Things to Know

• Textbook
  • Programming Language Pragmatics, v4
    • by M Scott
      • 17 chapters, 9 covered and not all of those
  Also
  • The Go Programming Language
    • Donovan & Kernighan
  • Learn Functional Programming with Elixir
    • Almeida
Programming Languages

- Why?
  - because it is required for the major
- Why is it required?
  - All PL are “Turing Complete” so at one level it does not matter what language you use
  - Grace Hopper — if you program in a better language you can be more efficient
Class WebSite

https://cs.brynmawr.edu/cs245

• Will have all homework, important dates, etc

• Lecture notes — I will post PDF “notes”. Literally my notes to myself.
  • This will likely be the only powerpoint for the class

• Tests — 2 midterms and a final. All will be open book, open notes, open computer, closed mouth.
  • midterms will be “take where you want, but on a given date”
  • final — similar idea.

• Homeworks — approximately 6 through the semester

• Lab — The first few will be formally in lab room. These will be graded on a “did you hand in something that is at least semi-correct”.

• Class participation — will be assessed on a “readiness to participate” rather than actual participation. Readiness will be assessed with occasional 5 minute quizzes at start of class.
Lab

• Both this week and next week.

• If you have timing issues, not a problem to arrive late.
  • Formally, labs are not due until midnight of the lab date.
  • Think of lab as
    • an office hour in which I am sitting in 231
    • I have given you a fairly simple task.
Goals

• Learn questions to ask, and how to evaluate answers, for choosing language appropriate to problem

• Improve ability to learn new programing languages
  • In my career: Basic, PL/1, Pascal, (rascal, spss, sas), C, Lisp, Prolog, Visual-C, Perl, Python, Visual-Basic, Java, SQL, Objective-C, PHP, Javascript, Kotlin, Go, Elixir. (and probably a dozen others)

• Increase ways in which you can express and implement programs

• Understand why and wherefore of “obscure” language features
First Generation

- Machine language -- literally working with 0 and 1

Second Generation

- Assembly language -- write commands that are directly supported by CPU

Third Generation

- Most PLs that you will ever work with: C, Java, ...

Fourth Generation

- Giving instructions to VM that specify what, not how: SQL, R(?)

Fifth Generation

- AI stuff: Prolog, Lisp(?)
# A Really Brief Genealogy of PLs

<table>
<thead>
<tr>
<th>First Compiler/ Interpreter</th>
<th>Language</th>
<th>Lead Designer</th>
<th>Progeny</th>
</tr>
</thead>
<tbody>
<tr>
<td>1952</td>
<td>A-0</td>
<td>Hopper</td>
<td>FLOW-MATIC</td>
</tr>
<tr>
<td>1955</td>
<td>FLOW-MATIC</td>
<td>Hopper</td>
<td>COBOL</td>
</tr>
<tr>
<td>1957</td>
<td>Fortran</td>
<td>Backus</td>
<td>ALGOL (Fortran influences, directly or indirectly, every other language on this page excepting Lisp, COBOL, and APL)</td>
</tr>
<tr>
<td>1958</td>
<td>ALGOL</td>
<td>committee</td>
<td>BCPL, Pascal</td>
</tr>
<tr>
<td>1958</td>
<td>Lisp</td>
<td>McCarthy</td>
<td>Scheme, all functional languages, Ruby</td>
</tr>
<tr>
<td>1959</td>
<td>COBOL</td>
<td>Hopper + committee</td>
<td></td>
</tr>
<tr>
<td>1964</td>
<td>APL</td>
<td>Iverson</td>
<td>(small family of descendants)</td>
</tr>
<tr>
<td>1964</td>
<td>BASIC</td>
<td>Kennedy / Kante</td>
<td>Apple and Microsoft Basics, scripting language in MS Office, Lotus Notes and many others</td>
</tr>
<tr>
<td>1964</td>
<td>PL/I</td>
<td>IBM</td>
<td></td>
</tr>
<tr>
<td>1966</td>
<td>BCL</td>
<td>Richards</td>
<td>B</td>
</tr>
<tr>
<td>1967</td>
<td>Nov</td>
<td>Well</td>
<td>influences many, Ruby</td>
</tr>
<tr>
<td>1969</td>
<td>B</td>
<td>Thompson</td>
<td>C</td>
</tr>
<tr>
<td>1970</td>
<td>Pascal</td>
<td>Wirth</td>
<td>Modula-2</td>
</tr>
<tr>
<td>1972</td>
<td>C</td>
<td>Ritchie</td>
<td>C++, Javascript (and most later languages), Go</td>
</tr>
<tr>
<td>1972</td>
<td>Smalltalk</td>
<td>Kay, Engel, Goldberg</td>
<td>C++, Javascript (via Self), all object languages</td>
</tr>
<tr>
<td>1975</td>
<td>Scheme</td>
<td>Steele and Norstrom</td>
<td>Javascript, all functional languages</td>
</tr>
<tr>
<td>1978</td>
<td>Modula-2</td>
<td>Wirth</td>
<td>Modula-3</td>
</tr>
<tr>
<td>1983</td>
<td>C++</td>
<td>Stroustrup</td>
<td>Java (and numerous others)</td>
</tr>
<tr>
<td>1987</td>
<td>Self</td>
<td>Ungar, Smith</td>
<td>Javascript</td>
</tr>
<tr>
<td>1997</td>
<td>Modula-3</td>
<td>committee</td>
<td>Java, Python</td>
</tr>
<tr>
<td>1991</td>
<td>Python</td>
<td>von Rossum</td>
<td>Ruby</td>
</tr>
<tr>
<td>1995</td>
<td>Java</td>
<td>Gosling</td>
<td>(most later languages)</td>
</tr>
<tr>
<td>1995</td>
<td>JavaScript (and Mocha)</td>
<td>Eich</td>
<td>(and Mocha)</td>
</tr>
<tr>
<td>1995</td>
<td>PHP</td>
<td>Lerdorf</td>
<td></td>
</tr>
<tr>
<td>1995</td>
<td>Ruby</td>
<td>Matsumoto</td>
<td>elixir</td>
</tr>
</tbody>
</table>

http://www.martinrinehart.com/pages/genealogy-programming-languages.html
Java

Why is Java the first language taught at BM

• Plusses
  • large, well-organized libraries
  • clean, consistent syntax
  • Easily available instructional support
  • Widely used

• Minuses
  • OO is big hurdle — have to “talk around it”
  • Comically wordy
... goals were to enable higher extensibility and productivity in the Erlang VM while keeping compatibility with Erlang's ecosystem. The designers wanted to address criticism of other languages in use at Google, but keep their useful characteristics:

- Static typing and run-time efficiency (like C)
- Readability and usability (like Python or JavaScript)
- High-performance networking and multiprocessing

Its designers were primarily motivated by their shared dislike of C++.

Clojure, Erlang, Ruby

C, Oberon-2, Limbo, Active Oberon, communicating sequential processes, Pascal, Oberon, Smalltalk, Newsqueak, Modula-2, Alef, APL, BCPL, Modula, occam

https://en.wikipedia.org/wiki/Go_(programming_language)
Why not ..... 

The complexity of C++ (even more complexity has been added in the new C++), and the resulting impact on productivity, is no longer justified. All the hoops that the C++ programmer had to jump through in order to use a C-compatible language make no sense anymore -- they're just a waste of time and effort.

Go makes much more sense for the class of problems that C++ was originally intended to solve.

Bruce Eckel -- a founding member of the ANSI/ISO C++ standard committee
Hello World

//GO
package main
import "fmt"
func main() {
    fmt.Println("hello geoff!")
}

//Somewhat briefer
package main
func main() {
    println("hello geoff!")
}

#ELIXIR
IO.puts("Hello world")

# Somewhat Longer
defmodule HW do
    @moduledoc ""
    A very minor module to support hello world
    ""
    
    @doc ""
    a basic function to print hello world ...  
    ""
    def hw do
        IO.puts "Hello World 2"
    end

    @doc ""
    As with the previous function, but using a  
    special syntax to go on a single line  
    ""
    def hw2, do: IO.puts("hello World 2b")
    end

    HW.hw
    HW.hw2
Functional and Imperative programming

• Imperative
  • programming by side effect
    • procedures that return nothing (in Java void)
  • lots of variables whose values are set and change frequently

• Functional
  • No variables
    • there are things that look like variables but they are better though of as constants
      • What is the difference between a variable and a constant whose value you can change?
  • Functions always return values, it is why they are executed
  • Functions are only dependent on their arguments
    • Programs can be provably correct (usually of academic interest only)
For next class

• If you could be a programming language, which one would you be
  • Why?
  • Why is that language so named?
    • Do not use: Java, C, Python, Fortran, Cobol, Javascript, Elixir, Go.

• Read
  • Scott 1.1-1.4

• Labor Day Weekend -- Watch (at least the first 30 minutes)
  • "The worst programming language ever"
    • https://www.youtube.com/watch?v=vcFBwt1nu2U
  • On Wednesday Sep 7 -- one statement that you did not understand or thought was really funny