

# Programming Languages: Concepts

(Lectures on High-performance Computing for Economists IV)

---

Jesús Fernández-Villaverde<sup>1</sup> and Pablo Guerrón<sup>2</sup>

August 27, 2024

<sup>1</sup>University of Pennsylvania

<sup>2</sup>Boston College

# Introduction

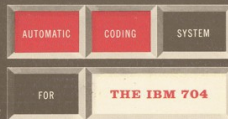
---

# Motivation

- Since the invention of Fortran in 1954-1957 to substitute assembly language, hundreds of programming languages have appeared.
- Some are more successful than others, some more useful than others.
- Moreover, languages evolve (different version of Fortran).
- Different languages are oriented toward certain goals and have different approaches.
- Our thinking about what is a good programming language has also changed as we accumulate more experience with computers.

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  $\text{\LaTeX}$ .

## Alan Perlis

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

- You will likely 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 Kubernetes 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.

- Levels:
  1. machine code.
  2. Low level: assembly language like NASM (<http://www.nasm.us/>), GAS, or HLA (*The Art of 64-Bit Assembly*, 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, 3 jbe @f mov  
ebx, ecx [ebx+ecx] cmp edx, 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. GPLs are more powerful and usually faster to run.
  2. DSLs are easier to learn, faster to code, built-in functions and procedures.

# Execution I

- Three basic modes to run code:
  1. Interpreted: Python, R, Mathematica.
  2. Compiled: Fortran, C/C++.
  3. JIT (Just-in-Time) compilation: Julia, Matlab.
- Interpreted languages can we used with:
  1. A command line in a REPL (Read-eval-print loop).
  2. A script file.
- Many DSLs 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.

- 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 = 10::Int
```

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

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

Aug 2024	Aug 2023	Change	Programming Language		Ratings	Change
1	1		 Python		18.04%	+4.71%
2	3	▲	 C++		10.04%	-0.59%
3	2	▼	 C		9.17%	-2.24%
4	4		 Java		9.16%	-1.16%
5	5		 C#		6.39%	-0.65%
6	6		 JavaScript		3.91%	+0.62%
7	8	▲	 SQL		2.21%	+0.68%
8	7	▼	 Visual Basic		2.18%	-0.45%
9	12	▲	 Go		2.03%	+0.87%
10	14	▲	 Fortran		1.79%	+0.75%
11	13	▲	 MATLAB		1.72%	+0.67%
12	23	▲	 Delphi/Object Pascal		1.63%	+0.83%
13	10	▼	 PHP		1.46%	+0.19%
14	19	▲	 Rust		1.28%	+0.39%
15	17	▲	 Ruby		1.28%	+0.37%
16	18	▲	 Swift		1.28%	+0.37%
17	9	▼	 Assembly language		1.21%	-0.13%
18	27	▲	 Kotlin		1.13%	+0.44%
19	16	▼	 R		1.11%	+0.19%
20	11	▼	 Scratch		1.09%	-0.13%

Programming Language	2024	2019	2014	2009	2004	1999	1994	1989
Python	1	3	8	6	8	26	23	-
C	2	2	1	2	2	1	1	1
C++	3	4	4	3	3	2	2	2
Java	4	1	2	1	1	16	-	-
C#	5	6	5	7	7	21	-	-
JavaScript	6	7	9	9	9	18	-	-
Visual Basic	7	19	236	-	-	-	-	-
SQL	8	9	-	-	92	-	-	-
Go	9	17	36	-	-	-	-	-
PHP	10	8	6	4	6	-	-	-
Objective-C	33	10	3	33	41	-	-	-
Lisp	34	32	14	20	14	14	6	3
(Visual) Basic	-	-	7	5	5	3	3	7

# Language popularity I

- C family (a subset of the ALGOL family), also known as “curly-brackets languages”:
  1. C, C++, C#: 25.60%: 3 out of top 5.
  2. C, C++, Java, C#, JavaScript: 38.67%: 5 out of top 10.
- Python: position 1, 18.04%.
- Fortran: position 10, 1.79%.
- Matlab: position 11, 1.72%.
- R: position 19, 1.11%.
- Julia: position 32, 0.48%.

# 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 Swift 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.

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