list is satisfied, Python will not check the later conditions. For example, in the temperature example, if you reversed the order of conditions, the program might not behave as expected.

- **Multiple Elif Statements**: You can have as many elif statements as needed, allowing for detailed decision trees based on complex conditions.

- **Omitting else**: The else clause is optional. If no else is included and none of the conditions are met, the program will simply move on without executing any of the code blocks.

2A. Example 5: Without else

```
x = 10

if x < 5:
   print("x is less than 5")
elif x == 10:
   print("x is equal to 10")
```

In this case, if x were not 10 or less than 5, the program would simply do nothing and move on without any output. No else block is necessary.

Conclusion

- The if-elif-else structure is a powerful tool for controlling program flow and making decisions based on multiple conditions.
- The if checks the first condition, elif checks the next conditions, and else provides a default action when none of the conditions are satisfied.
- You can add as many elif statements as you need, allowing for fine-grained control over different conditions and outcomes.

By mastering the use of if-elif-else structures, you can build more intelligent programs that adapt to a wide range of inputs.

Blocks In Python

In Python, indentation is critical to how the code runs. Python does not use braces {} like some other programming languages to mark the beginning and end of blocks of code. Instead, it relies on the indentation level of the lines of code. This makes the code cleaner and more readable, but also introduces a strict requirement that blocks of code must be indented properly. If the indentation is wrong, the code will throw an error or not work as expected.

Let's break it down:

1. Concept Of Blocks

- **Block**: A block is a set of statements that belong together logically and are grouped under the same indentation level.
- In Python, blocks of code follow control structures like if, for, while, def, class, and try. Each of these statements ends with a colon : and the following lines of code form the block, which **must be indented**.

Example:

Here, the two lines after the if statement belong to the same block and will only execute if condition is True.

```
if condition:
    # This is a block
    do_something()
    do_something_else()
```

2. Importance Of Indentation

- Indentation is the number of spaces or tabs at the beginning of a line to denote its level within a block.
- Each new block should be indented consistently (either by 4 spaces or 1 tab, but spaces are preferred in Python). When a block ends, you return to the previous indentation level.

2A. Example of Correct Indentation

```
x = 5
if x > 2:
   print('Bigger than 2')
   print('Still bigger')
print('Done with 2')
```

- **Explanation**:
  - `if x > 2:` creates a block.
  - Both `print('Bigger than 2')` and `print('Still bigger')` are part of the block because they are indented at the same level.
  - `print('Done with 2')` is not indented, so it's outside the `if` block. It will execute regardless of whether `x > 2` is true or not.

2B. Example of Incorrect Indentation

```
x = 5
if x > 2:
print('Bigger than 2')
   print('Still bigger')
print('Done with 2')
```

- **Explanation**:
  - The first `print('Bigger than 2')` is not indented properly under the `if` block. Python expects the code block to be indented, and since it's not, this will raise an **IndentationError**.
  - The second `print('Still bigger')` is indented more than the previous line, which also leads to an error because indentation levels within the same block must be consistent.

3. Further Example

3A. Good Indentation

```
x = 10
if x > 5:
    print("x is greater than 5")
    if x > 8:
        print("x is greater than 8")
    print("This is still part of the 'if x > 5' block")
print("This is outside the if block")
```

- **Explanation**
  - The first if checks if x > 5. If true, the block inside it (with two print statements) will run.
  - Inside that block, there's another if x > 8: statement. This also starts its own block, indented further by 4 spaces.
  - After this block ends, the print("This is still part of the 'if x > 5' block") runs because it's still indented to the if x > 5 block.
  - The last print statement is outside both if blocks, so it runs no matter what.

3B. Bad Indentation

```
x = 10
if x > 5:
  print("x is greater than 5")
    if x > 8:
      print("x is greater than 8")
  print("This is still part of the 'if x > 5' block")
print("This is outside the if block")
```

- **Explanation**
  - The indentation is inconsistent here: the first block is indented with 2 spaces, but the second block (inside if x > 8) is indented with 4 spaces.
  - This inconsistency will result in an error.
  - Python requires a **consistent level of indentation** within the same block. Indentation errors like this are common and can be hard to debug without paying close attention.

Conclusion

- Indentation is used to define the structure and flow of your program.
- All the lines in a block must be indented consistently.
- Even though the code might appear correct visually, slight variations in indentation (using tabs or spaces) can break your program.
- Always be careful and consistent when using indentation in Python to avoid errors.

pass-Statement For Empty Blocks

In Python, **empty blocks** (i.e., code blocks without any statements inside them) are **not allowed**. If you try to leave a block empty, Python will raise an error. This is because Python expects some code to be present after a control structure (like if, for, while, etc.) to determine what to do in that block.

1. Empty Blocks Are Not Allowed

If you define a block (for example, under an if statement) but don't provide any code in it, Python will raise an error. Here's an example:

```
height = 130
if height <= 140:
   # This block is supposed to contain code, but it's empty
print("I'm here")
```

In this case, Python will give a **SyntaxError**, because the if statement block doesn't contain any valid code.

2. Using The pass Statement For Empty Blocks

The pass statement is a placeholder that tells Python, "Do nothing here." It is commonly used when you need a block of code that is required syntactically but you haven't decided what to write inside that block yet.

2A. How The pass Statement Works

- The pass statement does nothing at runtime.
- It can be used to avoid errors when an empty block is necessary for syntactical reasons.

2B. Example With The pass Statement

```
height = 130
if height <= 140:
   pass  # This block does nothing, but prevents an error
print("I'm here")
```

Explanation

- The if height <= 140: block expects some code to follow.
- Instead of leaving it empty, we add the pass statement, which tells Python to do nothing and proceed.
- The block now does nothing if height <= 140, and the program moves on to print "I'm here".

3. Where The pass Statement Is Useful

There are situations where you might want to include a block of code that doesn't do anything yet. You might be writing the structure of a program and want to add logic later. The pass statement is useful in these scenarios.

3A. Common Use Cases For The pass Statement

A. Writing code with placeholder functions

When you're writing the skeleton of your code, and you haven't yet implemented a function but want to define its structure.

```
def future_function():
    pass  # Placeholder for future implementation

# Call the function
future_function()
```

- Here, future_function does nothing yet, but using pass lets you define it without raising any errors.

B. Placeholder for classes

You can define a class without implementing its methods yet by using pass.

```
class MyClass:
    pass  # Class is defined but does nothing yet

# Create an instance of the class
obj = MyClass()
```

- This allows you to outline the structure of your program, and fill in the logic later.

C. Writing control structures without logic

You might want to write control structures like if, for, or while, and leave their bodies empty for future work.

```
number = 7
if number % 2 == 0:
    pass  # We will handle the even case later
else:
    print("Odd number")
```

- Here, the if block is waiting for future logic to handle even numbers, but we still want the code to execute the else block for now.

4. Using pass In Loops

You can also use pass inside loops if you want the loop to run but not do anything for certain iterations.

4A. Example Of Using pass In A for Loop

```
for i in range(5):
    if i == 2:
        pass  # Do nothing when i equals 2
    else:
        print(i)
```

- **Explanation**:
    - The loop will print all numbers except when i == 2.
    - When i == 2, the pass statement is executed, which means nothing happens during that iteration.

5. Further Example

5A. Example Without The pass Statement (Raises Error)

```
height = 160
if height > 140:
    # Intentionally left blank (causes error)
print("Completed")
```

- This will raise a SyntaxError because the if block is empty.

5B. Example With The pass Statement (Works Correctly)

```
height = 160
if height > 140:
    pass  # We do nothing here for now
print("Completed")
```

- In this case, the program runs without errors, and "Completed" will be printed.

Conclusion

- **Empty blocks** in Python are not allowed. If you create a block (such as inside an if, for, while, or function), you must provide some code inside it.

- The **pass statement** is a useful placeholder that you can use when you need a syntactically valid block of code, but you don’t have the logic ready yet.

- The pass statement can be used in control structures, loops, function definitions, class definitions, or any other place where you need a placeholder for future code.

Thinking Corner 1

To solve the problem, we need to write a program that checks a score and determines whether the score is passing or failing, based on the following criteria:

- If the score is **less than 50**, the output will be "You failed".
- For **all other scores (50 or higher)**, the output will be "You passed".

This logic can be implemented using an if-else statement in Python. Let's break down the steps:

1. Step-by-Step Explanation

- **Input**: The program needs to accept a score from the user.
- **Condition**:
  - If the score is less than 50, print "You failed".
  - Otherwise, print "You passed".
- **Output**: Display the result of the condition check (whether the user passed or failed).

2. Example Program

```
# Step 1: Get input from the user
score = int(input("Enter your score: "))

# Step 2: Check if the score is less than 50
if score < 50:
   print("You failed")
else:
   print("You passed")
```

2A. Detailed Explanation

- **Input**:
  - The program uses input() to get the user's score. Since input() returns a string, it is converted to an integer using int().
  - Example: If the user enters 45, score will be 45.
- **Condition (if score < 50)**:
  - The if statement checks whether the score is less than 50. If this condition is true, the program will execute the block inside the if statement (i.e., print "You failed").
  - If the condition is false (i.e., the score is 50 or greater), the program will execute the block inside the else statement (i.e., print "You passed").
- **Output**:
  - If the score is less than 50, the output will be "You failed".
  - Otherwise, the output will be "You passed".

3. Further Examples

Example 1

- **Input**: 40
- **Condition**: 40 < 50 is True, so the program will print "You failed".
- **Output**: "You failed"

Code Flow:

```
Enter your score: 40
You failed
```

Example 2

- **Input**: 75
- **Condition**: 75 < 50 is False, so the program will execute the else block and print "You passed".
- **Output**: "You passed"

Code Flow:

```
Enter your score: 75
You passed
```

Example 3

- **Input**: 50
- **Condition**: 50 < 50 is False, so the program will print "You passed".
- **Output**: "You passed"

Code Flow:

```
Enter your score: 50
You passed
```

Edge Cases

- **Negative scores**: If you want to handle negative scores (which may not be valid for an exam), you could add additional conditions, but for now, any negative score would fall under the "failed" category.

Example:

```
Enter your score: -5
You failed
```

Conclusion

- This simple program evaluates a score using an if-else statement.
- The score is classified into two categories: pass (50 or greater) or fail (less than 50).
- It provides user-friendly output based on the result of the condition check.