2-1 Journal: Explored Conditional Statements in Programming

Marie-Chantal Foster

Cybersecurity, Southern New Hampshire University

CS-500-10018-M01 Introduction to Programming

Dr. Shadha Tabatabai

November 23, 2025

Explored Conditional Statements in Programming

Conditional statements are fundamental programming constructs in Python that allow a program to make decisions based on specific conditions and execute different blocks of code depending on whether a condition is true or false. They are known as decision-making statements or control flow statements. The conditional statements are broken down into these Boolean expressions: “if, if-else, and elif (if-else if-else chain)“ (Conditional Statements, 2024). Conditional statements provide the logic for how a program responds to different scenarios or inputs.

As we put this to the test in a real-life scenario, we will look at a basic calculator program to help programmers track study hours, manage a budget, or do arithmetic for fun. The calculator uses linear “if-elif-else” conditional statements to determine which operation to use. The user selects the operation, such as addition, subtraction, multiplication, or division, then enters the numbers (Conditional Statements, 2024). The program will evaluate the conditional statements “if, elif, and else” (Diagram on right) to determine which arithmetic function to execute. The “if” statement begins the chain of logic that will split the code into two possible paths, if the “if” is *true,* then the code executes the action, but if the “if” is *false* it will continue to go to the next “elif” block and continue until it reaches the “elif” statement operation until it is true (Jayne, 2019 & Conditional Statements, 2024). The example the user chose was a multiplication symbol (*) (as shown in the Diagram below). However, the same logic would apply: if the user selects division

```
operation = input("Enter operation (+, -, *, /): ")
num1 = float(input("Enter number 1: "))
num2 = float(input("Enter number 2: "))

if operation == "+":
    result = num1 + num2
elif operation == "-":
    result = num1 - num2
elif operation == "*":
    result = num1 * num2
elif operation == "/":
    if num2 != 0:
        result = num1 / num2
    else:
        result = None
        print("Error: cannot divide by zero")
else:
    result = None
    print("Invalid operation")

if result is not None:
    print(f"result: {result:.3f} hours to study python!")
```

```
welcome to the python calculator y'all!
mc, choose an operation: +, -, *, /
Enter operation (+, -, *, /): *
Enter number 1: 40
Enter number 2: 8
result: 320.000 hours to study python!
mcmindmachine@MacBookPro Practice python %
```

and the second number is zero, an error message would occur (Diagram below). This basic calculator example shows limitations: it assumes numeric input and cannot handle complex expressions (Conditional Statements, 2024). Although it has limitations, it demonstrates the logic it can apply to the conditions specified based on user input.

```
mc, choose an operation: +, -, *, /
Enter operation (+, -, *, /): /
Enter number 1: 20
Enter number 2: 0
Error: cannot divide by zero
mcmindmachine@MacBookPro Practice python %
```

That said, poorly designed conditional statements can affect *Code Readability.* For instance, deep nesting would make the logic hard to follow because of an "if" statement nested within another "if or else" block (Wassberg, 2020). *Maintenance* is challenging when updating, and debugging can be difficult when citations are scattered or redundant. Therefore, applying the *DRY* principle (Don't Repeat Yourself) would be the best practice (Wassberg, 2020). *Performance* can slow down conditional execution if, for instance, the algorithm is inefficient, such as using sets instead of lists or avoiding nested loops. *Testing complexity* of the conditional can be affected if the functions are too long, which can make them hard to read and follow the logic as they go in and out of "if" statements and loops (Wassberg, 2020). However, a good rule of thumb is to keep the number below 20 lines if possible.

As far as implementing the solution using Python code, the conditional statements of "if, elif, and else" are used in the calculator logic. This will help control the flow based on user input and execute different blocks of code based on specific condition evaluations that return either *true* or *false.* The "if" statement checks the initial condition, then goes to the next in line, "elif", which is a short version of the "else if" statement, and continues to check the conditions only if

the preceding “if” or “elif” statements are *false*. Lastly, the “else” statement will catch the remaining cases in which the prior conditions were not *true* (Wassberg, 2020).

The advantage of a well-structured “if” condition is that it provides clarity with clear and concise syntax, straightforwardness, and supports basic decision-making logic. It can also validate the user input. It can handle multiple conditions when using “if-elif-else” and reduce any nested “if-else” blocks (Conditional Statements, 2024). That said, the challenge is that it can check only one condition at a time until it reaches the chosen operation. The conditional statements can be lengthy and wordy in complex scenarios. Also, if the conditional statements are out of order, it can lead to unexpected outcomes, so the order of the conditions does matter (Conditional Statements, 2024). These advantages and challenges underscore the importance of carefully designing conditionals for effective code.

In conclusion, conditional statements are a foundation for adding control to the flow of a program. A basic calculator program demonstrates how the “if, elif, and else” structure can perform real-world arithmetic logic in a simple control flow of functions, but the challenges of unordered statements can lead to invalid operations. However, a well-structured code can demonstrate readable decision-making algorithms that provide the foundation for the confidence needed to perform functions, solve complex problems, and write reliable, scalable code.

References

Conditional Statements in Programming | Definition, Types, Best Practices. (2024, September 18). Geeksforgeeks. https://www.geeksforgeeks.org/dsa/conditional-statements-in-programming/

Jayne, B. (2019). *Python quick study guide*. BarCharts Publishing.

Wassberg, J. (2020). *Computer programming for absolute beginners*. Packt Publishing Ltd.