Programming Languages

Syllabus Overview
First Question:
What programming languages have you used?
- Python
- C++
- Java
- Ruby
- JavaScript
- QBASIC
- C#
- Matlab
- PHP
- C
- Assembly
Categories

High-level versus low-level:

assembly \[\xrightarrow{\text{assembler}}\] machine code

high-level \[\xrightarrow{\text{compiler}}\] machine code or assembly
High-Level Languages

- Began in 1950's with Fortran
- First machine-independent solution
- Slow to become popular because compilers were not as good as humans

(Not true now—plus, labor costs more than hardware)
Why so many?

- Evolution: Still very new!
  - Structured programming (using loops instead of go-to's) was only developed in the late 60's.
  - Object orientation was developed in the '80's.

- Personal preference
- Special purposes: Often the choice depends on what you want to do!
  - C is good for low level systems work
  - Prolog is good for logical relationships among data
  - Awk is good for character string manipulation
  - Python and Perl are good scripting tools
Other Issues

- Learning curve
- Ease of use
- Standardization
- Open Source
- Good Compilers
- Economics & patronage
- Inertia
Families of high-level Languages

1. Declarative Languages: Focus is on what the computer should do.

2. Imperative Languages: Focus is on how the computer should do it.

* C++, C, Java, ...
Imperative

Categories:

   - based on computation with variables

B. Scripting languages: bash, awk, php, perl, python, Ruby, etc.
   - subset of von Neumann, but tailored for ease of expression over speed

C. Object-oriented: traced from Simula 67, often related to von Neumann, but object-based
Declarative

Categories & Examples:

A. Functional languages: Lisp, Scheme, ML, Haskell
   - based on recursive definition of functions
     (inspired by lambda calculus)

B. Logic-based: prolog, SQL (?)
   - computation is based on attempts to find values that satisfy specified relationships

C. Data flow: I/O, Val
   - flow of information (tokens) among nodes
Example: Compute the gcd
(Stolen from my ISO lecture)

Set \( u = v \) equal to the numbers.

\[
\begin{align*}
42 & \\
54 & \\
\text{gcd} & \Rightarrow 6
\end{align*}
\]

Reset \( u + v \) to values \( v \) and \( u \), respectively.

Divide \( u \) by \( v \) and let \( r \) be remainder.

\[
\begin{align*}
\text{Is } v &= 0? \\
\text{No} & \\
\text{Yes} & \Rightarrow \text{output } u
\end{align*}
\]
GCD in a functional language

\[ \text{gcd}(a, b) := \begin{cases} a & \text{if } a = b \\ \text{gcd}(b, a-b) & \text{if } a > b \\ \text{gcd}(a, b-a) & \text{if } b > a \end{cases} \]

\[ a = 54, \quad b = 42 \]

\[ \text{gcd}(54, 42) = \text{gcd}(42, 12) = \text{gcd}(12, 30) = \text{gcd}(12, 18) = \text{gcd}(12, 6) \]
GCD in Prolog

gcd(a, b, g) is true if:

- \( a = b = g \)
- \( a > b \) and \( \exists c \) such that \( c = a - b \) and gcd\((c, b, g)\) is true
- \( b > a \) and \( \exists c \) st. \( c = b - a \) and gcd\((c, a, g)\) is true