IN4MATX 285:
Interactive Technology Studio

Programming: Language Roles

Today's goals

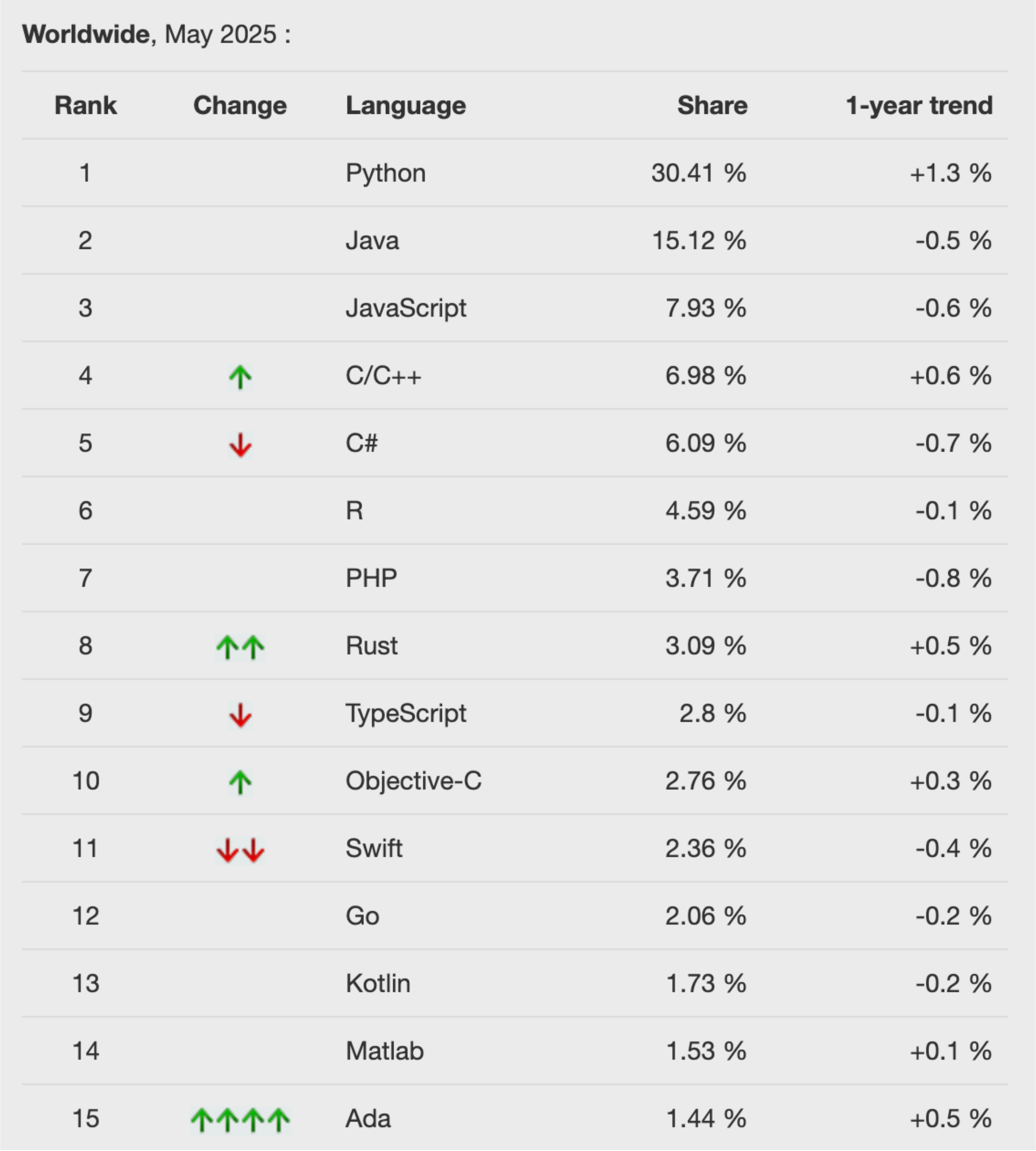
By the end of today, you should be able to...

- Differentiate programming languages based on factors such as types and level of abstraction

- Identify the sorts of tasks that a programming language is well-designed for, given a brief description

There are a lot of programming languages out there. They all fill a particular niche.

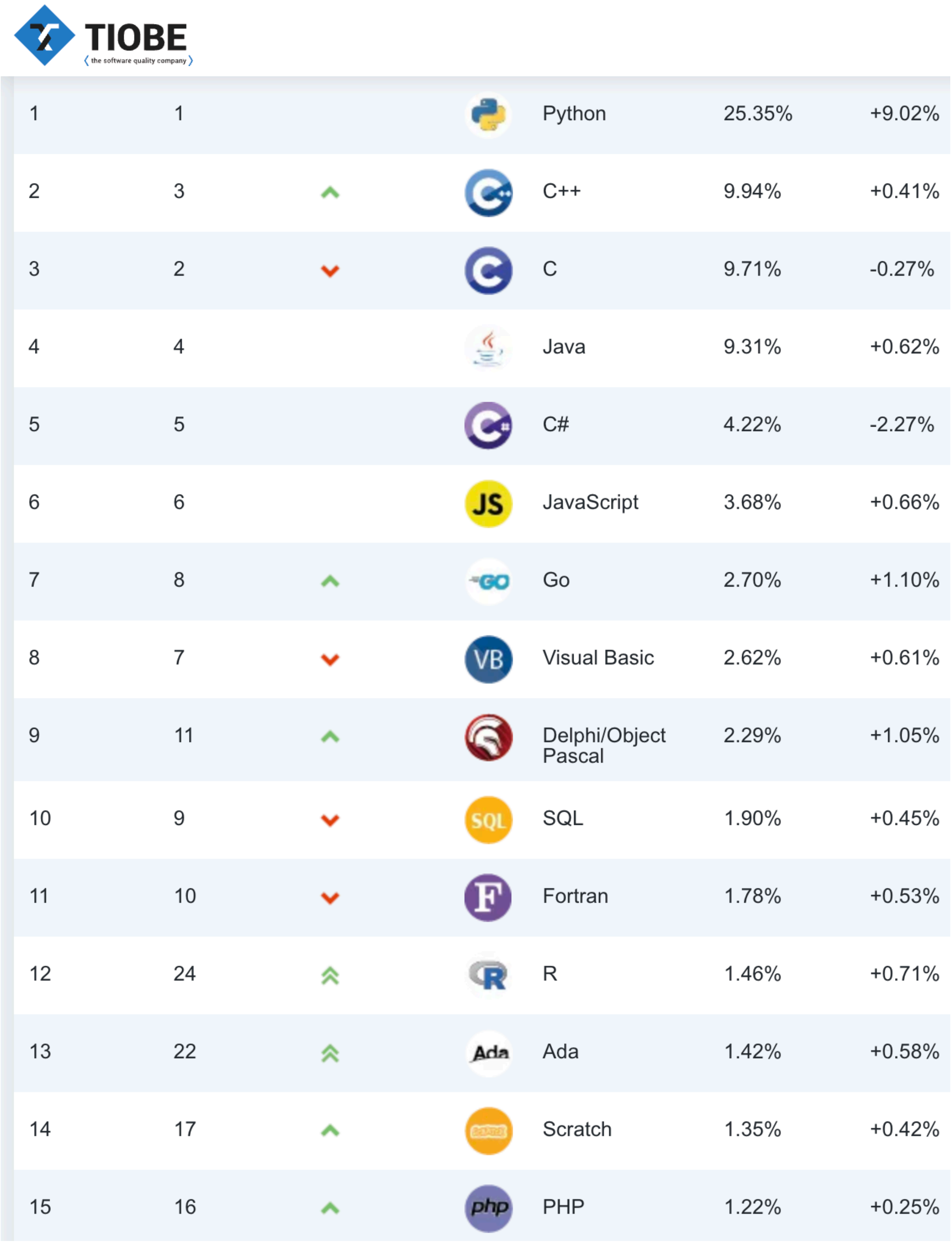
So many programming languages!



Worldwide, May 2025 :

Rank	Change	Language	Share	1-year trend
1		Python	30.41 %	+1.3 %
2		Java	15.12 %	-0.5 %
3		JavaScript	7.93 %	-0.6 %
4	↑	C/C++	6.98 %	+0.6 %
5	↓	C#	6.09 %	-0.7 %
6		R	4.59 %	-0.1 %
7		PHP	3.71 %	-0.8 %
8	↑↑	Rust	3.09 %	+0.5 %
9	↓	TypeScript	2.8 %	-0.1 %
10	↑	Objective-C	2.76 %	+0.3 %
11	↓↓	Swift	2.36 %	-0.4 %
12		Go	2.06 %	-0.2 %
13		Kotlin	1.73 %	-0.2 %
14		Matlab	1.53 %	+0.1 %
15	↑↑↑↑	Ada	1.44 %	+0.5 %

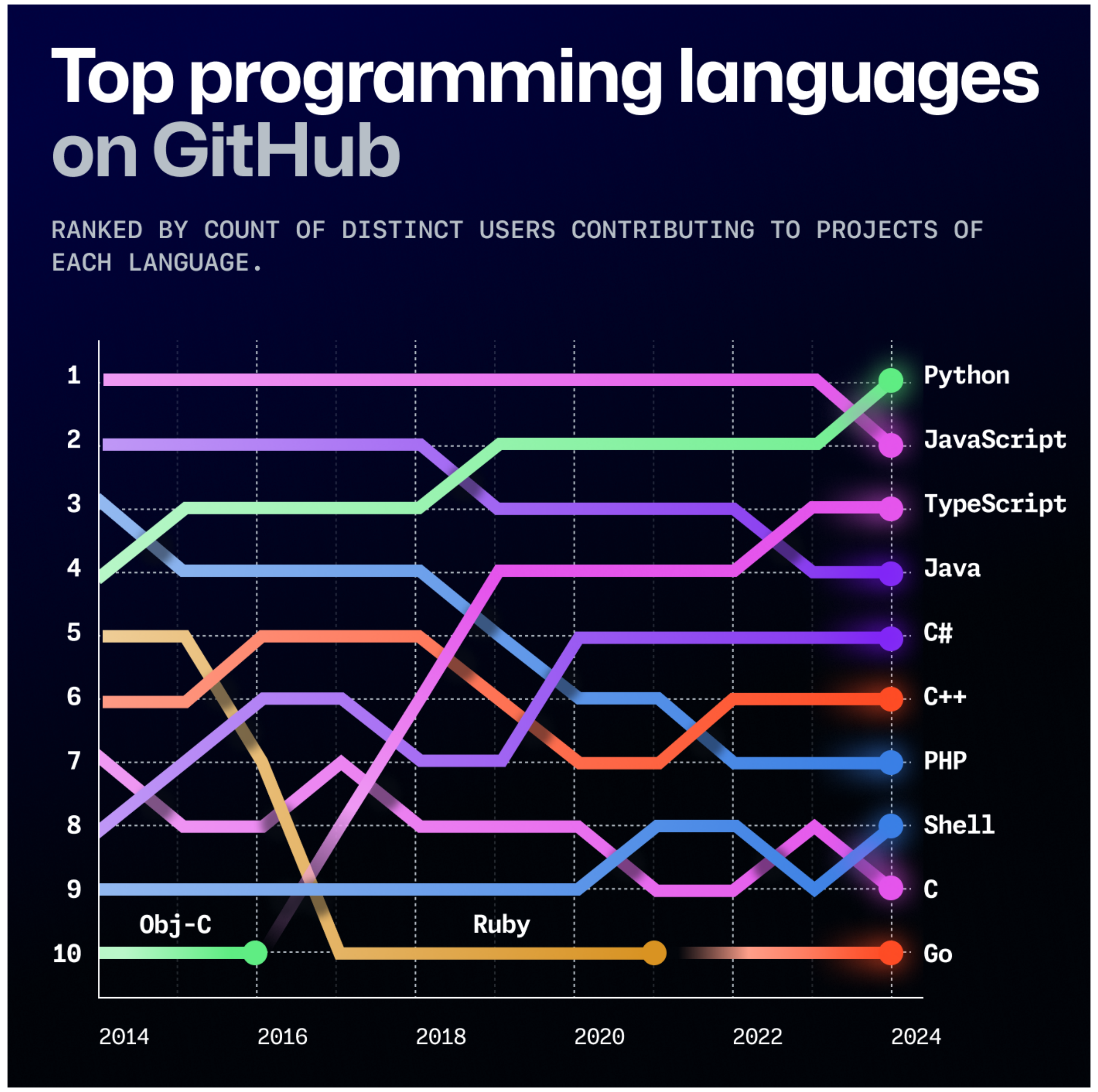
https://pypl.github.io/PYPL.html



TIOBE

1	1		Python	25.35%	+9.02%
2	3	↑	C++	9.94%	+0.41%
3	2	↓	C	9.71%	-0.27%
4	4		Java	9.31%	+0.62%
5	5		C#	4.22%	-2.27%
6	6		JavaScript	3.68%	+0.66%
7	8	↑	Go	2.70%	+1.10%
8	7	↓	Visual Basic	2.62%	+0.61%
9	11	↑	Delphi/Object Pascal	2.29%	+1.05%
10	9	↓	SQL	1.90%	+0.45%
11	10	↓	Fortran	1.78%	+0.53%
12	24	⇈	R	1.46%	+0.71%
13	22	⇈	Ada	1.42%	+0.58%
14	17	↑	Scratch	1.35%	+0.42%
15	16	↑	PHP	1.22%	+0.25%

https://www.tiobe.com/tiobe-index/



Top programming languages on GitHub

RANKED BY COUNT OF DISTINCT USERS CONTRIBUTING TO PROJECTS OF EACH LANGUAGE.

https://github.blog/news-insights/octoverse/octoverse-2024/

So many programming languages!

- Lots of differentiating factors
    - Level of abstraction (high, low)
    - Domain specialization (general-purpose, domain-specific)
    - Runtime and execution model (compiled, interpreted)
    - Programming paradigm (object-oriented, functional)
    - Typing (strong, weak)
- I won’t go through all of these, but you might hear these terms

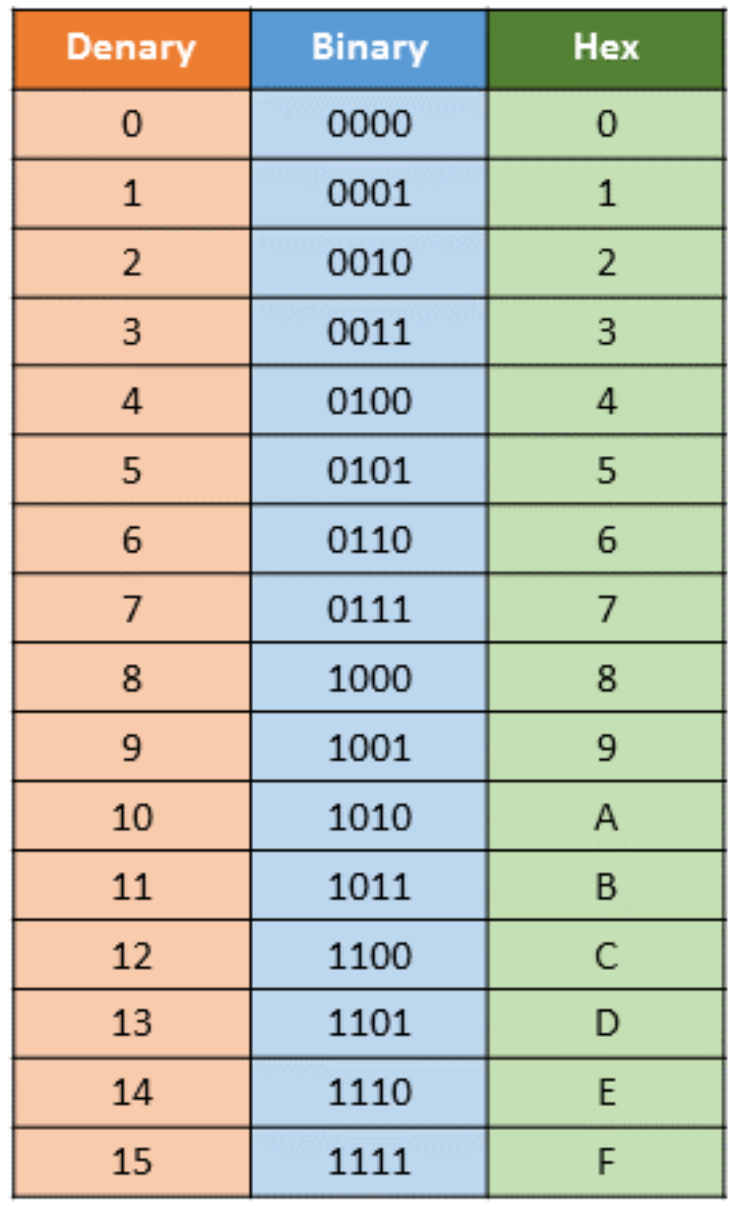
So many programming languages!

- Today, I’ll introduce some of the key differentiators
- I’ll spend a bit more time with C/C++ and Python to explain some of the core differences

Level of abstraction

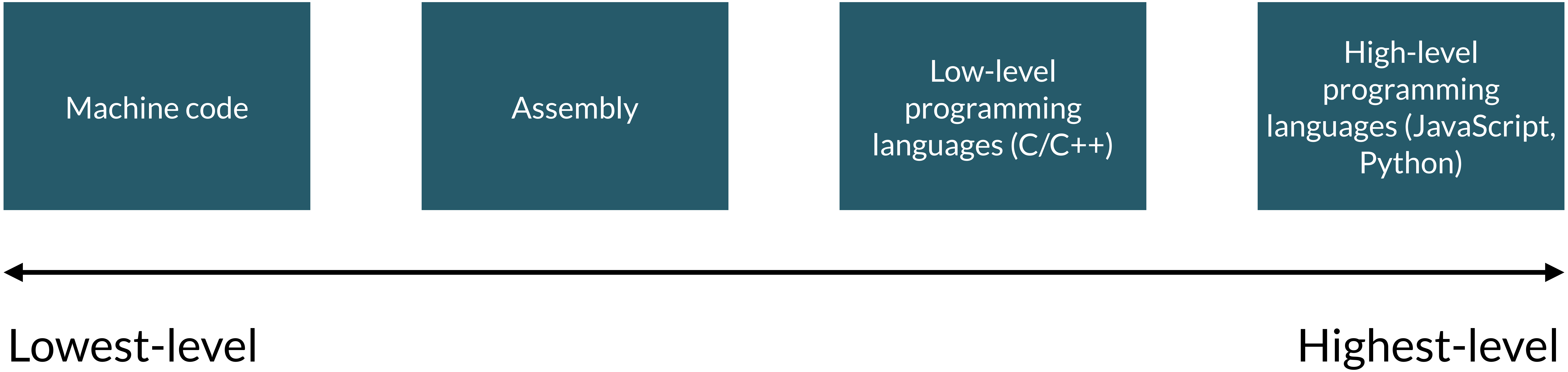
Level of abstraction

- Your computer is working entirely in binary (often shown as hexadecimal)
- Any code that you write eventually gets converted to binary
- Some programming languages are *low-level*, or very close to binary, whereas others are *high-level*



Denary	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Level of abstraction



Machine code

Assembly

Low-level programming languages (C/C++)

High-level programming languages (JavaScript, Python)

Lowest-level

Highest-level

Level of abstraction

Low-level languages

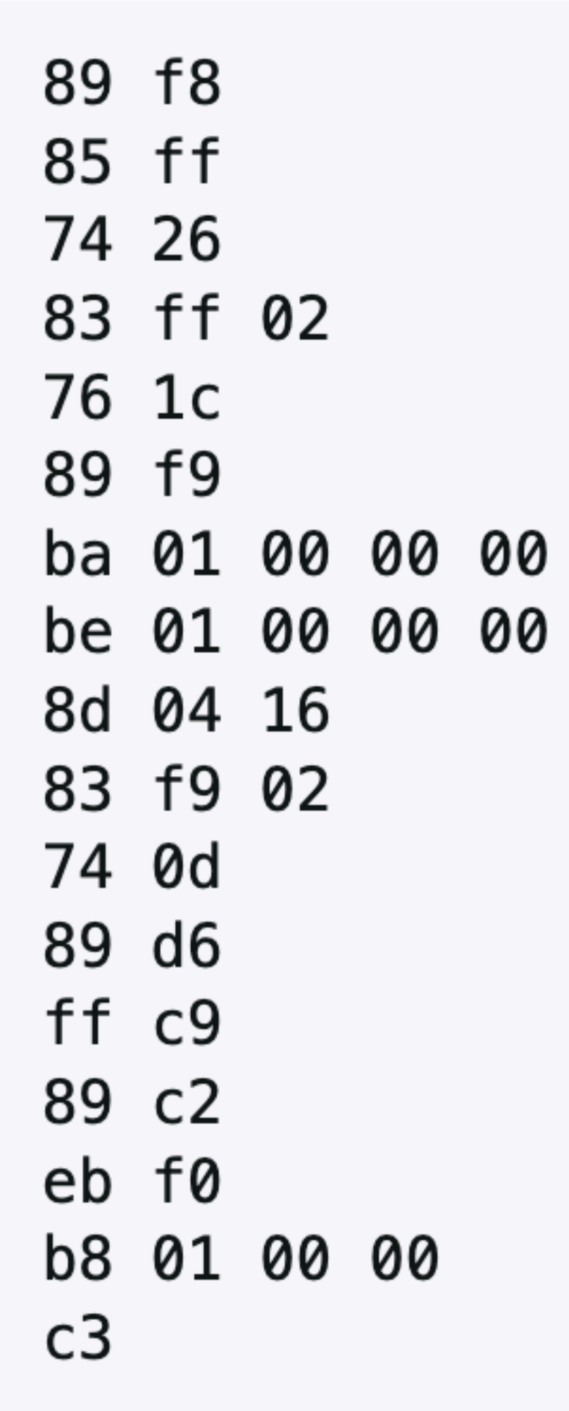
- No or minimal abstraction from the binary code running on a computer
- *Very* performant. Will run fast, won’t take up much memory
- All very bad choices if you want to make an interface, but good for other things :-)

https://en.wikipedia.org/wiki/Low-level_programming_language

Level of abstraction

Machine code

- The binary code which can be directly executed on a computer
- Very, very, very few people are programming in machine code
  - Occasionally people will learn to read it to understand memory dumps, like program crashes



```
89 f8
85 ff
74 26
83 ff 02
76 1c
89 f9
ba 01 00 00 00
be 01 00 00 00
8d 04 16
83 f9 02
74 0d
89 d6
ff c9
89 c2
eb f0
b8 01 00 00
c3
```

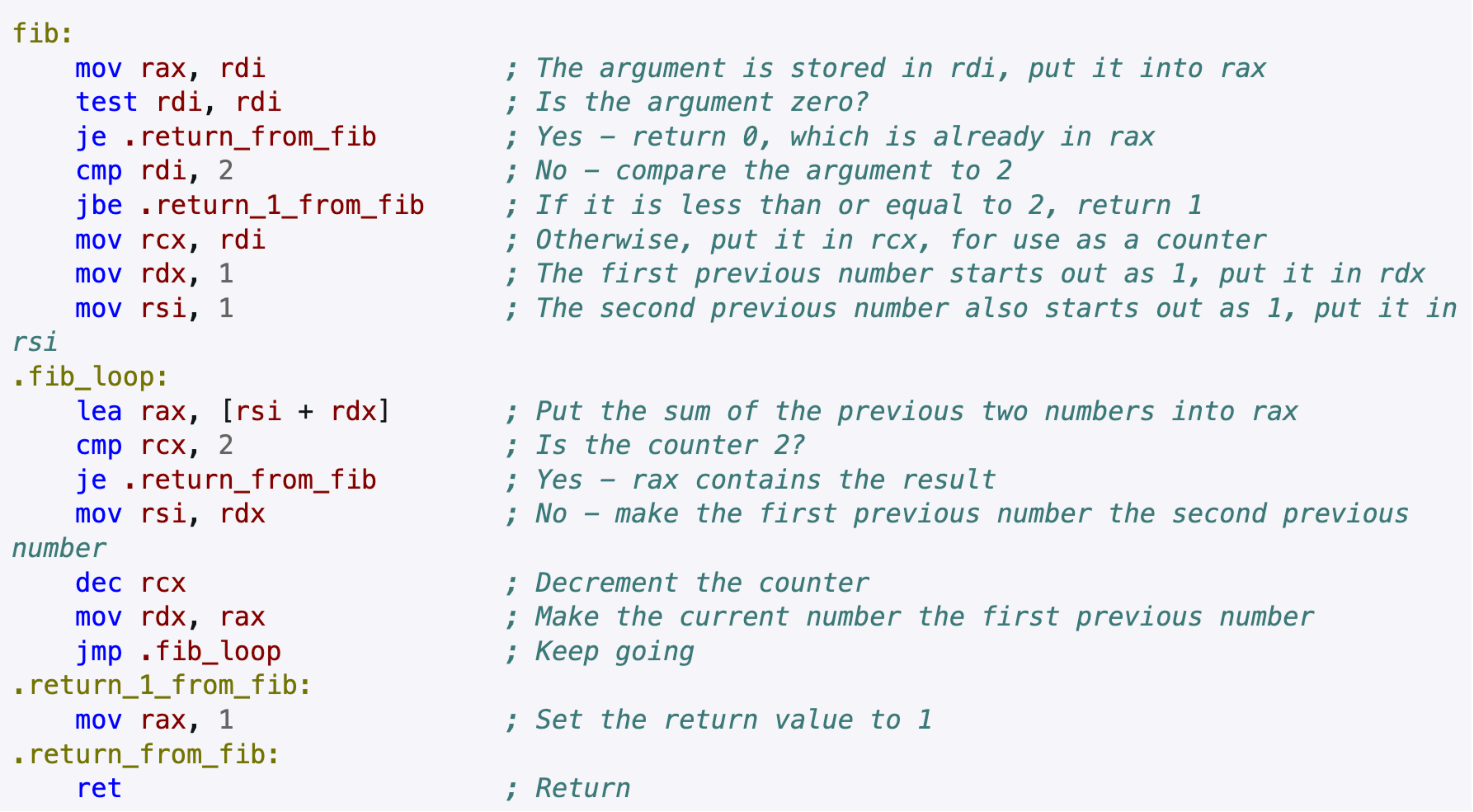
Fibonacci sequence (1, 1, 2, 3, 5, 8, 13, 21...)

https://en.wikipedia.org/wiki/Low-level_programming_language

Level of abstraction

Assembly

- Human-readable machine code
  - A lot of direct memory manipulation: moving (mov) bits around, comparing (comp), jumping to specific lines of code (je, jbe)
- People very occasionally code in assembly if optimization is crucial
  - But, we've gotten very good at optimizing automatically



```
fib:
    mov rax, rdi                 ; The argument is stored in rdi, put it into rax
    test rdi, rdi                ; Is the argument zero?
    je .return_from_fib          ; Yes - return 0, which is already in rax
    cmp rdi, 2                   ; No - compare the argument to 2
    jbe .return_1_from_fib       ; If it is less than or equal to 2, return 1
    mov rcx, rdi                 ; Otherwise, put it in rcx, for use as a counter
    mov rdx, 1                   ; The first previous number starts out as 1, put it in rdx
    mov rsi, 1                   ; The second previous number also starts out as 1, put it in
rsi
.fib_loop:
    lea rax, [rsi + rdx]         ; Put the sum of the previous two numbers into rax
    cmp rcx, 2                   ; Is the counter 2?
    je .return_from_fib          ; Yes - rax contains the result
    mov rsi, rdx                 ; No - make the first previous number the second previous
number
    dec rcx                      ; Decrement the counter
    mov rdx, rax                 ; Make the current number the first previous number
    jmp .fib_loop                ; Keep going
.return_1_from_fib:
    mov rax, 1                   ; Set the return value to 1
.return_from_fib:
    ret                          ; Return
```

Fibonacci sequence (1, 1, 2, 3, 5, 8, 13, 21…)

https://en.wikipedia.org/wiki/Low-level_programming_language

C/C++: a low(er) level programming language

C/C++

- C++ is slightly higher-level than C
    - C++ introduces objects similar to those in JavaScript
- Both are widely used and are fairly similar
    - Lowest-level languages which regularly show up in the top 10 most used languages
- I’ll talk about them interchangeably, writing code which should* work in both

https://en.wikipedia.org/wiki/C_(programming_language) https://en.wikipedia.org/wiki/C%2B%2B

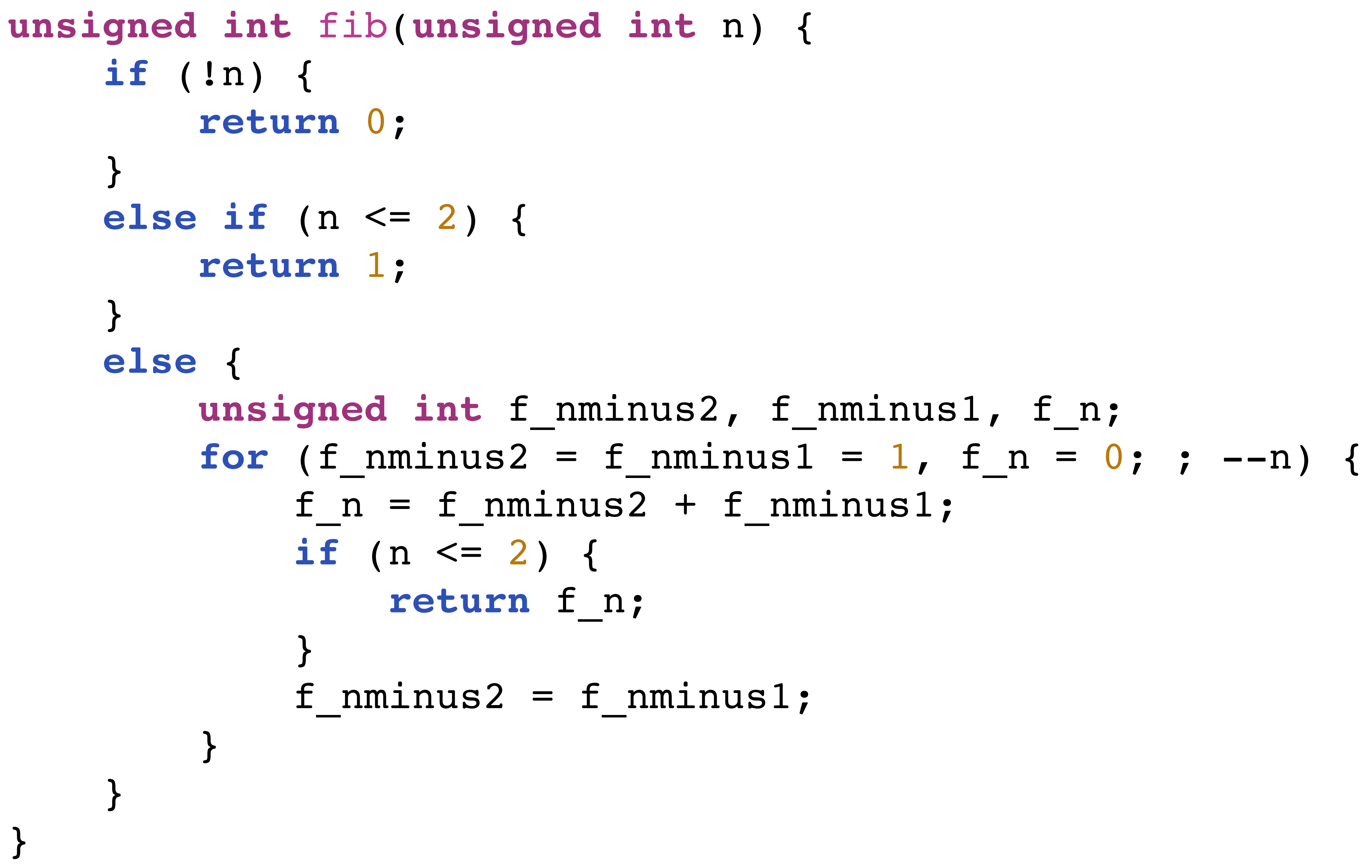
C/C++

- Both are especially used for coding:
  - Operating systems
  - Embedded systems (think Raspberry Pi, your smart washer/dryer)
  - Games
- Why?
  - Direct access to hardware and memory
  - High performance

https://en.wikipedia.org/wiki/C_(programming_language) https://en.wikipedia.org/wiki/C%2B%2B

C/C++

- Looks like a lot like JavaScript in syntax (for, if, else, return)
    - Compared to assembly, abstracts away the bits and some aspects of memory



```
unsigned int fib(unsigned int n) {
    if (!n) {
        return 0;
    }
    else if (n <= 2) {
        return 1;
    }
    else {
        unsigned int f_nminus2, f_nminus1, f_n;
        for (f_nminus2 = f_nminus1 = 1, f_n = 0; ; --n) {
            f_n = f_nminus2 + f_nminus1;
            if (n <= 2) {
                return f_n;
            }
            f_nminus2 = f_nminus1;
        }
    }
}
```

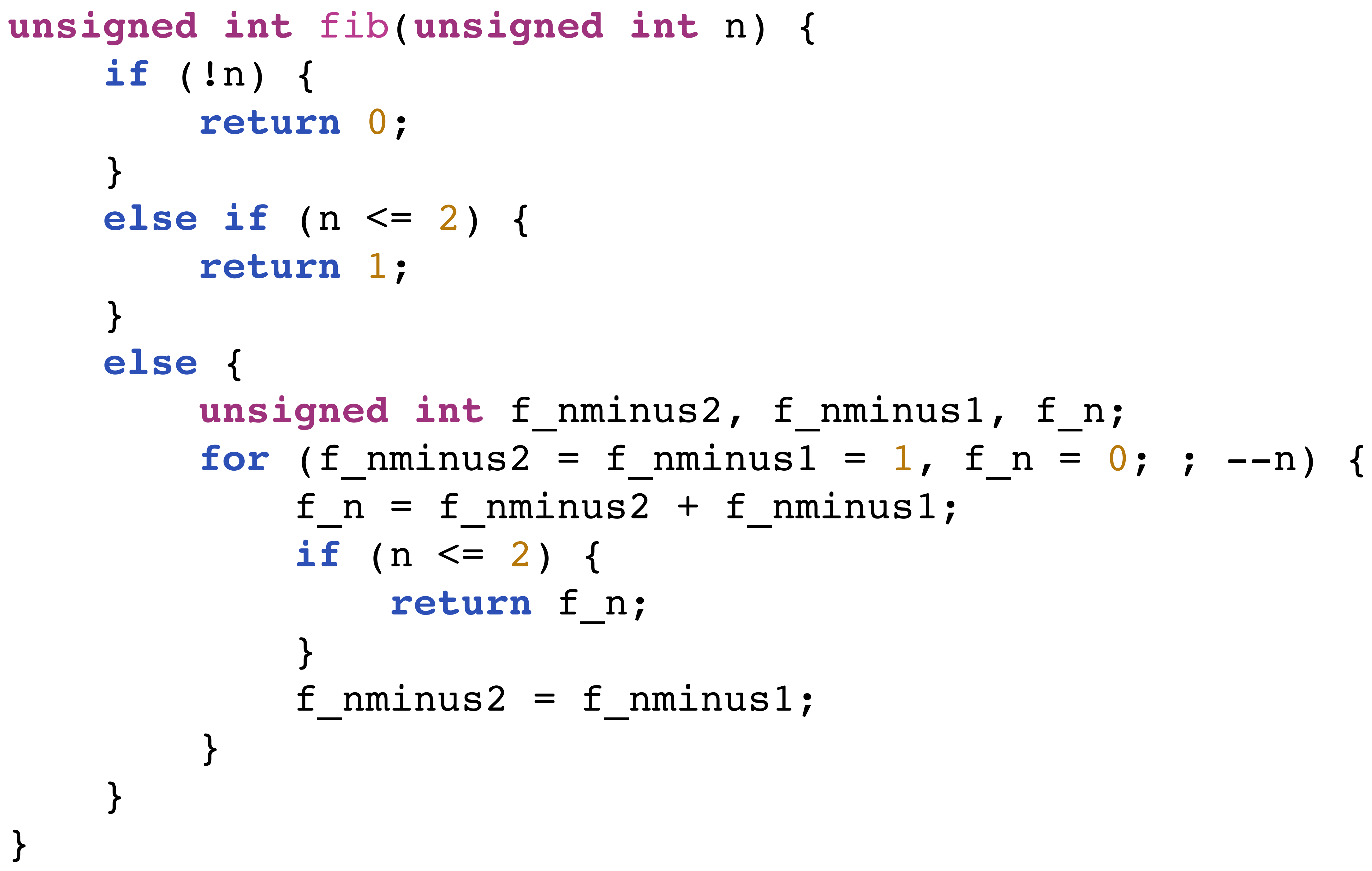
Fibonacci sequence (1, 1, 2, 3, 5, 8, 13, 21...)

https://en.wikipedia.org/wiki/C_(programming_language)

https://en.wikipedia.org/wiki/C%2B%2B

C/C++

- But what's an "int"?
  - These are *types*. Variables in C/C++ must always stay the same type
  - "Unsigned" ints must be positive numbers
  - Compared to JavaScript, where variables can change type
- Why types?
  - Reduces errors where code expects one type but receives another
  - Some languages are typed, others aren't



```
unsigned int fib(unsigned int n) {
    if (!n) {
        return 0;
    }
    else if (n <= 2) {
        return 1;
    }
    else {
        unsigned int f_nminus2, f_nminus1, f_n;
        for (f_nminus2 = f_nminus1 = 1, f_n = 0; ; --n) {
            f_n = f_nminus2 + f_nminus1;
            if (n <= 2) {
                return f_n;
            }
            f_nminus2 = f_nminus1;
        }
    }
}
```

Fibonacci sequence (1, 1, 2, 3, 5, 8, 13, 21...)

https://en.wikipedia.org/wiki/C_(programming_language)

https://en.wikipedia.org/wiki/C%2B%2B

C/C++

- But C/C++ still requires a lot of memory *management*
- Example: we want to make an array in memory
- Need to directly specify how big should be
  - E.g., it should hold exactly 5 integers/numbers
- Once we're done with it, we need to "free" the memory we've allocated

```
int *arr;

// Allocate memory for 5 integers
arr = (int *)malloc(5 * sizeof(int));

// Initialize and print the array
for (int i = 0; i < 5; i++) {
    arr[i] = i * 10;
    printf("arr[%d] = %d\n", i, arr[i]);
}

// Free the allocated memory
free(arr);
```

https://en.wikipedia.org/wiki/C_(programming_language)

https://en.wikipedia.org/wiki/C%2B%2B

Level of abstraction

Higher-level languages

- If you don’t need direct access to memory, and can sacrifice some performance
- Often viewed as “easier” to code
  - More interpretable
  - More concise

Python: a high(er)-level programming language

Python

- Known for being easy to develop with, having highly readable code, and a lot of rich libraries

- Increasingly becoming the way in which Computer Scientists/Software Engineers are introduced to programming
  - For undergraduates, we teach our first three programming courses in Python

- For me and many others, it’s the language to go to for a relatively-simple task that requires some coding
  - Data processing and organization, basic statistics or visualization

Python

- A few key syntax changes
  - No brackets {}, instead using tabs/ whitespace
  - No semicolons
  - Overall, attempts to be concise

```
def fib(n):
    if n <= 0:
        return 0
    elif n <= 2:
        return 1
    else:
        fib1, fib2, fib_total = 1, 1, 0
        while n > 0:
            fib_total = fib1 + fib2
            if n <= 2:
                return fib_total
            fib2 = fib1
            fib1 = fib_total
            n = n - 1
```

Fibonacci sequence (1, 1, 2, 3, 5, 8, 13, 21...)

Python

- Data structures are highly flexible
- And once you know what you’re doing, easy to use to concisely manipulate your data

```
titlesAndDois =
[{'title':
paper[‘title'],
'doi': paper[‘addons']
['doi']['url']}
for paper in
UCI_papers if 'addons'
in paper and 'doi' in
paper['addons']]
```

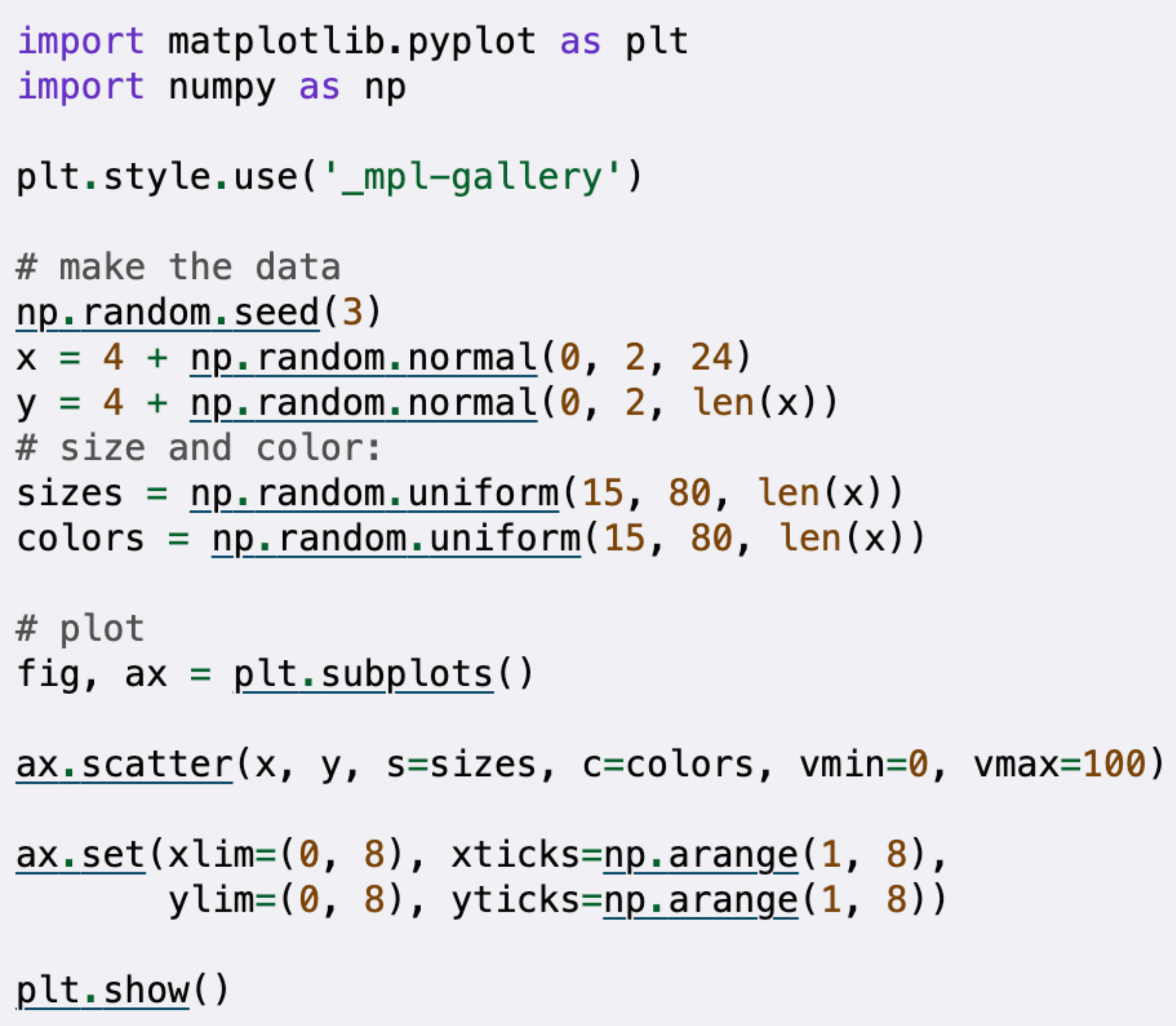
Filters and reshapes a list of objects
Will demonstrate in the demo!

Python

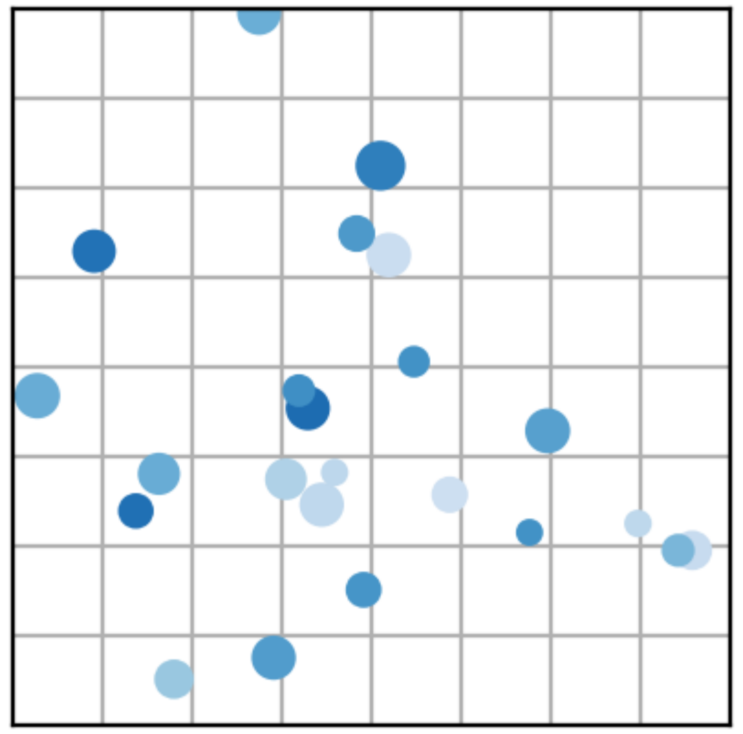
- Really good Data Science and Machine Learning libraries
  - Pandas
  - Numpy
  - Matplotlib
  - Tensorflow

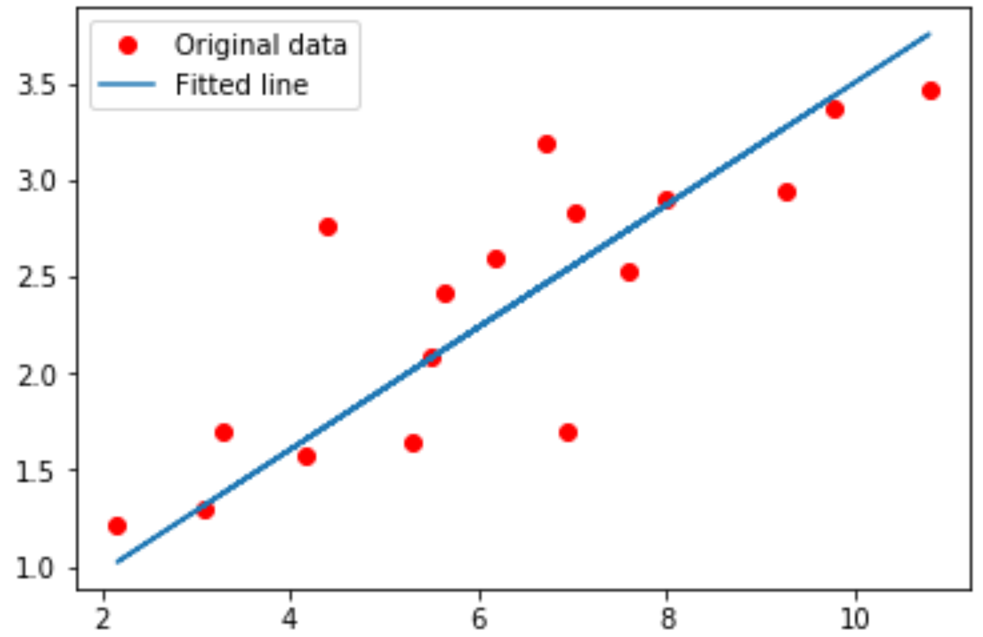
https://matplotlib.org/

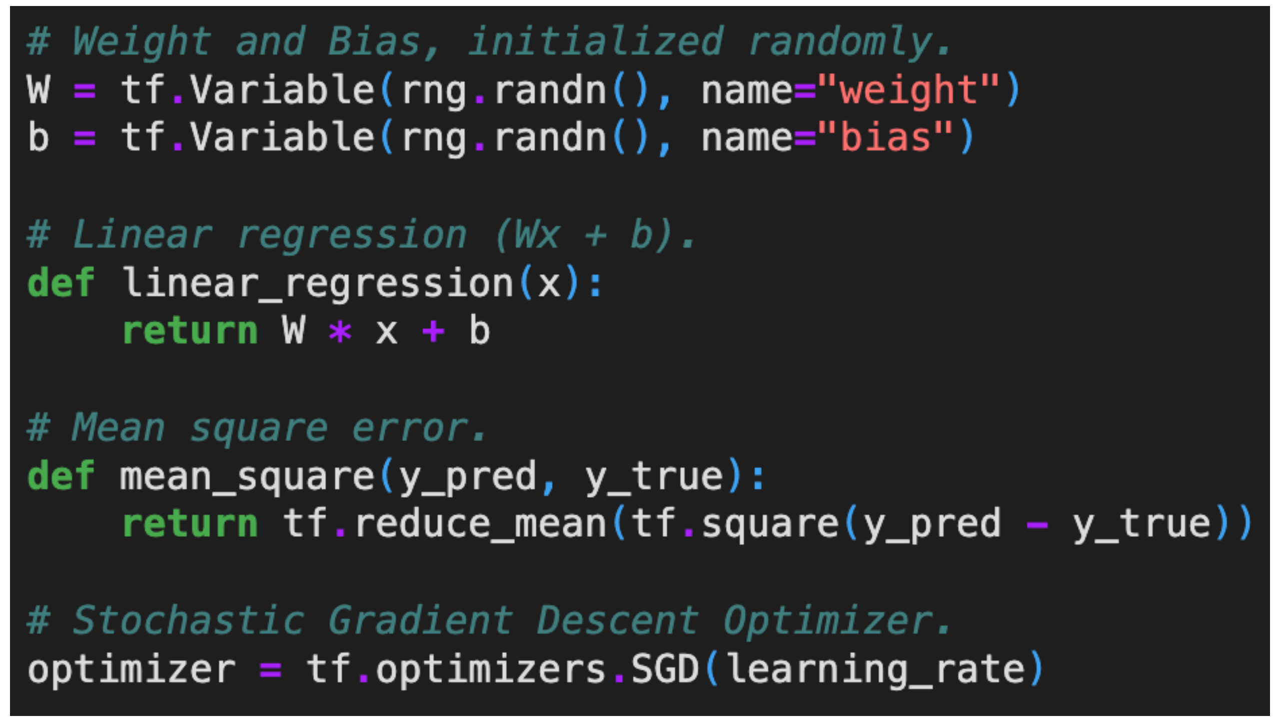
https://www.tensorflow.org/



```
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('_mpl-gallery')

# make the data
np.random.seed(3)
x = 4 + np.random.normal(0, 2, 24)
y = 4 + np.random.normal(0, 2, len(x))
# size and color:
sizes = np.random.uniform(15, 80, len(x))
colors = np.random.uniform(15, 80, len(x))

# plot
fig, ax = plt.subplots()

ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```







```
# Weight and Bias, initialized randomly.
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

# Linear regression (Wx + b).
def linear_regression(x):
    return W * x + b

# Mean square error.
def mean_square(y_pred, y_true):
    return tf.reduce_mean(tf.square(y_pred - y_true))

# Stochastic Gradient Descent Optimizer.
optimizer = tf.optimizers.SGD(learning_rate)
```

Other languages and wrapping up

Other languages

- PHP
  - Geared towards web server development, but largely replaced by other languages (JavaScript via NodeJS, Python)
- Java
  - Useful for being able to run on any device*, but not as designed for web/interfaces as JavaScript and not as performant as C/C++
- R
  - Used a lot for statistics and data visualization, but few use cases outside of those

My trajectory

- Java in 2005
- C/C++ in 2009
- PHP in 2010
- Python in 2011
- JavaScript in 2013
- R in 2016

- I mostly picked up programming languages when I was in school
    - I’ve never used Rust, Kotlin, Go. Wrote about 5 lines of Swift for an Apple Watch program
- I still use JavaScript and Python regularly, and R on occasion
- I’ve mostly forgotten how to use the others

Overall reflections

- It’s hardest to learn your first language, and relatively easier to learn subsequent ones
- Lots of knowledge transfers between programming languages
  - Syntax stays pretty similar
  - Constructs like loops, variables, and functions are pretty universal
- But languages are often intended to support different use cases, which changes how you use them
  - E.g., DOM manipulation makes JavaScript well-suited to interface development

Overall reflections

- Many can tasks *can* be done in a variety of programming languages
  - You can make an interface in C/C++, like with QT (https://www.qt.io/)
  - You can do many low-level memory operations in JavaScript
- But, your life will be *easier* if you choose a language that's well-suited for a particular task
  - Code will be more succinct, or easier to read/understand
  - What you/your team already knows should factor into that choice

Overall reflections

Programming Language design as UX design

- Like anything else, programming languages are designed, and are designed with particular use cases in mind

- Like in interfaces, it's often possible to do a variety of tasks, but common/important tasks are intentionally easier

- Maybe a bad analogy? You can tell me.

Overall reflections

Languages as community

- Part of what makes a language useful is its community of developers
    - We're all dependent on libraries, and particularly good libraries make a language more appealing
    - e.g., Python has really good libraries for Machine Learning, so people use it for that
    - JavaScript has been extended via React/Vue/Angular for improved interface development
    - Widely-used libraries are increasingly supporting multiple languages
- Chicken and egg question: do a lot of people use a language because it's good, or is a language good because a lot of people use it?

Today’s goals

By the end of today, you should be able to…

- Differentiate programming languages based on factors such as types and level of abstraction

- Identify the sorts of tasks that a programming language is well-designed for, given a brief description

IN4MATX 285:
Interactive Technology Studio

Programming: Language Roles