Dhruv Rana

Professor Robila

CSIT313\_01FA23

October 17, 2023

**Homework 2**

**A1.**

* Java and Python are both case-sensitive languages that treat uppercase and lowercase characters differently. This means that both in Java and Python, the variable “height” is different from “Height”. Python variables cannot be Python reserved or keywords, but can contain lowercase characters, uppercase characters, numbers, and underscores, but no other special characters. Python variables can begin with a letter or underscore character but not a number. Java variables can contain both uppercase and lowercase characters, numbers, underscores, and dollar signs but cannot contain Java reserved or keywords or other special characters. Java variables can begin with a letter, underscore character, or the dollar sign, but not a number.
* In both Java and Python, certain words have special meanings within the language known as keywords, and certain words that cannot be used as identifiers, known as reserved words. In Java, int, double, float, boolean are keywords but also reserved words used to declare an integer, double, float, and boolean data types respectively and cannot be used as identifiers. While in Python int, float, and boolean are not reserved words but keywords the language recognizes and can be used as identifiers.
* In Java, the variables are statically typed. The variable data type must be explicitly declared which determines the range of data it can hold. In Python, variable type is dynamically typed, meaning it is not necessary to declare the variable type as the data type of the variable is decided at the runtime of the program depending on the value given.

**A2.**

* If the JavaScript program was interpreted using static-scoping rules, the value of x displayed in the function sub1() is 5. This is because in static-scoping, the variable of a function refers to its top-level environment and is determined by the structure of the program written, where the variable was written and defined. This means that for his JavaScript program variable x is declared but not assigned a value and in sub1(), the variable x is not assigned a value within it, so it takes the global value of the variable which was assigned as x = 5. When function sub1() is called the compiler takes the global value of x, which is 5.
* If the JavaScript program was interpreted using dynamic-scoping rules, the value of x displayed in the function sub1() is 10. This is because in dynamic scoping the value of a variable is determined at the run-time of the program and by the most recent assignment and call. The value of x is in function sub1() determined by the environment in which it was called and since sub2() has a local variable x assigned a value of 10, and after which sub1() is called from within sub2(), the value of x in sub1() is 10.

**A3.**

1. Point 1:
   1. visible variables:
      1. **a - from Definition 1** because a is still visible and no new variable a is defined within the outer while loop that gives Definition 2
      2. **b - from Definition 2** because the innermost definition of b takes precedence over the outer scope which is outside of the outer loop
      3. **c - from Definition 2** because the innermost definition of c takes precedence over the outer scope which is outside of the outer loop
      4. **d - from Definition 2** because the innermost definition of d takes precedence over the outer scope which is outside of the outer loop
2. Point 2:
   1. Visible variables:
      1. **a - from Definition 1** because a is still visible and no new variable a is defined within the inner while loop that gives Definition 3
      2. **b - from Definition 2** because b is still visible and no new variable b is defined within the inner while loop that gives Definition 3
      3. **c - from Definition 3** because the innermost definition of c takes precedence over the outer scope which is outside the inner while loop
      4. **d - from Definition 3** because the innermost definition of d takes precedence over the outer scope which is outside the inner while loop
      5. **e - from Definition 3** because e is defined within the scope of the inner while loop that gives Definition 3
3. Point 3:
   1. Visible variables:
      1. **a - from Definition 1** because a is still visible and no new variable a is defined within the outer while loop that gives Definition 2 and outside the inner while loop.
      2. **b - from Definition 2** because the innermost definition of b takes precedence over the outer scope which is outside of the outer loop
      3. **c - from Definition 2** because the innermost definition of c takes precedence over the outer scope which is outside of the outer loop
      4. **d - from Definition 2** because the innermost definition of d takes precedence over the outer scope which is outside of the outer loop
4. Point 4:
   1. Visible variables
      1. **a - from Definition 1** because Point 4 is outside the outer while loop and in the scope of Definition 1
      2. **b - from Definition 1** because Point 4 is outside the outer while loop and in the scope of Definition 1
      3. **c - from Definition 1** because Point 4 is outside the outer while loop and in the scope of Definition 1

**B.**

The scope of a variable in a language is the range of statements over which it is visible and scope rules of a language determine how references to variable names impact variable value and function. Scoping is important to programming as it keeps variables in different parts of the program different from one another. Scoping can be static or dynamic. In static scoping, the value and function of a variable in a program is determined by the lexical structure of the program or the environment in which it was declared. In static scoping, a variable always refers to the top-level environment in which it was declared in, which is related to the program text. In dynamic scoping, the value and function of a variable in a program is determined by the most recent assigned value and location of the variable. In dynamic scoping, the programmer must always consider multiple scoping contexts for the variables defined. Both static and dynamic scoping are methods used by programming languages to determine the range of statements over which the variables are visible in a program.

The purpose of this assignment was to write test programs in C, C++, and Java to determine the scope of a variable declared in a for-statement, and if it was visible outside the body of the for-loop. For C, C++, and Java, the variable ‘number’ declared within the for-statement is visible within the body of the for-loop as the print statement ran as intended and printed the value of ‘number’ after each iteration. When the variable is referenced outside the for-loop it results in a compilation error. The scope of the variable is within the for-loop. This means that C, C++, and Java are statically scoped languages. In C and C++, the compiler outputted the following error message, ‘use of undeclared identifier 'number'’. In Java, the compiler outputted the following error message, ‘cannot find symbol’. What I really liked about C and C++ in regards to their error messages is that they specifically stated why the error occurred, an undeclared identifier, as ‘number’ was declared in the for-statement and not outside the for-loop.

This assignment was very helpful in understanding the importance of scopes in a programming language. To be honest, I never put much thought into the scope of my variables when programming. I am able to have two functions with the same variable names but they mean something entirely different, such a feature makes programming much easier and convenient. I prefer static scoping compared to dynamic scoping, as static scoping makes it easier to read the code. I can come to understand the values and the type of my variables at any given moment, unlike with dynamic scoping. Though static scoping also has disadvantages in that it allows for greater access to variables unnecessarily, and as a program evolves and becomes larger, the structure of the program shifts as local variables can become global variables.

I feel these types of programming assignments are very necessary and important. They expose students to programming languages they may have never seen or used before or have had little experience and exposure to. Having assignments that require coding in different languages helps to better understand the fundamentals of all if not many programming languages. Having this experience helps to build confidence and encourages learning new programming languages besides the ones you may already know. Additionally, this also helps to determine which programming languages to use to tackle certain programming challenges, developing problem solving skills. I think it would be an interesting assignment if you had a programming task to compare and contrast an older version of a language with its current version or compare and contrast programming in an “archaic” language to an “modern relevant” language to determine how languages have evolved.

While programming with C, C++, and Java, I have come to understand that I am still not comfortable with programming in C++ as I have very little experience with it. While C++ is object-oriented similar to Java, it also handles memory allocation and pointers similar to C, both languages I am very well versed in, C++ is a difficult language for me to grasp. I prefer to use Java over C and C++ as while it is more restrictive, it is less error prone and risky compared to C and C++ as Java does not allow for pointers. Pointers allow for direct access to memory and memory management but having direct access to memory and modifying it can lead to overriding memory and information, memory leakages and dangling pointers. Java attempts to solve the memory allocation problem through the use of garbage collection to delete code no longer needed or used. Though C and C++ have their uses especially in system programming. It’s all about picking the right language for the right task.