<table>
<thead>
<tr>
<th>Language</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>“write once run anywhere”</td>
</tr>
<tr>
<td>C/C++/C#/assembly</td>
<td>systems programming</td>
</tr>
<tr>
<td>Objective-C</td>
<td>“better” systems programming</td>
</tr>
<tr>
<td>Ruby</td>
<td>“happy programmers”</td>
</tr>
<tr>
<td>Scala</td>
<td>“scalable” language</td>
</tr>
<tr>
<td>Python</td>
<td>one-off scripting tasks</td>
</tr>
<tr>
<td>Erlang</td>
<td>safe, fast concurrency</td>
</tr>
</tbody>
</table>

### Why?

**Hint:** most were motivated by an app.
Course Organization

“programming in the small”

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

“programming in the large”

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

syllabus
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 \( f(x) \).

**example**

valid inputs are **integers**

\[ P(x) \text{ is:} \]
\[ f(x) = x + 5 \]

computable?
yes.

**example**

valid inputs are **integers**

\[ P(x) \text{ is:} \]
\[ f(x) = 5/x \]

computable?
yes, *partially.*
Total Function

\[ f: A \rightarrow B \text{ is a subset } f \subseteq A \times B \text{ subject to} \]

1. for every \( a \in A \), there is a \( b \in B \) with \( \langle a, b \rangle \in f \) \text{ totality} \\
2. if \( \langle a, b \rangle \in f \) and \( \langle a, c \rangle \in f \) then \( b = c \) \text{ single valued} \\

\[ \text{e.g., } \] \\
\[ f(x) = x + 5 \]

Partial Function

\[ f: A \rightarrow B \text{ is a subset } f \subseteq A \times B \text{ subject to} \]

\[ \text{1. for every } a \in A, \text{ there is a } b \in B \text{ with } \langle a, b \rangle \in f \text{ totality} \]
\[ \text{2. if } \langle a, b \rangle \in f \text{ and } \langle a, c \rangle \in f \text{ then } b = c \text{ single valued} \]

\[ \text{e.g., } \] \\
\[ f(x) = \frac{5}{x} \]

The Halting Problem

Decide whether program \( P \) halts on input \( x \).

Given program \( P \) and input \( x \),

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

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,

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

How might this work?

Fun fact: it is provably impossible to write \text{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.