Name: Dawson Demien

Student ID: 4001332

HC: I have neither given or received, nor have I tolerated others’ use of unauthorized aid.

0) Commenting tools are available in \*\*\*\*\*\*\*every\*\*\*\*\*\*\* language (besides TI Basic, apparently).

**True** or False?

1) Discuss why we care about the difference between compilation and interpretation….

(at least 2 categories of difference, please)

**We care about compiling and interpretation because it determines both the difference in efficiency (how quickly code is read) and how the code is read into the system. With compilation the code is built directly in the computer’s language (assembly) which can result in faster compilations. In interpretive languages the code is interpreted line by line into binary which then causes the code to run slower (less efficiently) than a compiled language.**

2) Explain the minimum two cases that we need for recursion:

(why is that a minimum - not a maximum??)

**The minimum two cases we need for recursion are the end case and the recursive call on itself. The reason why these are the minimum cases is because these two cases need each other. For example, if one only had the end case then the program would only navigate through the program once and not complete its full cycle. Moreover, if one only has the recursive call with no end case then the program would be stuck in a loop and probably end in a depth error.**

3) Describe two major innovations from Lisp….

**Two innovations from Lisp are that it is very good at recursion and it is machine independent.**

4) What does

(car (cdr (cdr '(a b c d))))

Get evaluated as? **c**

….. And briefly describe (in lisp terms) what’s going on:

**What is happening here is that (as usual) lisp is working backwards. Therefore, the first thing that occurs is that lisp takes cdr of a,b,c,d which is the command to grab everything from the list that is not the first element so it grabs b,c,d. Then cdr is called again and grabs c and d. Finally the last call on the list is car which grabs the first element of the list which is c. This is why the answer is c.**

5) Is Python a “pure” scripting language, why or why not?

**Python did start as a scripting language, but it is not a “pure” scripting language. The reason for this is because Python is an interpreted language like any other scripting language, but it can also be compiled. For example, Python compiles python code into bytecode and then interprets that.**

6) What must every programming language ultimately result in/become….? (Otherwise, well, it isn’t useful.)

**Every programming language must become a language that can be built upon. Either built upon itself or have administration built around the community. It must have this because with no improvements to the language, then it will either die off or become increasingly less compatible with most issues a computer scientist may want answered.**

7) List and discuss at least 2 factors that make a language “succeed” (get used) or “fail” (get thrown away/be ignored?)

**One factor that makes a language succeed, or fail, is if it has good libraries (or how well those libraires fit to the problem). The reason for this is because libraries can make a big difference when tackling a problem in code. Good libraires would consist of libraries that add extra tools or math to make calculations, or steps, for a procedure much less intensive. Without such libraries, one could be stuck brute forcing lines of code just to get the same output as the language with libraries that did the same compilation in hundred less lines of code and in less time.**

**Another factor that can make a language get used or not is one’s personal capacity/knowledge on the language. To put simply, if one is tasked with a project to make a program that pulls all the prime numbers in a range of numbers but have a choice between Lisp or Python, then that person will more than likely choose the language they are more comfortable in. In most cases, the obvious answer to creating such a program would be Python for the fact that it is more widely understood and known to most student computer scientists. Therefore, programming languages get used or cast aside also by how well the programmer, or company, at hand understands the language.**

8) Describe at least two big needs/area/challenges that the scripting languages tend to be the solution for… as in, where are they found?

**One area a scripting language tends to be a solution for is execute programs on a web server. This would include controlling the interaction used on a webpage and displaying the correct results to the user. Another area scripting languages are the solution for is automating tasks at the application level. In this case, scripting languages would be used to automate long processes used within an application or platform to further streamline one’s long and repetitive tasks.**

9) (doubled…) Besides python, what is one \*other\* example of a scripting language. What is it used for?

**Javascript is another example of a scripting language. Javascript can be used for Web Development. It helps perform some of the “thinking” for the website. Therefore, it allows the webpage to be interactive.**

10) (also a double) Describe at least 2 things that can vary between languages (and sometimes even implementations of the same language?!) that it is a good idea to check before (or when) using it?

**Two things that can vary between languages and one would want to check before using the language would be whether the language is procedural or functional, and whether the language is a strong or weak programming language. Procedural languages tend to be languages that are object oriented and can modify variables and act off these objects, whereas functional languages are not object oriented and the language returns values without modifying the original variable. Strong programming languages tend to have more rules as to how variables are assigned, what values a function return, and function calling. Whereas a weak programming language has less rules referring to the above situations and can even make implicit type conversions at runtime.**

11) (double) What makes a language “portable” (or not)?

**What makes a language portable is whether it can be ran on multiple different kinds of computers or not. If the language can be ran on many different kinds of computers then it would be considered portable; otherwise, if the language can’t then it would not be considered a portable language. For a language to be portable then even at compilation the machine code from the language would be understood the same on different types of computers. This is what determines whether a language is portable.**

12) (also worth double) Discuss - *with some detail, please* - at least one example of where a functional language gets used --- and how the various attributes, as in the criteria we use to describe/categorize/label/discuss a language - are relevant to that use.

**Functional languages are used when one wants to perform different operations on the same set of data. The reason for this is because functional programming uses immutable data which means it returns values without modifying the original set of data. This is important to note because this allows one to perform computation on data and return those values all without modifying the original data set. Furthermore, functional languages are stateless which goes hand and hand with using immutable data, because being stateless means that previous events in the code will not affect the current output of the code. This piggy backs more off the idea of not wanting to change the data set at hand. Functional languages are all about performing tasks on an immutable data set to provide outcomes without changing the data.**

13) Explain the difference between imperative and declarative languages.

**The difference between an imperative language and a declarative language is the decision point. An imperative language asks how the task is being completed; whereas a declarative language asks what is being accomplished. Declarative languages tend to have functional logic and imperative languages tend to have procedural logic.**

14) Which of the languages you’ve used to date (not just in class) do you like the most. Why… in programming language terms (as in the ideas we’ve talked about) to the extent possible.

**The language I enjoy using the most is SQL. The reason for this is because I feel as though SQL is a straightforward programming language. It is in fact a declarative language, so the main question it asks is what is being accomplished. Therefore, I like the idea of simply querying a dataset to receive information. The way in which data is accessed and received is simple and the code that allows these procedures is easy to understand. The best part of SQL is that it uses immutable data and is stateless just like other functional languages. The reason why this is so nice is because it makes accessing data so easy and stress free knowing that you will not be altering the data directly. The language has friendly (for the most part) code, syntax, and capabilities that helps to make work easier for the coder. Finally, when reading the code one can have a general idea of what it is doing since the language is relatively uncomplicated.**

15) Why is COBOL (and various other ancient languages) still in use? What do these reasons or circumstances tell us?

**The reason other ancient languages are still in use are because there are either migration issues/ operational issues, a great cost in replacing the legacy system, clever extensions/ wrappers preventing their replacement, or there are physical issues preventing the company/ person from replacing the legacy system. These reasons tell us that legacy systems can be a problem for companies and that sometimes there may be no way to replace a system so some people may have to learn them to a certain extent. Furthermore, it projects that the people with knowledge on these kinds of ancient languages can be a rare but great find for a company since the knowledge on the languages may not be as common as newer languages like Java. Finally, these circumstances tell us that legacy systems are expensive and very hard to upkeep.**

Pointless language humor:

