What programming languages do you know?

Some historically interesting and/or currently visible languages:


Don’t forget things like:

- HTML, PHP, other web page description languages
- SQL, other database query languages
- EXCEL formula language
- application-specific languages
# What languages?

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>![icon] ![icon] ![icon]</td>
<td>100.0</td>
</tr>
<tr>
<td>2.</td>
<td>![icon] ![icon] ![icon]</td>
<td>99.9</td>
</tr>
<tr>
<td>3.</td>
<td>![icon] ![icon] ![icon]</td>
<td>99.6</td>
</tr>
<tr>
<td>4.</td>
<td>![icon] ![icon] ![icon]</td>
<td>95.8</td>
</tr>
<tr>
<td>5.</td>
<td>![icon] ![icon] ![icon]</td>
<td>91.8</td>
</tr>
<tr>
<td>6.</td>
<td>![icon] ![icon] ![icon]</td>
<td>84.7</td>
</tr>
<tr>
<td>7.</td>
<td>![icon] ![icon] ![icon]</td>
<td>84.5</td>
</tr>
<tr>
<td>8.</td>
<td>![icon] ![icon] ![icon]</td>
<td>83.0</td>
</tr>
<tr>
<td>9.</td>
<td>![icon] ![icon] ![icon]</td>
<td>75.3</td>
</tr>
<tr>
<td>10.</td>
<td>![icon] ![icon] ![icon]</td>
<td>72.4</td>
</tr>
<tr>
<td>11.</td>
<td>![icon] ![icon] ![icon]</td>
<td>71.4</td>
</tr>
<tr>
<td>12.</td>
<td>![icon] ![icon] ![icon]</td>
<td>70.9</td>
</tr>
<tr>
<td>13.</td>
<td>![icon] ![icon] ![icon]</td>
<td>67.9</td>
</tr>
<tr>
<td>14.</td>
<td>![icon] ![icon] ![icon]</td>
<td>67.9</td>
</tr>
<tr>
<td>15.</td>
<td>![icon] ![icon] ![icon]</td>
<td>66.9</td>
</tr>
<tr>
<td>16.</td>
<td>![icon] ![icon] ![icon]</td>
<td>63.4</td>
</tr>
<tr>
<td>17.</td>
<td>![icon] ![icon] ![icon]</td>
<td>63.0</td>
</tr>
<tr>
<td>18.</td>
<td>![icon] ![icon] ![icon]</td>
<td>62.4</td>
</tr>
</tbody>
</table>

Source: spectrum.ieee.org/static/interactive-the-top-programming-languages-2015
What languages?
What languages?

Rank of top languages on GitHub.com over time

Source: GitHub.com
What languages?

History of Programming Languages

For more than half of the fifty years computer programmers have been writing code, O'Reilly has provided developers with comprehensive, in-depth technical information. We've kept pace with rapidly changing technologies as new languages have emerged, developed, and matured. Whether you want to learn something new or need answers to tough technical questions, you'll find what you need in O'Reilly books and on the O'Reilly Network.

This timeline includes fifty of the more than 2500 documented programming languages. It is based on an original diagram created by Éric Lévénez (www.levenez.com), augmented with suggestions from O'Reilly authors, friends, and conference attendees.

For information and discussion on this poster, go to www.oreilly.com/go/languageposter.

www.oreilly.com
“Why are coders angry?”

“Programmers carve out a sliver of cognitive territory for themselves and go to conferences, and yet they know their position is vulnerable. They get defensive when they hear someone suggest that Python is better than Ruby, because [insert 500-comment message thread here]. Is the next great wave swelling somewhere, and will it wash away Java when it comes? Will Go conquer Python? Do I need to learn JavaScript to remain profitable? Programmers are often angry because they’re often scared. We are, most of us, stumbling around with only a few candles to guide the way. We can’t always see the whole system, so we need to puzzle it out, bit by bit, in the dark.”

- Paul Ford, “What is Code?”, Bloomberg Businessweek, 6/11/15
Course Goals

☐ Learn fundamental structure of programming languages

☐ Understand key issues in language design and implementation

☐ Become aware of range of available languages and their uses

☐ Learn how to learn a new language

☐ Get a small taste of programming language theory
Course Method

- Organized around key **anatomical features** of PLs
  - Expressions, control flow, functional abstraction, state, types, modules, ...

- Lab exercises mostly involving **implementing interpreters** for “toy” languages

- Roughly following on-line textbook “Programming and Programming Languages” by Krishnamurthi

- Exercises will use **Scala**, a modern language that blends OO and functional paradigms
Course Non-Goals

- Teaching you how to program
- Teaching you how to program in Scala although you will learn something about this!
- Surveying the detailed of lots of languages
- Covering all important programming paradigms e.g. we’ll skip logic programming and concurrent programming
- Learning how real compilers work
“High-level” Programming Languages

Consider a simple (dumb and slightly wrong) algorithm for testing primality.

In Scala, using imperative programming style:

```scala
// return true if n has no divisor in the interval [2,n) 
def isPrime(n:Int) : Boolean = {
  for (d <- 2 until n)
    if (n%d == 0)
      return false;
  true
}
```
“High-level” Programming Languages

In Scala, using a local recursive function:

```scala
// return true if n has no divisor in the interval [2,n) 
def isPrime(n:Int) : Boolean = {
  // return true if n has no divisor in the interval [d,n) 
def noDivFrom (d:Int) : Boolean =
    (d >= n) || ((n % d != 0) && noDivFrom (d+1))
  noDivFrom(2)
}
```
In Intel X86 (32bit) assembler

.globl isprime
.isprime:

    pushl %ebp          ; set up procedure entry
    movl %esp,%ebp      
    pushl %esi          
    pushl %ebx          
    movl 8(%ebp),%ebx   ; fetch arg n from stack
    movl $2,%esi        ; set divisor d := 2
    cmpl %ebx,%esi      ; compare n,d
    jge true            ; jump if d >= n
    loop: movl %ebx,%eax ; set n into ....
           cltd             ; ... dividend register
           idivl %esi       ; divide by d
           testl %edx,%edx ; remainder 0?
           jne next        ; jump if remainder non-0
           xorl %eax,%eax ; set ret value := false(0)
           jmp done
    next: incl %esi      ; increment d
           cmpl %ebx,%esi  ; compare n,d
           jl loop         ; jump if d < n
    true: movl $1,%eax   ; set ret value := true(1)
    done: leal -8(%ebp),%esp ; clean up and exit
           popl %ebx
           popl %esi
           leave
           ret
What make a language “high-level”?

- Complex expressions (arithmetic, logical, ...)
- Structured control (loops, conditionals, cases, ...)
- Composite types (arrays, records, ...)
- Type declarations and type checking
- Multiple data storage classes (global/local/heap/GC?)
- Procedures/functions, with private scope (first class?)
- Non-local control (exceptions, threads, ...)
- Data abstraction (ADTs, modules, objects...)
What does hardware give us?

- Low-level machine instructions
- Control flow based on labels and conditional branches
- Flat memory model
- Explicit locations (e.g. registers) for values and intermediate results of computations
- Explicit memory management (e.g., stacks for procedure local data)
How do we bridge the gap?

Two classic approaches:

- A compiler translates high-level language programs into a lower-level language (e.g. machine code)

- An interpreter is a fixed program that reads in (the representation of) an arbitrary high-level program and executes it

Compilers can generate code that runs faster than interpreted code

Interpreters are quicker and easier to write, maintain and understand
A stack machine is a simple architecture based on a stack of operand values.

Each machine instruction pops its operands from the stack and pushes its result back on the stack.

So instructions are very simple, because there’s no need to specify operand locations.

Often used in abstract machines, such as the Java Virtual Machine (which Scala also uses).

Often compile from high-level language to stack machine byte code which is then interpreted (or perhaps further compiled to machine code).
### Stack machine example(1)

Instruction set for a very simple stack machine

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Stack before</th>
<th>Stack after</th>
<th>Side effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST i</td>
<td>$s_1 \ldots s_n$</td>
<td>$i s_1 \ldots s_n$</td>
<td>-</td>
</tr>
<tr>
<td>LOAD x</td>
<td>$s_1 \ldots s_n$</td>
<td>$\text{Vars}[x] s_1 \ldots s_n$</td>
<td>-</td>
</tr>
<tr>
<td>STORE x</td>
<td>$s_1 \ldots s_n$</td>
<td>$s_2 \ldots s_n$</td>
<td>$\text{Vars}[x] := s_1$</td>
</tr>
<tr>
<td>PLUS</td>
<td>$s_1 s_2 s_3 \ldots s_n$</td>
<td>$(s_1+s_2) s_3 \ldots s_n$</td>
<td>-</td>
</tr>
<tr>
<td>MINUS</td>
<td>$s_1 s_2 s_3 \ldots s_n$</td>
<td>$(s_1-s_2) s_3 \ldots s_n$</td>
<td>-</td>
</tr>
</tbody>
</table>

Here $\text{Vars}[\cdot]$ is an auxiliary memory mapping variables to values.
And here’s a stack machine program corresponding to the simple statement $c = 3 - a + (b - 7)$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST 3</td>
<td>LOAD a</td>
</tr>
<tr>
<td>MINUS</td>
<td>LOAD b</td>
</tr>
<tr>
<td>CONST 7</td>
<td>MINUS</td>
</tr>
<tr>
<td>PLUS</td>
<td>STORE c</td>
</tr>
</tbody>
</table>

Is this code sequence unique?

Observe that high-level expressions are more expressive than machine code.
Other structuring ideas

Paradigms

- Imperative
- Object-oriented
- Functional
- Logic
- Concurrent/Parallel
- Scripting

Design Themes

- Expressiveness
- Efficiency
- Correctness

Scale

- “Programming in the Small”
  - what’s important for $10^2$ lines?
- “Programming in the Large”
  - what’s important for $10^6$ lines?
Course Structure

- Weekly lab assignments
  - Working with interpreters for “toy” languages illustrating important language features
  - Implementation in Scala
  - Using WebLab, a new (and somewhat experimental) platform for testing and submitting assignments
  - Backed up by on-line textbook “Programming and Programming Languages” (PAPL) by Krishnamurthi

- You are **encouraged** to work collaboratively on these assignments (but everyone must submit separately)
Books

First edition is available free on line, and is good enough for us.
Grading

☐ 10% Weekly Labs
☐ 20% Midterm #1 (Oct. 22)
☐ 30% Midterm #2 (Nov. 12)
☐ 40% Final exam (Monday Dec. 7)

☐ The labs don’t count for much directly, but you are unlikely to pass the exams unless you do them.

☐ Again, collaboration on the labs is encouraged, but you must submit individually to get credit (and to learn)
WebLab

- Web-based system for assignments
- Assignments are issued
- You develop solutions in the embedded editor
- You test your solutions against your own tests and against (secret) tests we provide
- You submit your solutions
- Your scores are automatically recorded
- We (usually) publish correct solutions