2-1 Journal: Explore Conditional Statements in Programming

Justice Williamson

Southern New Hampshire University

CS-500: Introduction to Programming

Instructor: Ronak Nouri

September 14, 2025

Introduction

Programs need conditional statements to execute decisions which leads to creating functional software applications. The journal entry demonstrates how conditionals function in a personal budgeting application. The paper presents the decision structure followed by Python code implementation of if/elif/else statements and examines the advantages and difficulties of creating effective conditional logic.

Real-World Scenario: Personal Budgeting App

The budgeting application determines user performance in each spending category while checking their ability to reach their monthly savings targets. The program begins by checking user input validity before it compares spending amounts to category restrictions and displays dashboard indicators to users. The main conditions in this system involve numerical value assessments between user expenses and category restrictions and warning threshold evaluations at 90% and above and verification of valid numeric data entry. The system requires users to track their spending across defined time periods because it accumulates data from each month. The system faces certain restrictions during its operation. Users sometimes enter incorrect data into the system while the established warning levels might not work effectively for all users' spending patterns. The system faces problems when multiple rules apply to the same category because it produces conflicting results when the decision sequence is not properly arranged.

The implementation of poorly designed conditionals leads to multiple problems in software development. The complexity of conditional statements increases dramatically when they contain multiple nested levels which makes them difficult to understand. The system becomes harder to maintain while basic modifications become dangerous to perform. The system performance suffers when the program executes identical checks multiple times across different execution paths. The testing process becomes more complicated because developers need to run the entire logic sequence to detect bugs before users encounter them.

Python Implementation Approach

The decision-making process should follow a structured sequence of if statements followed by elif statements and then else statements. The program should begin by checking user input through number conversion and negative value rejection and blank data handling. The system should calculate the percentage of category utilization after processing user data. The system should execute its checks in a specific order starting with utilization above 100% to trigger 'over budget' followed by utilization between 90% and 100% to trigger 'near limit' and finally utilization below 90% to trigger 'on track.' The most critical factor in this process is the correct sequence of checks because the most stringent condition needs to run first to prevent multiple branches from activating. The program should maintain short conditional branches through the use of helper functions which include classify_category_status(utilization) and is_valid_amount(x). The system reduces duplicate code and maintains single decision responsibility for each conditional statement. The application should store all threshold values including NEAR_LIMIT = 0.9 in a centralized location to prevent code modifications when policy changes occur.

Advantages

The application provides immediate and easy-to-understand feedback to users through its well-designed conditional statements. The system enables me to provide personalized guidance through warnings and badges and tips which match the user's current circumstances. The system maintains its ability to handle new information through simple additions of new branches.

Challenges

The main difficulties in this process involve dealing with special cases and preventing multiple rules from interfering with each other. The system produces conflicting results when different conditions exist in an unordered or duplicated sequence. The system will prevent contradictory results by performing input validation first and then arranging conditions from most detailed to least detailed and using early exit mechanisms for invalid data and conducting tests for all possible paths.

Conclusion

Software programs depend on conditional statements to perform their decision-making functions. The implementation of reliable and extendable logic becomes possible through input validation and proper branch organization and complete path testing in the system.