

Programming Languages: Concepts

(Lectures on High-performance Computing for Economists IV)

Jesús Fernández-Villaverde,¹ Pablo Guerrón,² and David Zarruk Valencia³

March 28, 2019

¹University of Pennsylvania

²Boston College

³ITAM

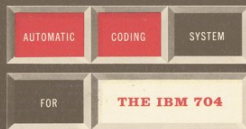
Introduction

Motivation

- Since the invention of Fortran in 1954-1957 to substitute assembly language, hundreds of programming languages have appeared.
- Some more successful than others, some more useful than others.
- Moreover, languages evolve over time (different version of Fortran).
- Different languages are oriented toward certain goals and have different approaches.

PROGRAMMER'S REFERENCE MANUAL

Fortran



Some references

- *Programming Language Pragmatics (4th Edition)*, by Michael L. Scott.
- *Essentials of Programming Languages (3rd Edition)*, by Daniel P. Friedman and Mitchell Wand.
- *Concepts of Programming Languages (11th Edition)*, by Robert W. Sebesta.
- <http://hyperpolyglot.org/>

The basic questions

- Which programming language to learn?
- Which programming language to use in *this* project?
- Do I need to learn a *new* language?

Which programming language? I

- Likely to be a large investment.
- Also, you will probably want to be familiar at least with a couple of them (good mental flexibility) plus \LaTeX .

Alan Perlis

A language that doesn't affect the way you think about programming is not worth knowing.

- There is a good chance you will need to recycle yourself over your career.

Which programming language? II

- Typical problems in economics can be:
 1. CPU-intensive.
 2. Memory-intensive.
- Imply different emphasis.
- Because of time constraints, we will not discuss memory-intensive tools such as Hadoop and Spark.

Classification

Classification

- There is no “best” solution.
- But there are some good tips.
- We can classify programming languages according to different criteria.
- We will pick several criteria that are relevant for economists:
 1. Level.
 2. Domain.
 3. Execution.
 4. Type.
 5. Paradigm

Level

- Levels:
 1. machine code.
 2. Low level: assembly language like NASM (<http://www.nasm.us/>), GAS, or HLA (*The Art of Assembly Language (2nd Edition)*, by Randall Hyde).
 3. High level: like C/C++, Julia, ...
- You can actually mix different levels (C).
- Portability.
- You are unlikely to see low level programming unless you get into the absolute frontier of performance (for instance, with extremely aggressive parallelization).

Fibonacci number

Machine code:

```
8B542408 83FA0077 06B80000 0000C383 FA027706 B8010000 00C353BB  
01000000 B9010000 008D0419 83FA0376 078BD98B C84AEBF1 5BC3
```

Assembler:

```
ib:  mov edx, [esp+8]  cmp edx, 0  ja @f  mov eax, 0  ret  @@:  
    cmp edx, 2  ja @f  mov eax, 1  ret  @@:  push ebx  mov ebx, 1  
    mov ecx, 1  @@:  lea eax, [ebx+ecx]  cmp edx, 3  jbe @f  mov ebx,  
    ecx  mov ecx, eax  dec edx  jmp @b  @@:  pop ebx  ret
```

C++:

```
int fibonacci(const int x) {  
    if (x==0) return(0);  
    if (x==1) return(1);  
    return (fibonacci(x-1))+fibonacci(x-2);} 
```



- Domain:
 1. General-purpose programming languages (GPL), such as Fortran, C/C++, Python, ...
 2. Domain specific language (DSL) such as Julia, R, Matlab, Mathematica, ...
- Advantages/disadvantages:
 1. GPL are more powerful, usually faster to run.
 2. DSL are easier to learn, faster to code, built-in functions and procedures.

Execution I

- Three basic modes to run code:
 1. Interpreted: Python, R, Matlab, Mathematica.
 2. Compiled: Fortran, C/C++.
 3. JIT (Just-in-Time) compilation: Julia.
- Interpreted languages can we used with:
 1. A command line in a REPL (Read-eval-print loop).
 2. A script file.
- Many DSL are interpreted, but this is neither necessary nor sufficient.
- Advantages/disadvantages: similar to GPL versus DSL.
- Interpreted and JIT programs are easier to move across platforms.

Execution II

- In reality, things are somewhat messier.
- Some languages are explicitly designed with an interpreter and a compiler (Haskell, Scala, F#).
- Compiled programs can be extended with third-party interpreters (CINT and Cling for C/C++).
- Often, interpreted programs can be compiled with an auxiliary tool (Matlab, Mathematica,...).
- Interpreted programs can also be compiled into byte code (R, languages that run on the JVM -by design or by a third party compiler).
- We can mix interpretation/compilation with libraries.

Types I

- Type strength:
 1. Strong: type enforced.
 2. Weak: type is tried to be adapted.
- Type expression:
 1. Manifest: explicit type.
 2. Inferred: implicit.
- Type checking:
 1. Static: type checking is performed during compile-time.
 2. Dynamic: type checking is performed during run-time.
- Type safety:
 1. Safe: error message.
 2. Unsafe: no error.

- Advantages of strong/manifest/static/safe type:
 1. Easier to find programming mistakes⇒ADA, for critical real-time applications, is strongly typed.
 2. Easier to read.
 3. Easier to optimize for compilers.
 4. Faster runtime not all values need to carry a dynamic type.
- Disadvantages:
 1. Harder to code.
 2. Harder to learn.
 3. Harder to prototype.

Types III

- You implement strong/manifest/static/safe typing in dynamically typed languages.
- You can define variables explicitly. For example, in Julia:

```
a::Int = 10
```

- It often improve performance speed and safety.
- You can introduce checks:

```
a = "This is a string"  
if typeof(a) == String  
    println(a)  
else  
    println("Error")  
end
```

Mar 2019	Mar 2018	Change	Programming Language	Ratings	Change
1	1		Java	14.880%	-0.06%
2	2		C	13.305%	+0.55%
3	4	▲	Python	8.262%	+2.39%
4	3	▼	C++	8.126%	+1.67%
5	6	▲	Visual Basic .NET	6.429%	+2.34%
6	5	▼	C#	3.267%	-1.80%
7	8	▲	JavaScript	2.426%	-1.49%
8	7	▼	PHP	2.420%	-1.59%
9	10	▲	SQL	1.926%	-0.76%
10	14	▲▲	Objective-C	1.681%	-0.09%
11	18	▲▲	MATLAB	1.469%	+0.06%
12	16	▲▲	Assembly language	1.413%	-0.29%
13	11	▼	Perl	1.302%	-0.93%
14	20	▲▲	R	1.278%	+0.15%
15	9	▼▼	Ruby	1.202%	-1.54%
16	60	▲▲	Groovy	1.178%	+1.04%
17	12	▼▼	Swift	1.158%	-0.99%
18	17	▼	Go	1.016%	-0.43%
19	13	▼▼	Delphi/Object Pascal	1.012%	-0.78%
20	15	▼▼	Visual Basic	0.954%	-0.79%

Programming Language	2019	2014	2009	2004	1999	1994	1989
Java	1	2	1	1	10	-	-
C	2	1	2	2	1	1	1
C++	3	4	3	3	2	2	3
Python	4	7	5	10	29	21	-
Visual Basic .NET	5	11	-	-	-	-	-
C#	6	5	7	7	27	-	-
JavaScript	7	8	8	8	18	-	-
PHP	8	6	4	5	-	-	-
SQL	9	-	-	6	-	-	-
Objective-C	10	3	36	45	-	-	-
COBOL	25	19	15	11	3	9	13
Lisp	27	13	18	14	13	5	2
Pascal	204	14	14	96	6	3	7

Language popularity I

- C family (a subset of the ALGOL family), also known as “curly-brackets languages”:
 1. C, C++, C#: 24.70%: 3 out of top 6.
 2. Java, C, C++, C#, Objective-C, JavaScript, PHP, Perl: 47.41%: 7 out of top 10.
- Python: position 3, 8.26%.
- Matlab: position 11, 1.47%.
- R: position 14, 1.28%.
- Fortran: position 27, 0.47%.
- Julia: position 42, 0.21%.

Language popularity II

- High-performance and scientific computing is a small area within the programming community.
- Thus, you need to read the previous numbers carefully.
- For example:
 1. You will most likely never use JavaScript or PHP (at least while wearing with your “economist” hat) or deal with an embedded system.
 2. C# and Objective-C are cousins of C focused on industry applications not very relevant for you.
 3. Java (usually) pays a speed penalty.
 4. Fortran is still used in some circles in high-performance programming, but most programmers will never bump into anyone who uses Fortran.

Multiprogramming

- Attractive approach in many situations.
- Best IDEs can easily link files from different languages.
- Easier examples:
 1. Cpp.jl and PyCall in Julia.
 2. Rcpp.
 3. Mex files in Matlab.