PPL 1/15

Reagan Shirk

January 15, 2020

Historical Overview

- Machine instructions are in fact considered a programmming language but who in their right mind actually wants to write in it
  - Each computer had its own language- its own instruction set
- Assembly was created to alleviate some of the difficulty (but still sucks)
- There was a need for a language that could be used on any machine to assist with portability and reuse
  - Fortran, Lisp, and Algol were born
  - Apparently Fortran is still popular...? Guess I should learn it
    - It was created to describe mathematics and formulas in machine instructions
- Languages were implemented as compilers and interpreters: they embody how programming languages are ran/interpreted (I think that's what he said, I forgot by the time I started typing)
  - Compilers: translate from "high level" (what we write in) to "low level" (machine instruction)
    - Translate sorce language to target language
    - They can perform source to source translation
    - There are also decompilers which translate from "low level" to "high level"
  - Interpreters: similar to compilers but they execute commands
    - Interpreter will not translate the executable/binary representation of the code
- Beginning of Programming Languages
  - Programming languages evolve and die just like we do
  - The 60's and 70's brought structured programming
    - Explicit control with loops and ifs and switches
    - GoTo based control flow (whatever tf that means)
    - Fortran, Cobol, Basic, etc
  - The 80's brought a nested block structure which was a precursor to object oriented programming
    - Algol, Pascal, Ada
  - The 90's brought Smalltalk, C++, Eiffel, Java (lots of object oriented languages)
  - The late 90's and 00's brought scripting languages like PHP, Python, and Ruby
- Evolution of Programming Languages
  - Changes are made to programming languages as the need arrises
    - Productivity:
      - machine code inspired development of assembly
      - need fore shorter, more compact code like array slices
      - reusability and maintainability inspired structures, functions, and objects
    - New areas and domains:
      - Classical scientific computations
      - Web development
      - Circuit design
      - Quantum computing

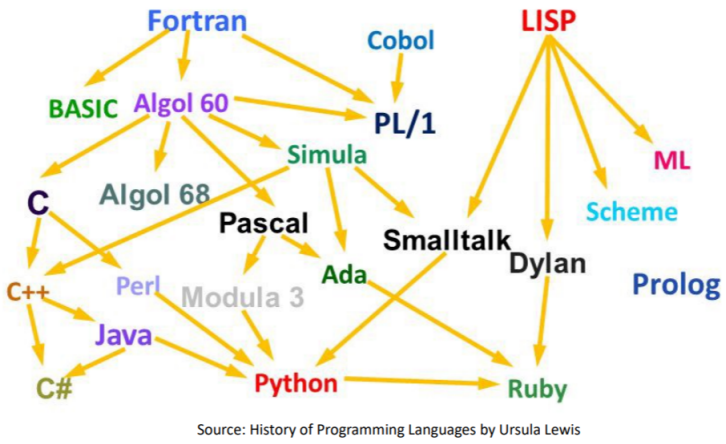

Figure 1: A tree representing the relationships between programming languages

Classification

- One of the (many) classifications for programming languages is declarative vs imperative languages
  - Imperative:
    - Focuses on how, we want to have a lot of control over the language
    - Similar to how we write pseudocode
    - Von Neumann: C, Ada, Pascall, Fortran
    - Object oriented: Smalltalk, Java, C++
    - Scripting: Perl, Python, PHP, Javascript, Bash
  - Declarative
    - Focuses on what to do rather than how to do it
    - Compiler and interpreter have freedom of implementation- the developer has a little less control
    - Gap between pseudocode and implementation
    - Functional: Lisp, Scheme, Haskell
    - Dataflow: Id, Val, Dot
    - Logic/Contraint based: Prolog, Spreadsheet, SQL

von Neumann Model

- This is a classical computation model
- It conists of
  - Main memory
  - CPU: Control unit, arithmetic/logic unit, machine registers
  - I/O mechanisms

Programming langauge spectrum

- von Neumann Languages
  - Are based on statements, expressions, and assignments
    - x := 0.5 * y + z
  - Allow side effects and variable modification

  - Fortran, Ada, C
- Logic and Contraint based languages
  - They provide mechanisms to specify relationships and constraints
  - They are goal directed and rule based
  - Prolog, Excel, SQL
- Functional Languages
  - Inspired in the Church Lambda-Calculus thesisi (circa 1930)
  - Computational model, recursive definition of functions
  - The program itself is considered a function from inputs to outputs
  - The data can not be mutated; the variables are pure with nothing to modify
  - Lisp, Scheme, Caml, Ocaml, Scala
- Dataflow Languages
  - Graph like, nodes connected by edges
  - Describes the flow of information
  - Inherently parallel
- Object Oriented Languages
  - Data and operations bundled together
  - Pure OO languages to not support "free functions"
  - C++, Java, C#, Scala, Ocaml
- Scripting Langauges
  - Languages that "put stuff together" - Professor Martin Kong
  - Developed for very specific reasons such as job control or dynamic web pages
    - Shell and bash (job control), PHP and Javascript (dynamic web pages)
  - Others for a general purpose: PHP, Python, Ruby, Matlab
- Markup Languages
  - Motivated by wanting to "mark up" paper manuscripts
  - Tag based
  - GenCode, Tex/LaTex, Scribe, Markdown, HTML, XML