**Basic Programming: Lessons Learned**

Andrew Barnes

Colorado State University Global

23SD-ITS320-1

Dr. Ray Fernandez

8/6/2023

**Basic Programming: Lessons Learned**

Programming demands a structured learning approach to grasp its foundational concepts effectively. Through a week-by-week journey into Python, I reiterate the lessons I've become grateful to have learned. Beginning with a basic understanding of programming, the narrative progresses into the depths of data handling, control structures, functions, object-oriented programming, and modular programming. As you read through, you'll see the world of Python programming I now reside in.

**Introduction to Computer Programming**

In week one, I learned programming is writing instructions for a computer to perform specific tasks using programming languages. As Codecademy (n.d.) compares, “ If you’ve ever cooked using a recipe before, you can think of yourself as the computer and the recipe’s author as a programmer.” It involves creating algorithms and code that direct the computer to solve problems. We started to review how Pythong functions using the Python Interactive Interpreter. This command-line interface enables real-time execution of Python code, providing feedback for testing and running basic operations. When making small scripts, I gained an understanding of functions like input() and print(), allowing the creation of interactive programs that communicate with users through the terminal.

**Understanding Variables & Data Types**

Our second week was about how Python objects and variables have distinct roles in programming. While variables store data and assign names to values, objects are data structures that hold data and the methods to operate on that data. Variables reference objects, and multiple variables can reference the same object. I learned that strings are objects representing sequences of characters, and I can use various string methods to manipulate and process them. Numeric data types in Python can perform mathematical operations using numeric operators. Also, lists are a container that allows storing multiple items in a single variable. Finally, dictionaries enable the storage of key-value pairs and are useful for organizing and accessing data efficiently. Python programs can create and manipulate dictionaries to manage data in a structured manner.

**Understanding Python Decision Control Structure**

In week three, I learned that Python offers a range of decision control statements, including if, if-else, if-elif-else, and nested if statements. Among these, the if-else statement is most common, formatted as "if condition: statement(s) else: statement(s)." Python's relational operators, "==", "!=", "<", ">", "<=", and ">=", assess the relationships between operands and return True or False. Combining multiple relational operators with logical operators, such as "and" and "or," forms compound relational operators. For example, "x > 5 and x < 10" evaluates to True only if both conditions are met. Using compound relational operators, I could craft applications that make decisions based on intricate conditions and optimize program flow.

**Python Repetition**

Learning about loops was one of my favorite things because I could see how software keeps running until the user does what they need. There are three types of loops. While each loop does similar things, each type's syntax and condition checking differ slightly (GeeksforGeeks, 2023). I implemented each type of loop into some code effectively during week four. While loops execute a code block repeatedly as long as a condition is True, and for loops iterate over a sequence of elements using the "for" keyword. Counting loops are a type of for loop that repeats a task a specified number of times using the range() function. Python also supports nested loops, allowing efficient data processing across multiple dimensions, like nested lists or matrices. Using loops simplifies repetitive tasks and data manipulation.

**Python Functions & Strings**

Week five was a game-changer as well. Functions are constructed using the "def" keyword followed by the function name and parentheses containing optional parameters. I understand how data can be passed to functions as arguments within the parentheses. Functions can return results using the "return" statement, providing an output back to the caller. Using a combination of functions and loops with different data types was a massive step forward in my computer science journey.

We also learned how to slice strings in Python. You would use square brackets and specify the start and end indices, creating a substring. Python string methods are built-in functions that manipulate strings, such as "upper()" to convert a string to uppercase and "replace()" to substitute substrings. Splitting strings can be achieved using the "split()" method, which separates a string into a list of substrings based on a specified delimiter. Conversely, joining strings can be accomplished using the "join()" method, merging list elements into a single string with a specified delimiter. These are the incredible techniques I learned for string manipulation.

**Python Lists, Dictionaries, & Classes**

Python offers various list methods that facilitate the manipulation and modification of lists. Examples include "append()" to add elements at the end, "remove()" to remove a specific element, and "sort()" to sort elements in ascending order. Week six showed how to implement some of the mentioned methods to interact with the list object. We also did some looping over Python lists that can be achieved using "for" loops, where each element in the list is accessed in sequence. Alternatively, list comprehensions provide a concise way to create lists using a single line of code. Then we played with dictionaries. As mentioned in a prior paragraph, Python dictionaries are key-value pairs, allowing efficient data retrieval and storage. Looping over dictionaries can be accomplished using "for" loops, where keys or values are accessed, or using the "items()" method to access both keys and values simultaneously. Eventually, we learned about Python classes. These classes are user-defined blueprints for objects, encapsulating data and behavior. Class constructors, defined using the "\_\_init\_\_()" method, initialize objects with specified attributes. Class interfaces define the methods and properties that objects of the class will have access to. Python also supports operator overloading, allowing custom definitions for operators like "+", "-", and "==" to work with class objects, enabling more intuitive interactions with custom objects.

**Python Exception Handling**

Week seven had its sights on exception handling. An exception is when an error spawns from syntactically correct code (Klundert, n.d.). Python exceptions are managed using the "try" and "except" blocks. By enclosing a potentially problematic code within a "try" block, developers can catch errors and handle them gracefully in the corresponding "except" block. Custom and multiple exception handlers can be defined to handle specific errors differently. The "raise" keyword can manually raise exceptions, allowing programmers to generate custom exception messages and specify when to trigger them. Exceptions can also be created within functions to communicate errors to the caller. Python also provides the "finally" block, allowing developers to specify code that executes regardless of whether an exception occurs. This ensures that essential cleanup or finalization tasks are performed, enhancing the robustness of exception handling in Python.

**Modular Programming In Python**

Last but not least, we undertook modular programming. Modular programming in Python involves breaking a program into smaller, self-contained modules, each responsible for specific functionality. These modules can be developed and tested independently, making code maintenance and collaboration more manageable. Importing functions from Python modules allows code reusability, reducing project redundancy. Developers can import specific functions using the "import" statement, gaining access to their capabilities within their code. Python modules can also be executed as scripts by including executable code at the module's top level. This allows direct execution of the module and provides a way to run standalone scripts or perform tests. Python packages are directories that contain multiple modules and an additional "init.py" file, indicating that the directory is a package. Developers create a hierarchical structure by organizing related modules into packages for better project organization and grouping.

**Conclusion**

Every week a new layer of knowledge was presented, from understanding the fundamentals of writing computer instructions to dive into aspects like data structures, control flows, and modular programming. This knowledge deepened my comprehension of Python and laid a solid foundation for further exploration in the vast ocean of computer programming. As I reflect upon the lessons learned, I am reminded of the limitless possibilities that programming offers and the doors it opens for innovation and problem-solving. With these foundational skills in place, I can tread deeper into the more advanced terrains of the coding universe.

**References**

Codecademy. (n.d.). What is Programming? https://www.codecademy.com/article/what-is-programming

GeeksforGeeks. (2023). Loops in Python. https://www.geeksforgeeks.org/loops-in-python/

Klundert, Said van de. (n.d.). Python Exceptions: An Introduction. *Real Python*. https://realpython.com/python-exceptions/