CSCI 334:
Principles of Programming Languages

Instructor: Dan Barowy

Williams

MINECRAFT
Java

C/assembly

C/C++/C#/assembly

iOS
C/Objective-C

twitter
Scala (orig. Ruby)

Google Maps
JavaScript

T - -Mobile
Erlang

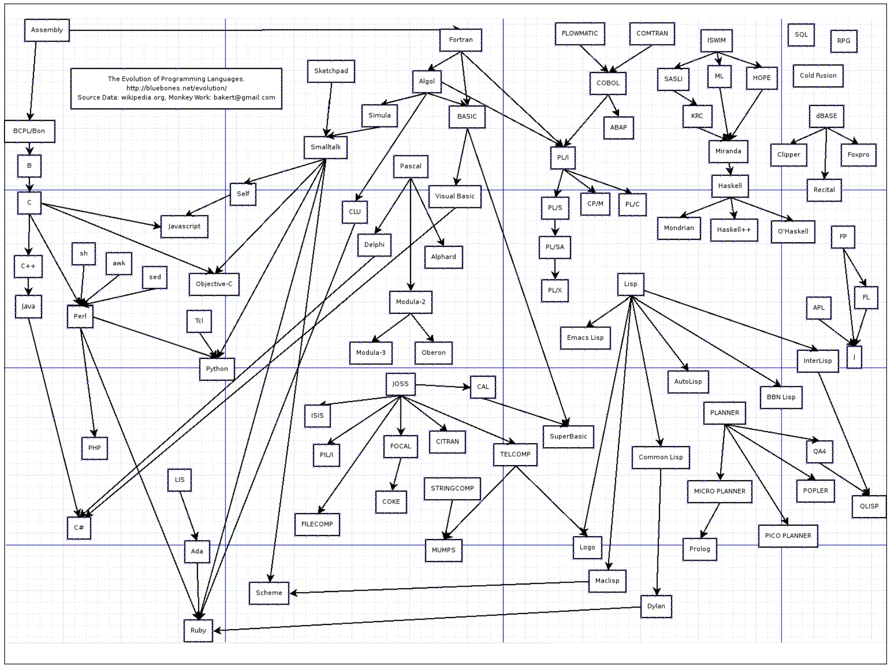
FiveThirtyEight
Python

Why?

Hint: most were motivated by an app.

---

assembly	*raw lang. of machines*
Java	*"write once run anywhere"*
C/C++	*systems programming*
Objective-C	*"better"systems programming*
Ruby	*"happy programmers"*
Scala	*"scalable" language*
Python	*one-off scripting tasks*
Erlang	*safe, fast concurrency*



Course Organization

"programming in the small"

- theoretical foundations
- common elements of languages
- functional vs. imperative languages
- new ways of thinking

Course Organization

"programming in the large"

- modularity
- implementation mechanisms
- object oriented programming
- concurrency
- domain-specific languages

syllabus

Computability

Computability

i.e., what can and cannot be done with a computer

def: a function *f* is **computable** if there is a program *P* that computes *f*.

In other words, for **any** (valid) input *x*, the computation *P(x)* **halts** with output *f(x)*.

Computability

example

valid inputs are **integers**

P(x) is:
f(x) = x + 5

computable?
yes.

Computability

example

valid inputs are **integers**

P(x) is:
f(x) = 5/x

computable?
yes, *partially*.

Total Function

f: $A \to B$ is a subset $f \subseteq A \times B$ subject to

1. for every $a \in A$, there is a $b \in B$ with $\langle a,b \rangle \in f$ **totality**

2. if $\langle a,b \rangle \in f$ and $\langle a,c \rangle \in f$ then $b=c$ **single valued**

e.g,
$f(x) = x + 5$

Partial Function

f: $A \to B$ is a subset $f \subseteq A \times B$ subject to

1. ~~for every $a \in A$, there is a $b \in B$ with $\langle a,b \rangle \in f$ **totality**~~

2. if $\langle a,b \rangle \in f$ and $\langle a,c \rangle \in f$ then $b=c$ **single valued**

e.g,
$f(x) = 5/x$

Exercise

The Halting Problem

Decide whether program P halts on input x.

Given program P and input x,

$$\texttt{Halt(P,x)} = \begin{cases} \text{print } \texttt{"halts"} \text{ if } \texttt{P(x)} \text{ halts} \\ \text{print } \texttt{"does not halt"} \text{ otherwise} \end{cases}$$

How might this work?

Clarifications:

`P(x)` is the output of program `P` run on input `x`.

Assume `x` is a string.

The Halting Problem

Decide whether program P halts on input x.

Given program P and input x,

$$\texttt{Halt(P,x)} = \begin{cases} \text{print } \texttt{"halts"} \text{ if } \texttt{P(x)} \text{ halts} \\ \text{print } \texttt{"does not halt"} \text{ otherwise} \end{cases}$$

How might this work?

Fun fact: it is provably impossible to write `Halt`

The Halting Problem

Proof:

Suppose Q(P,x) is a program that:
returns “halts” if P(x) halts
returns “does not halt” if P(x) does not halt

Construct new program D(P)
D(P) = runs forever if Q(P,P) returns “halts”

D(P) will halt if P(P) runs forever.
D(P) will run forever if P(P) halts.

The Halting Problem

Proof:

What happens when we call D(D)?

recall:
D(P) = runs forever if Q(P,P) returns “halts”

D(D) will halt if D(D) runs forever.
D(D) will run forever if D(D) halts.

This makes (literally) no sense. Contradiction.

Thus the Halting Problem is not computable.

Implications of The Halting Problem

We cannot tell, in general…

… if a program will run forever.

… if a program eventually produces an error.

… if a program will re-read an item in memory.