Why do Java, Go and Rust all have a “main”?
https://alexwlchan.net/2020/why-do-programming-languages-have-a-main-function/
The do because C, does
C does because B did but BCPL (which preceded B) has Start rather than Main

Sec 1.2
Declarative
Not how but what (a 4th/5th generation language)?
SQL, ProLog

facts:

```
%% Footloose (1984)
cast(['Kevin Bacon', 'Lori Singer', 'John Lithgow', 'Dianne Wiest']).
cast(['John Lithgow', 'Robin Williams', 'Mary Beth Hurt', 'Glenn Close']). % world according to garp
```

rules:

```
co_starred(X, Y) :-
        cast(L),
        member(X, L),
        member(Y, L),
        X \== Y.

next_node(C, N, P) :-
        co_starred(C, N),
        \+ member(N, P).
dfs(G, G, _, [G]).
dfs(S, G, V, [S|P]) :-
        next_node(S, NN, V),
        dfs(NN, G, [NN|V], P).

count_edges([_], 0).
count_edges([_|T], N) :-
        count_edges(T, N1),
        N is N1 + 1.

bacon_number(X, N) :-
        setof((L, P), (dfs('Kevin Bacon', X, [], P), length(P, L)), [(_, SP)|_]),
        count_edges(SP, N).
```

USE:

```
co_starred('Kevin Bacon', 'Val Kilmer').
false.
bacon_number('Kevin Bacon', 0).
true .

?- bacon_number(X, 1).
X = 'Bill Paxton' ;

bacon_number(X,2).
X=‘Glenn Close’ ;
```

etc

Imperative
Most languages

Functional
No variables, no side effects
see Quicksort/qsort.ex

sec 1.3
what makes a programming language successful?
The first three are about the language itself, the rest are about social factors.
• Successful languages must have modest or minimal computer resource requirements.
• Successful languages must have a simple performance model.
• Successful languages must not require users have “mathematical sophistication.”
• The language must be available on a wide variety of hardware.
• It helps to have local wizards or gurus for the language.
• It must be a minimally acceptable language.
• It must be similar to existing popular languages

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69

Sec 1.4
Greenness??

On average, the compiled languages consumed 120J to execute solutions, while for virtual machine languages and interpreted languages, the value was 576J and 2365J, respectively.

This trend can also be observed in the case of execution time, as compiled languages required 5103ms, virtual machine languages 20623ms, and interpreted languages 87614ms (on average).

https://stratoflow.com/efficient-and-environment-friendly-programming-languages/

Why new languages
- languages are accepted and evolve socially
- languages minimal resources
- simple "performance model"
  - easy to know fast vs slow operations
  - alternately: what is a primitive operation in Big-O sense.

  - Non trivial to determine in many langs:
  - drawing 10,000,000 rands ints then sorting

Go:	array of structs by number:	2.88
	array of ints	2.636930156s
Elixir:	list of integers	5.7
	list of structs sort by integer	17.98
Rust:	vec of integers	7.57 - 8.54s
	vec of struct of int	6.96s
	optimized:	
	0.807	
	0.824	
Java:	array of int:	1.1
	array of instances of a class	6.6
python:	array of int:	14.24
	array or array of 1 int	43.12 or 18.96
	array of dicts	22.6
	array of objects	23.1

  - factor numbers up to 200,000 by "trial division"
    - Go: 122.5 seconds
    - Rust: 116.67 seconds
      - optimized 23.04 sec
    - Elixir: ~250 seconds
  - so what is the speed difference between Go, Elixir and Rust?

- easy to understand
- ALSO for a new lang:
  - widely available
  - local experts
  - "minimally acceptable"
  - similar to existing langs

- reason to move

Bad languages

APL: instrumented in greek

Game of life

If:((2×+⌿¯1 0 1⊖⍤0 2+⌿¯1 0 1⌽⍤0 2 ⍵)-⍵)∈5 6 7

Every cell interacts with its eight *neighbours*, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Null — why is null a “billion dollar mistake”

Null was introduced in Algol to speed programs. Idea, allow reading off the end of array. Just always return null if you do so.

How does this speed things up?

what cost saftey?

https://hinchman-amanda.medium.com/null-pointer-references-the-billion-dollar-mistake-1e616534d485

“…. a null pointer reference could be a bad idea. Comparing a null pointer reference to a promiscuous adulterer he noted that the null assignment for every bachelor represented in an object structure “will seem to be married polyamorously to the same person Null”.

WHAT IS A COMPILER?

what does a compiler do?

translate from high-level language into machine language

2 aspects

thorough analysis

non-trivial transformation

eg. tail-recursion to iteration

Compiled languages

Source code ——> COMPILER ——> machine executable

input // \\ output

C, Go, fortran, …..

What is alternative to compiler?

interpreter

Source Code ——>

Interpreter ——> Output

Input ——>

Python, lisp, perl, javascript

why have a compiler vs interpreter?

tradeoffs between

interpret: flexibility, diagnostics (at run time)
LISP: program can write code that it executes
late binding
compile: speed!!!!

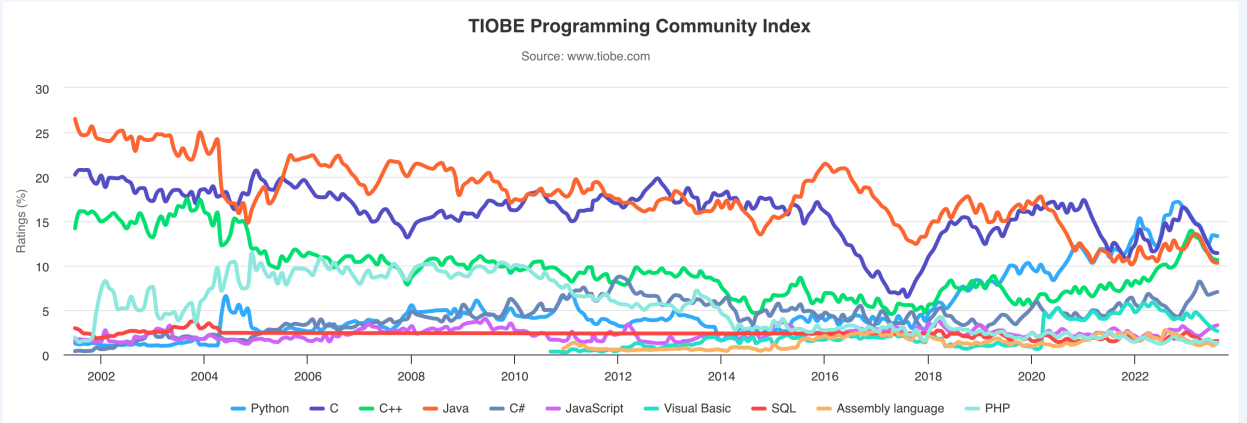
"Byte-compiled languages" — Java, Erlang
Source ——> Translator ——> Intermediate program
\\//
Input ——> Virtual Machine ——> Output
Virtual Machine???? Why?
"java — write once, run everywhere"

Languages built on top of other languages
Kotlin, groovy, scala, clojure uses JVM
also JVM implementations of many common languages
Elixir uses Erlang VM

Why this use of virtual machines at all?
Does it make sense?

Another way of looking at PLs
https://medium.com/codex/what-are-the-greenest-programming-languages-e738774b1957

Tiobe Community popularity Survey from tiobe.com
https://www.tiobe.com/tiobe-index/



also rust == 19th, go=15th

Factors making a programming language easier to learn.

1. Availability of free resources: Various courses and tutorials make it easier for learners to grasp the language quickly.
2. Consistent rules and simplicity: Programming languages that adhere to uniform rules will simplify learning new commands and functions, enabling beginners to grasp the concepts more quickly.
3. Integration with other languages: Making it more versatile and user-friendly for beginners.
4. Syntax resembling simple English: Programming languages with straightforward syntax that incorporates English words and closely resembles the English language structure are more relatable and accessible for beginners to comprehend and learn.
5. Large developer community: A supportive community helps learners to grow and troubleshoot any issues they may face.

Factors making a programming language harder to learn.

1. Complex and unusual syntax: Some languages have complicated rules for structuring code or unconventional ways of organizing it, making it difficult for beginners to understand and write.
2. Multiple programming paradigms: A language that supports various programming styles can take more work, as it requires understanding different approaches to solving problems, such as functional programming, which emphasizes immutability and mathematical functions.
3. Error handling and predicting output: In some languages, it is challenging to identify and fix errors or bugs, and developers may need to anticipate the results of their code before running it, making it more difficult for newcomers.
4. Technical jargon and abstract concepts: Some languages use specialized terms and concepts or involve abstract ideas, making them harder to grasp for people without a background in computer science.
5. Domain-specific focus and limited code reuse: Some languages are designed for specific purposes, such as artificial intelligence, making them less accessible to general users. Additionally, it can be challenging to reuse code snippets in specific languages, making it harder to learn from examples and build upon existing code.
6. Expert guidance needed: Some languages require the guidance of an experienced tutor or mentor to understand and learn effectively.

https://techreviewer.co/blog/the-easiest-and-hardest-programming-languages-to-learn