

Basic Programming with Python

Prepared By:

Professor Dr. Md. Mijanur Rahman

Department of Computer Science & Engineering

Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

www.mijanrahman.com

3

Control Structures in Python

CONTENTS

3.1. Conditional Statements.....	1
3.1.1. if Statement.....	3
3.1.2. if...else Statement.....	4
3.1.3. if...elif Statement.....	6
3.1.4. Nested Conditionals	8
3.1.5. Conditional Operator (Conditional Expression).....	10
3.1.6. Match-Case Statement	11

3.1. CONDITIONAL STATEMENTS

Conditional statements in Python are used to control the flow of execution based on certain conditions. They allow a program to make decisions and execute different blocks of code depending on whether a specified condition evaluates to True or False. Python provides the following conditional statements:

- **if statement:**

The if statement is used to execute a block of code if a specified condition is True. It can be followed by an optional elif (short for "else if") statement and an optional else statement.

3. Control Structures in Python

If the condition provided with the if statement is False, the code block associated with the elif statement (if present) is executed. If none of the conditions are True, the code block associated with the else statement (if present) is executed.

```
x = 10
if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
```

- **Nested if statements:**

if statements can be nested inside other if, elif, or else statements to create more complex decision-making logic.

```
x = 10
if x >= 0:
    if x == 0:
        print("x is zero")
    else:
        print("x is positive")
else:
    print("x is negative")
```

- **Ternary conditional operator (Conditional Expression):**

Python also supports a ternary conditional operator, which provides a shorter way to write conditional expressions.

It has the syntax: expression1 if condition else expression2.

```
x = 10
result = "Positive" if x > 0 else "Non-positive"
print(result)
```

Conditional statements are essential for writing programs that can make decisions and adapt their behavior based on specific conditions. They allow for the execution of different code blocks depending on the state of the program or the values of certain variables.

Example 3.1: Conditional Statements in Python

```
# Prompt the user for input
age = int(input("Enter your age: "))

# Conditional statements
if age < 0:
    print("Invalid age entered. Age cannot be negative.")
elif age < 18:
    print("You are underage.")
elif age < 65:
```

```
print("You are an adult.")
else:
    print("You are a senior citizen.")

# Prompt the user for another input
num = int(input("Enter a number: "))

# Nested conditional statements
if num % 2 == 0:
    if num == 0:
        print("The number is zero.")
    else:
        print("The number is even.")
else:
    print("The number is odd.")

# Conditional expression (ternary operator)
result = "Even" if num % 2 == 0 else "Odd"
print("Result:", result)
```

This Python program utilizes various conditional statements. In this program, the user is prompted to enter their age and a number. The program then uses various conditional statements to determine and display information based on the input. The first set of conditional statements checks the age input and prints corresponding messages based on whether the age is negative, underage, an adult, or a senior citizen. The second set of nested conditional statements checks the number input and determines whether it is zero, even, or odd. The result is then printed using both a conditional expression (ternary operator) and a regular if-else statement.

Output:

A terminal window titled "Terminal" with a close button and a maximize button. The terminal shows the following output:

```
Enter your age: 72
You are a senior citizen.
Enter a number: 8
The number is even.
Result: Even
```

3.1.1. if Statement

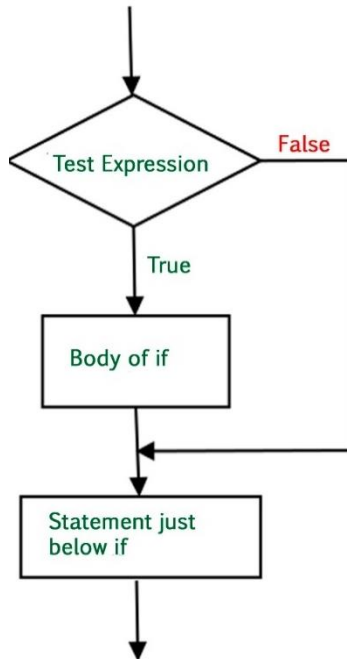
In Python, the if statement is used for conditional execution of code. It allows us to specify a condition, and if that condition evaluates to True, then the code block under the if statement is executed.

The basic syntax of if statement is:

```
if condition:
    # Block of statements
```

If the condition is true, then the block of statements will be executed.

The following is the flowchart of how if statement works in Python:



Example 3.2: Illustration of using if statement in Python.

```
# If Statement
x = 10
y = 5
if x > y:
    print(x, "is greater than", y)

print("Program ended...")
```

In this example, an if statement checks if x is greater than y. If true, it prints “10 greater than 5”; regardless, it then prints “Program ended...” as the next statement, indicating the program flow.

Output:

```
Terminal
10 is greater than 5
Program ended...
```

3.1.2. if...else Statement

In Python, the if...else statement is used for conditional execution of code. It allows us to specify a condition, and if that condition evaluates to True, then the code block under the if statement is executed. Additionally, we can include an else statement to execute a different code block if the condition evaluates to False.

The basic syntax of the if-else statement in Python is given below:

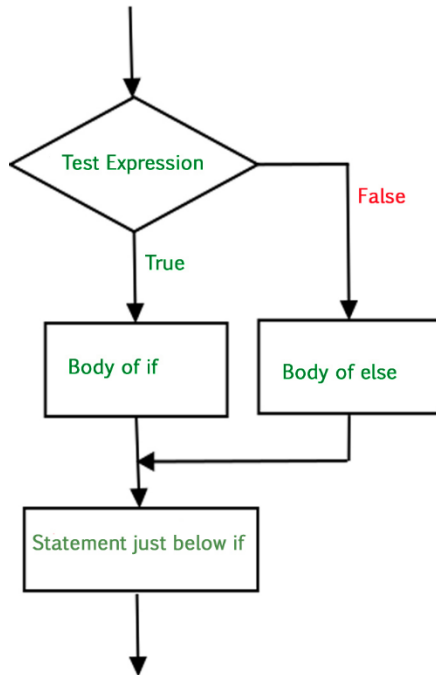
```
if condition:
    # Code block to execute if the condition is True
```

3. Control Structures in Python

```
else:  
    # Code block to execute if the condition is False
```

The condition can be any expression that evaluates to either True or False. If the condition is True, the code block under the if statement is executed. If the condition is False, the code block under the else statement is executed.

The following flowchart illustrates how an if-else statements works in Python:



Example 3.3: Illustration of using if-else statement in Python.

```
# If-else Statement  
x = 5  
y = 10  
if x >= y:  
    print(x, "is greater than", y)  
else:  
    print(x, "is less than", y)  
  
print("Program ended...")
```

In this example, an if-else statement first checks if x is greater than y. If true, it prints “x greater than y”; otherwise, it prints “x is less than y”.

Output:

```
Terminal  
5 is less than 10  
Program ended...
```

3.1.3. if...elif Statement

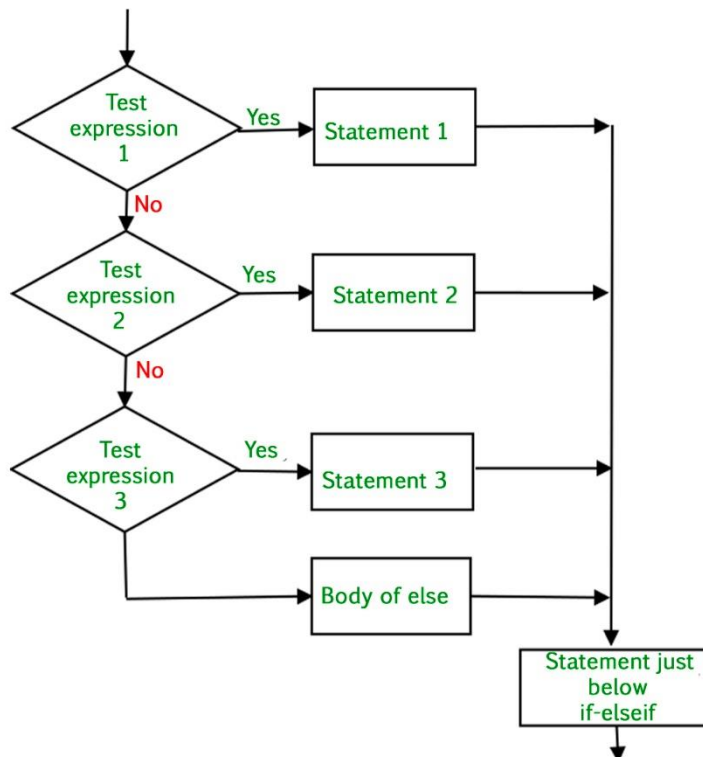
In Python, the elif statement is short for "else if". It is used in conjunction with an if statement to check for multiple conditions sequentially. If the condition preceding an elif evaluates to False, Python checks the condition following the elif. If that condition evaluates to True, the code block associated with that elif statement is executed. If none of the preceding conditions evaluate to True, the else block (if present) is executed.

The syntax for using elif:

```
if condition1:
    # Block1 to execute if condition1 is True
elif condition2:
    # Block2 to execute if condition1 is False and condition2 is True
elif condition3:
    # Block3 to execute if condition1 and condition2 are False and condition3 is True
else:
    # Block to execute if none of the conditions are True
```

Each elif statement is followed by a condition, and the code block associated with the first True condition (if any) is executed. If no conditions are True and an else block is present, the code block under else is executed.

The following flowchart illustrates how if...elif statement works:



Example 3.4: Illustration of if-elif-else statement in Python

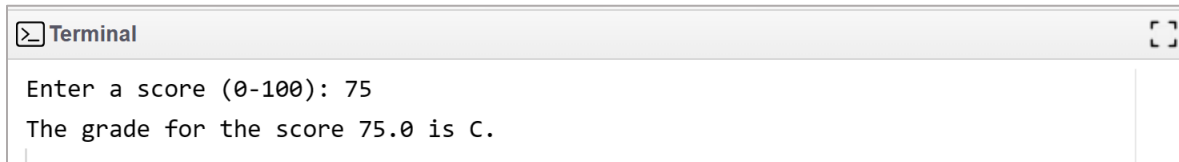
```
# Conditional statement
score = float(input("Enter a score (0-100): "))
```

3. Control Structures in Python

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

print(f"The grade for the score {score} is {grade}.")
```

Output:



```
Terminal
Enter a score (0-100): 75
The grade for the score 75.0 is C.
```

Example 3.5: Solving a quadratic equation using conditional statements.

The following Python script solves a quadratic equation $ax^2 + bx + c = 0$ using conditional statements:

```
# Solve quadratic equation
import math

def Equation(a, b, c):
    discriminant = b**2 - 4*a*c

    if discriminant > 0:
        # Two real and distinct roots
        root1 = (-b + math.sqrt(discriminant)) / (2*a)
        root2 = (-b - math.sqrt(discriminant)) / (2*a)
        print("The roots are:", root1, "and", root2)
    elif discriminant == 0:
        # Two real and equal roots
        root = -b / (2*a)
        print("The root is:", root)
    else:
        # No real roots (complex roots)
        real = -b / (2*a)
        imag = math.sqrt(abs(discriminant)) / (2*a)
        print("The roots are complex:", real, "+", imag, "i and", real, "-", imag, "i")

# Example usage
```

3. Control Structures in Python

```
a = float(input("Enter the coefficient of x^2 (a): "))
b = float(input("Enter the coefficient of x (b): "))
c = float(input("Enter the constant term (c): "))
```

```
Equation(a, b, c)
```

In this script, the Equation() function takes the coefficients a, b, and c of the quadratic equation as input parameters. The discriminant b^2-4ac is calculated to determine the nature of the roots. Nested if-elif-else statements are used to handle different cases based on the value of the discriminant:

- If the discriminant is positive, there are two real and distinct roots.
- If the discriminant is zero, there are two real and equal roots.
- If the discriminant is negative, there are no real roots, and the roots are complex.

The roots are then calculated and printed accordingly.

Output:



```
Terminal
Enter the coefficient of x^2 (a): 1
Enter the coefficient of x (b): 5
Enter the constant term (c): 6
The roots are: -2.0 and -3.0
```

3.1.4. Nested Conditionals

Nested conditionals in Python refer to the practice of placing one conditional statement inside another conditional statement. This allows for more complex decision-making logic where certain conditions need to be evaluated within the context of other conditions.

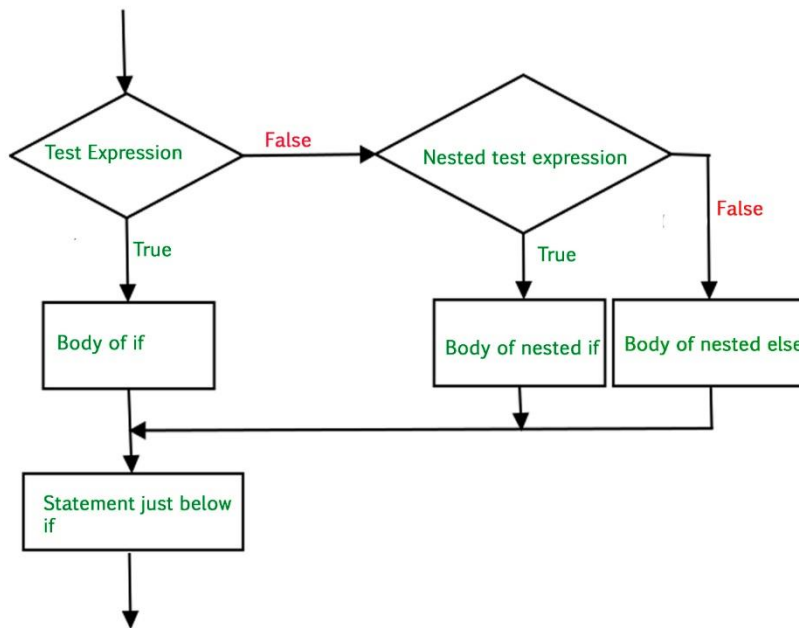
The syntax of nested conditionals in Python is straightforward. We can simply include one if statement within another if, elif, or else block. The basic syntax is given below:

```
if condition1:
    # Code block to execute if condition1 is True

    if condition2:
        # Code block to execute if both condition1 and condition2 are True
    else:
        # Code block to execute if condition1 is True but condition2 is False
elif condition3:
    # Code block to execute if condition1 is False and condition3 is True
else:
    # Code block to execute if none of the above conditions are True
```

The following flowchart illustrates how a nested conditional statement works:

3. Control Structures in Python



The following is an example of nested conditionals in Python:

```
x = 10

if x >= 0:
    if x == 0:
        print("x is zero")
    else:
        print("x is positive")
else:
    print("x is negative")
```

In this example:

- The outer if statement checks if x is greater than or equal to 0.
- If x is greater than or equal to 0, the inner if statement is evaluated.
- Inside the inner if statement, we check if x is exactly equal to 0. If it is, we print "x is zero". Otherwise, we print "x is positive".
- If the outer if statement evaluates to False, indicating that x is negative, the corresponding else block is executed, printing "x is negative".

Nested conditionals are useful when you need to handle multiple levels of conditions or when certain conditions are dependent on the evaluation of other conditions. However, it's important to maintain clarity and readability in your code when using nested conditionals, as excessive nesting can make code difficult to understand and maintain.

Example 3.6: Illustration of nested conditional statement in Python

```
def findLargest(num1, num2, num3):
    if num1 >= num2:
        if num1 >= num3:
```

```
    largest = num1
else:
    largest = num3
else:
    if num2 >= num3:
        largest = num2
    else:
        largest = num3

return largest

# Example usage
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
num3 = float(input("Enter the third number: "))

largest = findLargest(num1, num2, num3)
print(f"The largest number is: {largest}")
```

This Python script finds the largest number among three numbers using nested conditionals. In this script, the `findLargest()` function takes three numbers as input parameters. Nested if-else statements are used inside the function to compare the numbers and determine the largest one. The function returns the largest number among the three. This script prompts the user to enter three numbers, calls the `findLargest()` function with the provided numbers, and prints the largest number.

Output:



```
Terminal
Enter the first number: 20
Enter the second number: 40
Enter the third number: 30
The largest number is: 40.0
```

3.1.5. Conditional Operator (Conditional Expression)

In Python, the Conditional Operator, also known as the Conditional Expression, provides a concise way to express conditional statements in a single line of code. It's a ternary operator, meaning it takes three operands. The syntax of the conditional operator in Python is:

<expression-true> if <condition> else <expression-false>

Explanation of each part:

- <condition>: This is the condition that evaluates to either True or False.
- <expression-true>: This is the value or expression that is returned if the condition is True.
- <expression-false>: This is the value or expression that is returned if the condition is False.

The Conditional Operator is particularly useful when we need to assign a value to a variable or return a value based on a condition. It can make the code more concise and readable, especially for simple conditional checks.

Example 3.7: Illustration of the usage of the Conditional Operator.

```
x = int(input("Enter an integer number: "))  
  
result = "Even" if x % 2 == 0 else "Odd"  
print("The given number is " + result)
```

In this example, the condition `x % 2 == 0` checks if `x` is divisible by 2, hence determining if `x` is even. If the condition is True, the expression "Even" is assigned to the variable `result`. If the condition is False, the expression "Odd" is assigned to the variable `result`. Finally, the value of `result` is printed.

Output:



```
Terminal  
Enter an integer number: 10  
The given number is Even
```

3.1.6. Match-Case Statement

A Python match-case statement takes an expression and compares its value to successive patterns given as one or more case blocks. It resembles the switch-case statement found in some other programming languages, like C or C++. Only the first pattern that matches gets executed. It is also possible to extract components (sequence elements or object attributes) from the value into variables. The basic usage of match-case is to compare a variable against one or more values.

Therefore, the match-case statement allows us to match a value against a series of patterns and execute code based on the first pattern that matches the value. The following is the syntax of match-case statement in Python:

```
match variable_name:  
    case 'pattern 1': statement 1  
    case 'pattern 2': statement 2  
    ...  
    case 'pattern n': statement n
```

An example of how match-case is used in Python is given below:

```
def process_data(data):  
    match data:  
        case 0:  
            print("Data is zero")  
        case 1:  
            print("Data is one")  
        case _:  
            print("Data is something else")  
  
# Example usage  
process_data(0)  
process_data(1)  
process_data(5)
```

3. Control Structures in Python

In this example, the match statement checks the value of data against different patterns specified using the case keyword.

- If data is equal to 0, it executes the code block under case 0.
- If data is equal to 1, it executes the code block under case 1.
- If none of the patterns match (denoted by `_`), it executes the code block under the default case.

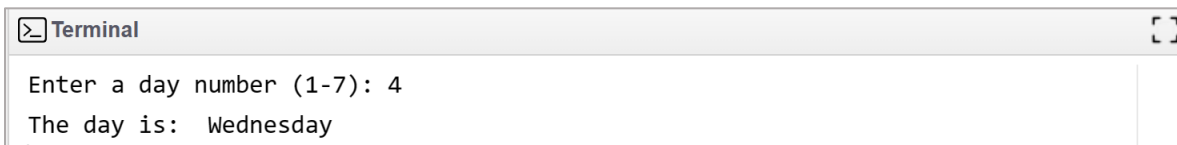
Example 3.8: Illustration of the usage of the Conditional Operator.

```
def weekday(n):
    match n:
        case 1: return "Sunday"
        case 2: return "Monday"
        case 3: return "Tuesday"
        case 4: return "Wednesday"
        case 5: return "Thursday"
        case 6: return "Friday"
        case 7: return "Saturday"
        case _: return "Invalid day number"

# Example Usage
day = int(input("Enter a day number (1-7): "))
print("The day is: ", weekday(day))
```

In this script, the function named `weekday()` receives an integer argument, matches it with all possible weekday number values, and returns the corresponding name of day. The last case statement in the function has `"_"` as the value to compare, will be executed if all other cases are not true.

Output:



```
Terminal
Enter a day number (1-7): 4
The day is: Wednesday
```

