**Claude Segla - Reflective report**

throughout the course, I had learned a great deal about programming languages and how they differ from each over, this has definitely influenced the way I think about and design code. One of the first language we covered was ruby, ruby is a reflective, dynamic typed language meaning types are not checked until runtime as opposed to compile time. Ruby is strongly typed meaning the compiler is strict about how types are interchanged it differs from JavaScript one of the example of weak typing is string concatenation in JavaScript, you can concatenate string and an integer “wat!” we can never do that in ruby. Ruby also uses a unique system called duck typing which allows us to pass objects to method in another class as its less concerned about what class the object is in, as long as the method accepts objects. Ruby is interpreted but there are other compilers that let you compile it such as the Rubinious compiler. Ruby creators must’ve been inspired by Smalltalk as everything is an object. Another interpreted language is Smalltalk, it was an influential and important language as it was the first language to introduce unit testing and it also brought OOP into the limelight Smalltalk is deeply OO even the classes are objects. Smalltalk has very few reserved words. Objects needs a way to talk with other objects in order to progress through the program Smalltalk does this elegantly with message passing, newer languages like java, c have adopted this slightly in the way of method calls. JavaScript is one of the modern interpreted important languages, originally developed by Netscape for their browser. Despite the name it has nothing in common with java except the c like syntax. JavaScript is Object oriented so you would expect it to have class based inheritance but no JavaScript does things a bit different. JavaScript is prototype based, every object has a prototype so you can do inheritance that way, each prototype also has an objected linked it to it, null has no prototype so if the prototype is null then we know that’s the end. Whilst JavaScript is object oriented its easy for it to be used in a non OO way such as functional style with arrow functions (lambda). JavaScript lives in the browser. Most modern languages are very stable yet have some technical glitches JavaScript being one of them it can be a weird language at times as some behaviours are undefined such as out of bound of arrays and all number is the only numeric data type it uses 64 bit Float, as result ‘0.1 + 0.2 !== 3’ .

One language I practically find interesting and admire is Lisp. Lisp is different from the languages I described earlier, it follows the functional programming paradigm. One thing I noticed about lisp is that the data structures are immutable I was especially as most OO languages as mutable. I thought this was inefficient as we have to create a new copy every time we make an edit. Lisp uses a great deal of brackets which takes time to get used to, but as I got used to lisp syntax I started to enjoy the language and can see how it makes it easy to express lambdas. Another thing I would like to mention is that since lisp is a functional language, it uses lambda calculus to express ourselves. Lambda allows us to write expression from a mathematical view, we write this expression in a list. Operators are always on the left because of reverse polish notation, lisp supports currying by nesting lists. Lisp always evaluates every list therefore It doesn’t really make any distinction from code and data. One variant of Lisp is clojure which runs on the Java virtual machine one benefit of this is that it can use existing java libraries. In pure functional like lisp. functions must be pure meaning variables are only in the function and not global. This prevents side effects as there is no external factors such a variable being changed outside. Haskell is another functional language the syntax is more readable yet still makes a great use of lambda calculus. Just like clojure functions do not alter states as they only return values leading to no side effects. What is particularly interesting about Haskell is its type system. Haskell is strict about its types especially being careful about I/O as those use monads. Since functions must remain pure and not be effected by outside states we use monads, also when we define a function signature the function must take and return the type declared in the signature sometimes this can get tricky as I had experienced in the portfolio exercise. The type system is lazy so types are not evaluated till necessary, Haskell makes use of type inference whereby the types are guessed by the compiler based on its operator. Just like clojure Haskell makes use of high order function whereby we chain functions together to create an output much like pipes in UNIX. Haskell is usually compiled using the ghc compiler or ghci repl. Functional languages differ greatly from imperative languages such as javascript, in a imperative language you usually define what you want to do and how to do it whereas in a functional language we just have functions somewhat like a blackbox you tell it what you want to do and not how its done. Back to when I mentioned functional languages like Haskell and lisp use immutable data structures, whilst it creates lots of duplication of object meaning more overhead, its not always a bad thing as one benefit is it improves concurrency as there’s no mutable conditions. Haskell, clojure, ruby can all be run in a read evaluate print loop on the command line