COMP201

Computer Systems & Programming

Lecture #01 – Introduction

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Spring 2026

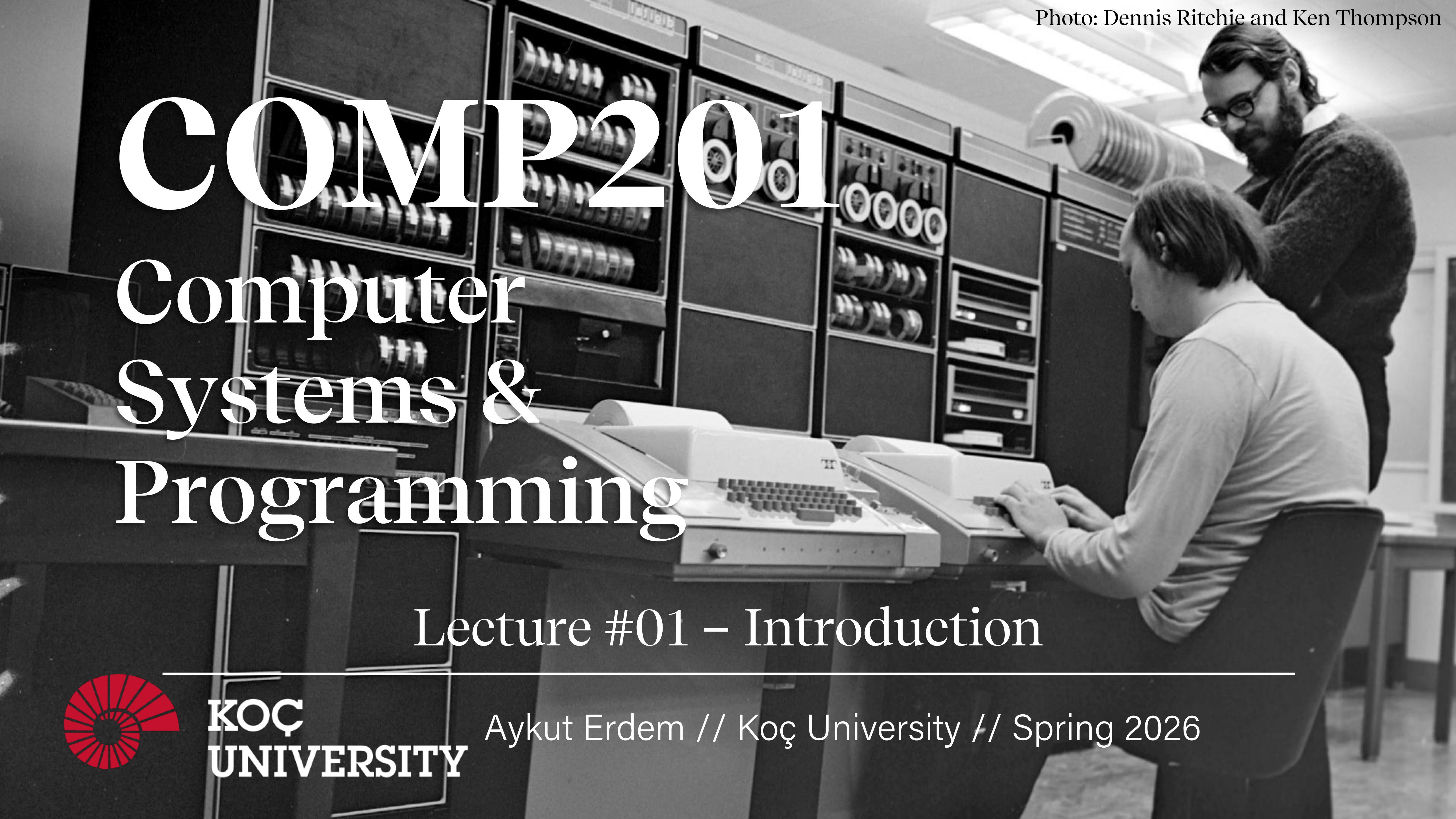


Photo: Dennis Ritchie and Ken Thompson

A little about me…

Koç University
Associate Professor
2020-now

Hacettepe University
Associate Professor
2010-2020

Universitá Ca' Foscari di Venezia
Post-doctoral Researcher
2008-2010

Middle East Technical University
1997-2008
Ph.D., 2008
M.Sc., 2003
B.Sc., 2001

MIT
Fall 2007
Visiting Student

VirginiaTech
Visiting Research Scholar
Summer 2006

The broad goal of my research is to explore better ways to **understand**, **interpret**, and **manipulate** visual data.

Research Interests

- Deep Learning
- Generative AI
- Computer Vision
- Language Understanding

🏠 https://aykuterdem.github.io

Plan For Today

- Course Introduction
- COMP201 Course Policies
- Unix and the Command Line
- Getting Started With C

Disclaimer: Slides for this lecture were borrowed from

—Nick Troccoli's Stanford CS107 class

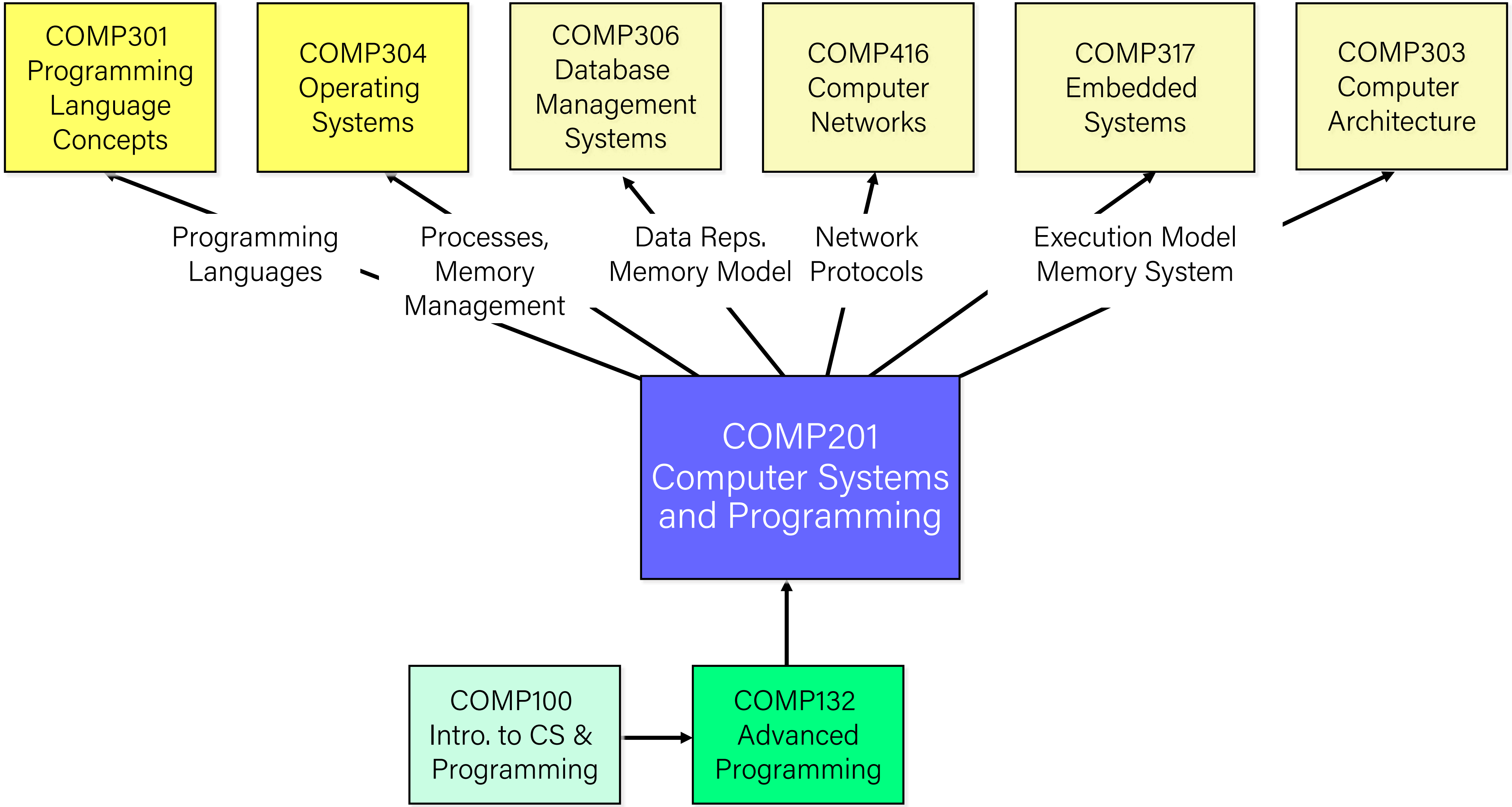
Lecture Plan

- Course Introduction
- COMP201 Course Policies
- Unix and the Command Line
- Getting Started With C

What is COMP201?

- The third course in the line of COMP’s introductory programming courses (COMP100, COMP132, and COMP201)
  - COMP100 teaches you the notion of computational thinking and how to solve problems as a programmer (using Python)
  - COMP132 introduces you object-oriented programming paradigm (using Java)
- COMP201 takes you **behind the scenes**:
  - Not quite down to hardware or physics/electromagnetism (that’s for later…)
  - It’s how things work **inside C++/Python/Java**, and how your programs map onto the components of computer systems
  - Not only does it just feel good to know how these work, it can also inform projects you work on in the future.

Role within COMP Curriculum



COMP301 Programming Language Concepts

COMP304 Operating Systems

COMP306 Database Management Systems

COMP416 Computer Networks

COMP317 Embedded Systems

COMP303 Computer Architecture

Programming Languages

Processes, Memory Management

Data Reps. Memory Model

Network Protocols

Execution Model Memory System

COMP201 Computer Systems and Programming

COMP100 Intro. to CS & Programming

COMP132 Advanced Programming

What is COMP201?

Computer Systems and Programming

- How languages like C++ and Java **represent data** under the hood
- How programming structures are encoded in **bits and bytes**
- How to efficiently **manipulate and manage memory**
- How computers **compile** programs
- How **cache memories work** and **how to exploit them** to improve the performance of your programs
- Uses the **C** programming language
- Programming **style** and software development practices

COMP201 Learning Goals

The goals for COMP201 are for students to gain **mastery** of

- writing C programs with complex use of memory and pointers
- an accurate model of the address space and compile/runtime behavior of C programs

to achieve **competence** in

- translating C to/from assembly
- writing programs that respect the limitations of computer arithmetic
- finding bottlenecks and improving runtime performance
- working effectively in a Unix development environment

and have **exposure** to

- a working understanding of the basics of cache memories

MIT CSAIL
@MIT_CSAIL

"Programming is like cooking: in Python, you use pre-made bolognese sauce; in C++, you start from fresh tomatoes and minced meat; in Assembly, you have a farm where you grow your tomatoes and raise your cow." - @gv_barroso

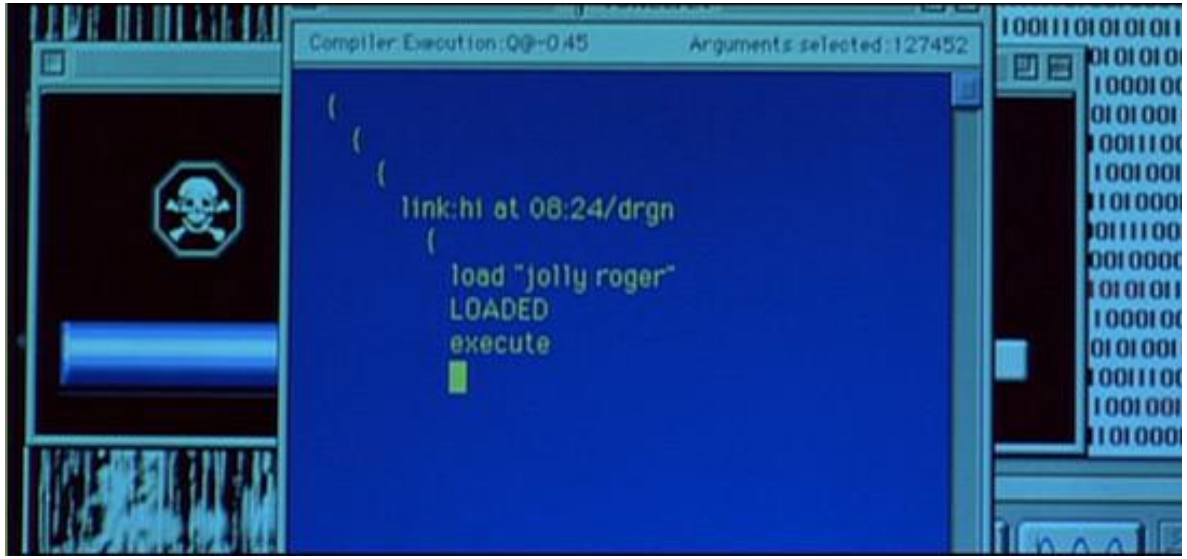
h/t @programmerwisdom

#tuesdaythoughts

5:28 PM · Sep 29, 2020 · TweetDeck

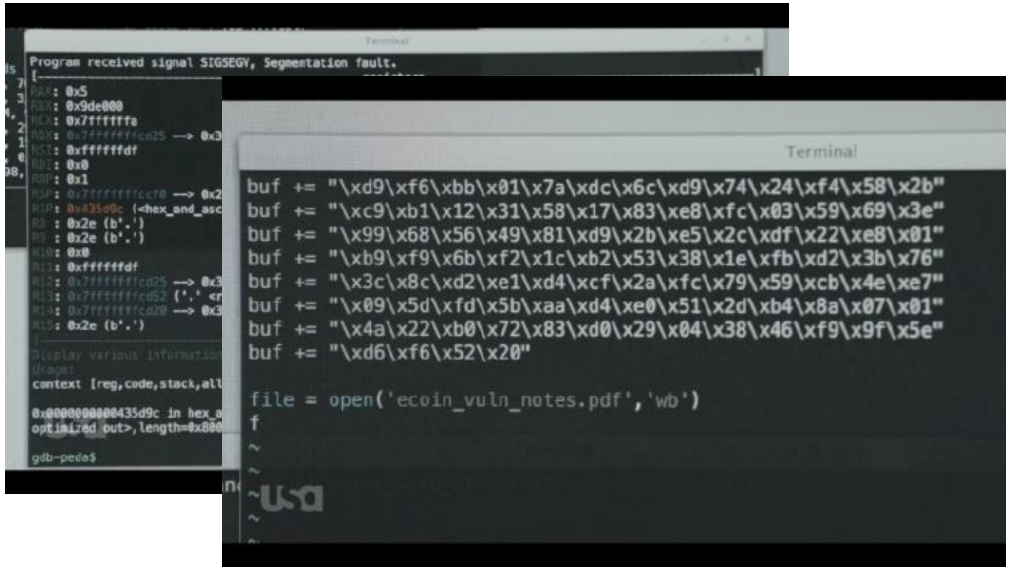
COMP201 Learning Goals

(also learn to identify legitimate programmer scenes in Hollywood movies)



```
(
  (
    (
      link:hi at 08:24/drgn
        (
          load "jolly roger"
          LOADED
          execute
```

Jeff Goldblum's character saving the world by uploading a virus to the alien mothership
Independence Day, 1996
(Directed by Roland Emmerich)



```
Program received signal SIGSEGV, Segmentation fault.
```

```
buf += "\xd9\xf6\xbb\x01\x7a\xdc\x6c\xd9\x74\x24\xf4\x58\x2b"
buf += "\xc9\xb1\x12\x31\x58\x17\x83\xe8\xfc\x03\x59\x69\x3e"
buf += "\x99\x68\x56\x49\x81\xd9\x2b\xe5\x2c\xdf\x22\xe8\x01"
buf += "\xb9\xf9\x6b\xf2\x1c\xb2\x53\x38\x1e\xfb\xd2\x3b\x76"
buf += "\x3c\x8c\xd2\xe1\xd4\xcf\x2a\xfc\x79\x59\xcb\x4e\xe7"
buf += "\x09\x5d\xfd\x5b\xaa\xd4\xe0\x51\x2d\xb4\x8a\x07\x01"
buf += "\x4a\x22\xb0\x72\x83\xd0\x29\x04\x38\x46\xf9\x9f\x5e"
buf += "\xd6\xf6\x52\x20"

file = open('ecoin_vuln_notes.pdf','wb')
f
~
~
~
~
```

Elliot creating a malicious PDF file, which contains some sort of shellcode that will allow him to take over any Linux computer that opens that file in Evince.
Mr. Robot, S3, Ep9 - eps3.8_stage3.torrent (2017)

Source: https://www.geekwire.com/2017/mr-robot-rewind-pwning-dark-army-episode-9/

Course Overview

1. **Bits and Bytes** - *How can a computer represent integer numbers?*
2. **Chars and C-Strings** - *How can a computer represent and manipulate more complex data like text?*
3. **Pointers, Stack and Heap** – *How can we effectively manage all types of memory in our programs?*
4. **Generics** - *How can we use our knowledge of memory and data representation to write code that works with any data type?*
5. **Assembly** - *How does a computer interpret and execute C programs?*
6. **The Memory Hierarchy** - *How does the memory system is organized as a hierarchy of different storage devices with unique capacities*
7. **The Heap Allocators** - *How do core memory-allocation operations like malloc and free work?*

Teaching Team

Aykut Erdem

Ali Kerem Bozkurt

Burak Kizil

Enes Şanlı

Deniz Bilge Akkoç

Course Website

https://aykuterdem.github.io/classes/comp201.s26/

*lecture videos on Panopto – can be accesses through KUHub Learn or from the course webpage

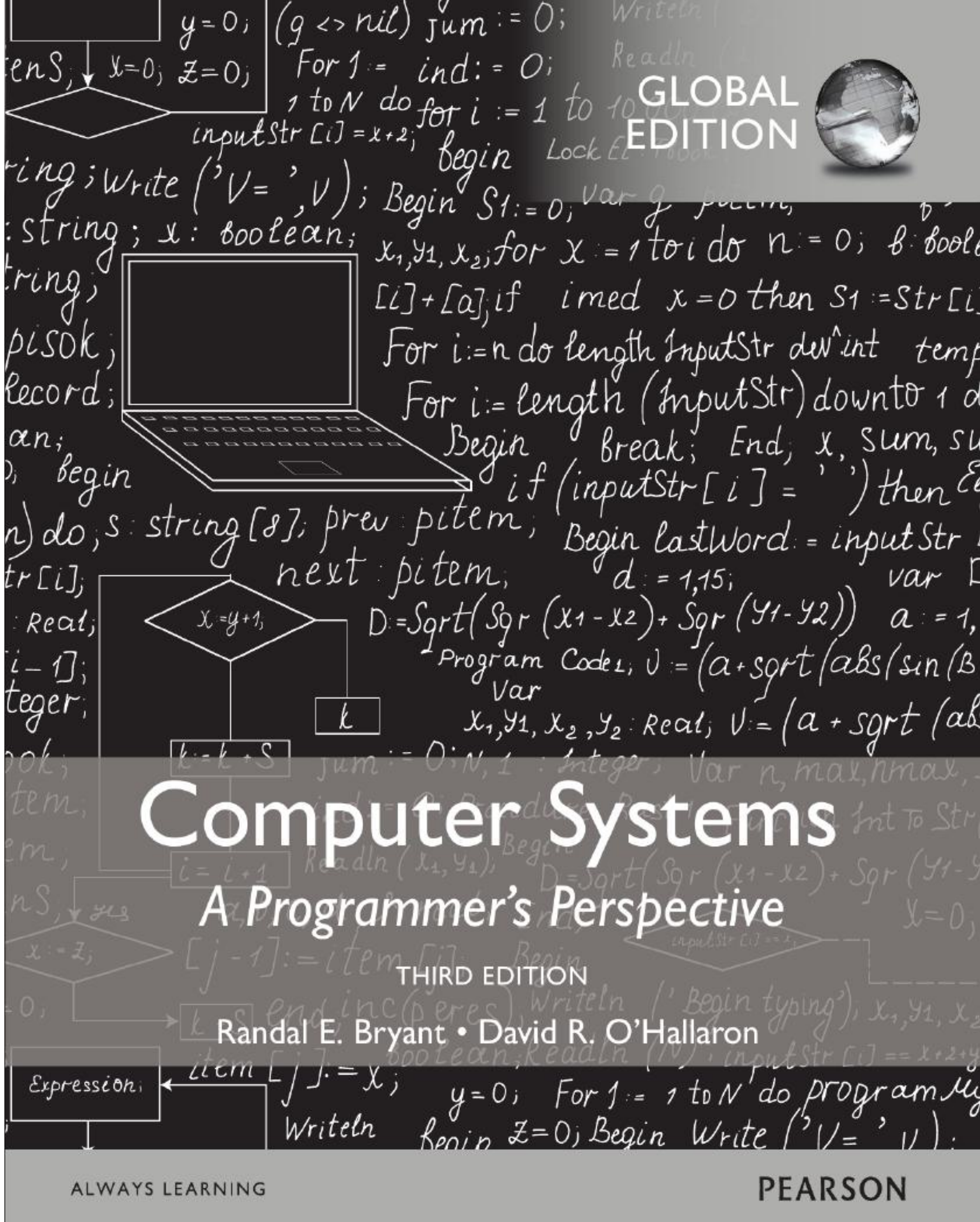
Question Break!

Lecture Plan

- Introduction
- COMP201 Course Policies
- Unix and the Command Line
- Getting Started With C

Textbooks

- *Computer Systems: A Programmer's Perspective* by Bryant & O'Hallaron, **3rd Edition**
  - **3rd edition matters** – important updates to course materials
- A C programming reference of your choice
  - *The C Programming Language* by Kernighan and Ritchie
  - Other C programming books, websites, or reference sheets



Course Structure

- **Lectures:** understand concepts, see demos
- **Labs:** learn tools, study code, discuss with peers
- **Assignments:** build programming skills, synthesize lecture/lab content

Tuesday	Thursday	Friday
Lecture	Lecture	Lab-A-B

- **assg0**: out next week, due Feb 24
- **C bootcamp**: this week (details will be announced soon)

- Lecture recordings will be released roughly 2 weeks after the lecture date.

Grading

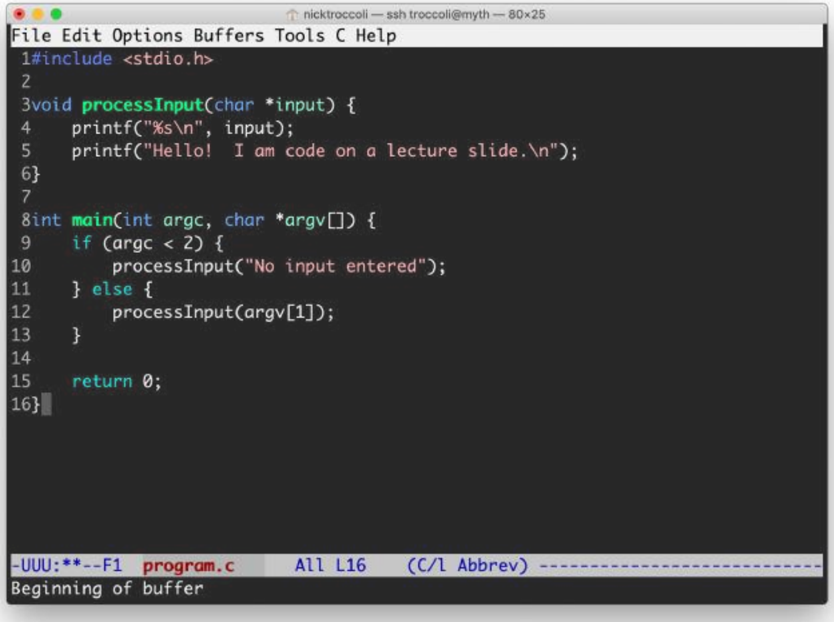
17%	5 Programming assignments
10%	7 Labs *(-2 from last semester)*
10%	2 Lab exams *(new this semester)*
28%	Midterm exam
30%	Final exam
5%	Class participation

Grading

17%	5 Programming assignments
10%	7 Labs
10%	2 Lab exams
28%	Midterm exam
30%	Final exam
5%	Class participation

Assignments

- 5 programming assignments completed individually using **Unix command line tools**
  - Free software, pre-installed on `linuxpool` cluster dedicated to COMP students
  - GitHub Classroom
  - We will give out starter projects for each assignment
- Graded on **functionality** (behavior) and **style** (elegance)
  - Functionality graded using automated tools, given as point score
  - Style graded via automated tests and TA code review,
  - Grades returned via KUHub Learn


nicktroccoli — ssh troccoli@myth — 80×25
File Edit Options Buffers Tools C Help

```
#include <stdio.h>

void processInput(char *input) {
    printf("%s\n", input);
    printf("Hello!  I am code on a lecture slide.\n");
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        processInput("No input entered");
    } else {
        processInput(argv[1]);
    }

    return 0;
}
```

```
-UUU:**--F1  program.c      All L16     (C/l Abbrev) ---------------------------
Beginning of buffer
```

Late Policy

- **Start out with 7 grace days**: each late day allows you to submit an assignment without penalty if you have free grace days left.
- **Hard deadline:** No submissions will be accepted **48 hours** after the original due date of an assignment (<u>regardless of grace days used</u>!)
- Penalty per day after grace days are exhausted
  - 1 day: 20% off
  - 2 days: 40% off

- Submissions made on KuHub Learn at 00:01am after the deadline counts as late and are considered as using 1 grace day

Grading

17%	5 Programming assignments
10%	7 Labs
10%	2 Lab exams
28%	Midterm exam
30%	Final exam
5%	Class participation

Lab Sections

- Weekly 100-minute labs led by a TA, starting next Friday.
- Hands-on practice with lecture material and course topics.

KUHub Learn + `linuxpool.ku.edu.tr`

- Graded on attendance + participation *(verified by submitting lab work)*
  - Two graded part:
    - Pre-lab problem (40%)
    - In-lab practice problem (60%)
  - Your lowest 2 scores will be dropped, hence there will be no make-up

Grading

17%	5 Programming assignments
10%	7 Labs
10%	2 Lab exams
28%	Midterm exam
30%	Final exam
5%	Class participation

Lab Exams

- To support meaningful learning and individual skill assessment, the course will include two in-person lab exams: one mid-semester and one near the end.
- Exams will mirror lab exercises and programming assignments, focusing on applied use of practiced concepts and tools.
- Further details on format, timing, and scope will be shared well in advance.

Grading

17%	5 Programming assignments
10%	7 Labs
10%	2 Lab exams
28%	Midterm exam
30%	Final exam
5%	Class participation

Midterm and Final Exams

- Pen and paper exams
  - Midterm Exam: Date and time will be announced later
  - Final Exam: Date and time will be announced later

- You can only take a make-up for either the midterm or the final exam, but not both!

Grading

17%	5 Programming assignments
10%	7 Labs
10%	2 Lab exams
28%	Midterm exam
30%	Final exam
5%	Class participation

Class participation

- 2.5% Attendance
- 2.5% Actively participating in-class discussions

! The students are expected to attend at least 70% of the lectures.

What foreign TV series are you watching now? Just specify the one that you are enjoying the most?

Question Break!

Getting Help

- Post on the **Discussion Forum at KUHub Learn**
  - Online discussion forum for students; post questions, answer other students' questions
  - Best for general assignment questions (DON'T POST ASSIGNMENT CODE!)
- Visit **Office Hours**
  - More info to come soon!
- **Email** the Course Staff
  - Best for **private matters** (e.g. grading questions).

Koç University Honor Code

- For assignments students should be required to digitally add and approve a version of the agreement below.

> *I hereby declare that I have completed this examination individually, without support from anyone else.*
>
> *I hereby accept that only the below-listed sources are approved to be used during this open-source examination:*
>
> *(i) Coursebook,*
>
> *(ii) All material that is made available to students via KUHub Learn for this course,*
>
> *(iii) Notes taken by me during lectures.*
>
> *I have not used, accessed or taken any unpermitted information from any other source. Hence, all effort belongs to me.*

Honor Code and COMP201

- Please help us ensure academic integrity:
  - Indicate any assistance received on HW (books, friends, etc.).
  - Do not look at other people's solution code or answers
  - Do not give your solutions to others or post them on the web or to the forum.
  - Report any inappropriate activity you see performed by others.
- Assignments are checked regularly for similarity with help of automated software tools.
- If you realize that you have made a mistake, you may retract your submission to any assignment at any time, no questions asked. Come to use before we come for you.
- If you need help, please contact us and we will help you.
  - We do not want you to feel any pressure to violate the Honor Code in order to succeed in this course.

Use of Generative AI Tools

- Although AI tools can be useful, they may **limit your growth** by reducing the benefits of actively working through challenges.
- You may use AI tools **only as you would seek help from a classmate**, to ask broad, high-level questions or for general guidance, with proper citation if applicable.
- **Do not use AI tools** to write code, generate responses, or complete any part of **graded assignments**.
- **Do not input your code** into AI tools for feedback or debugging help. This is considered a **violation of KU Honor Code**.
- Instead, make use of the **course's official support resources.** The teaching team is ready and eager to assist you.

Use of Generative AI Tools

- Although AI tools can be useful, they may **limit your growth** by reducing the benefits of actively working through challenges.
- You n **te**, to
  ask b r
  citati
- **Do no** any
  part o
- **Do no** p.
  This is considered a **violati** **U Honor Code.**
- Instead, make use of the **course's official support resources.** The teaching team is ready and eager to assist you.

This semester, we will teach you some GenAI tools you can use, but approach them wisely!

Poll Time

Do not edit
How to change the design

What is your favorite programming language?

ⓘ The Slido app must be installed on every computer you’re presenting from

Do not edit
How to change the design

How often do you use the command line (i.e. Mac OS X Terminal, Linux shell or may be Windows PowerShell)?

ⓘ The Slido app must be installed on every computer you're presenting from

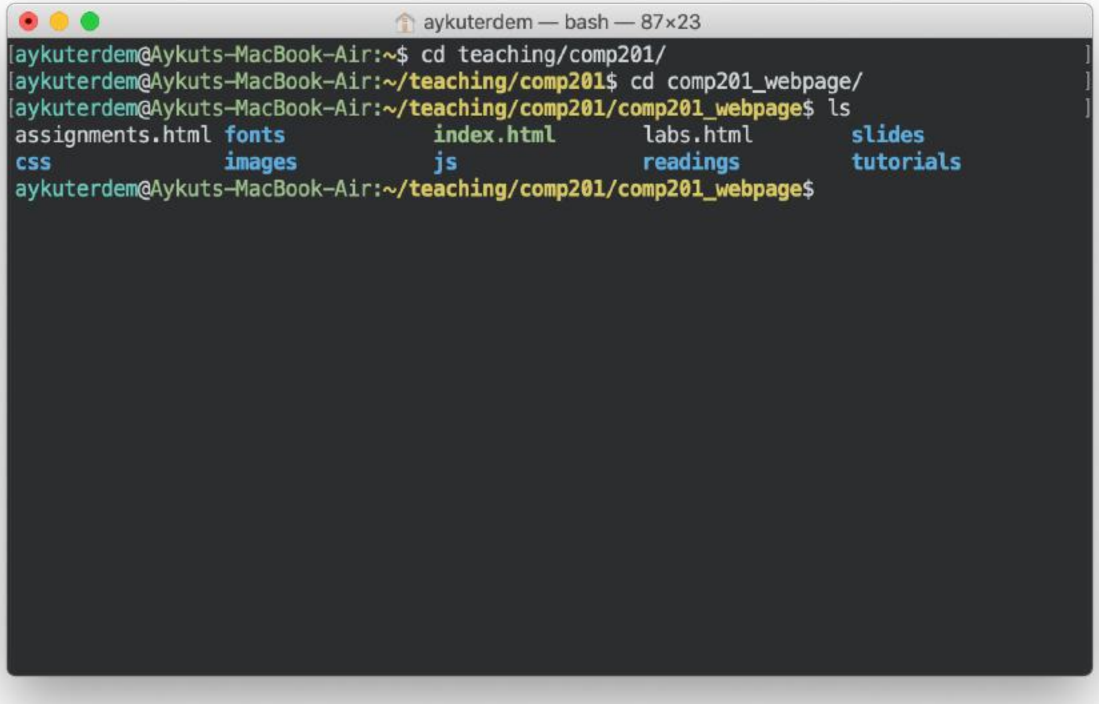
slido

Lecture Plan

- Introduction
- COMP201 Course Policies
- Unix and the Command Line
- Getting Started With C

What is Unix?

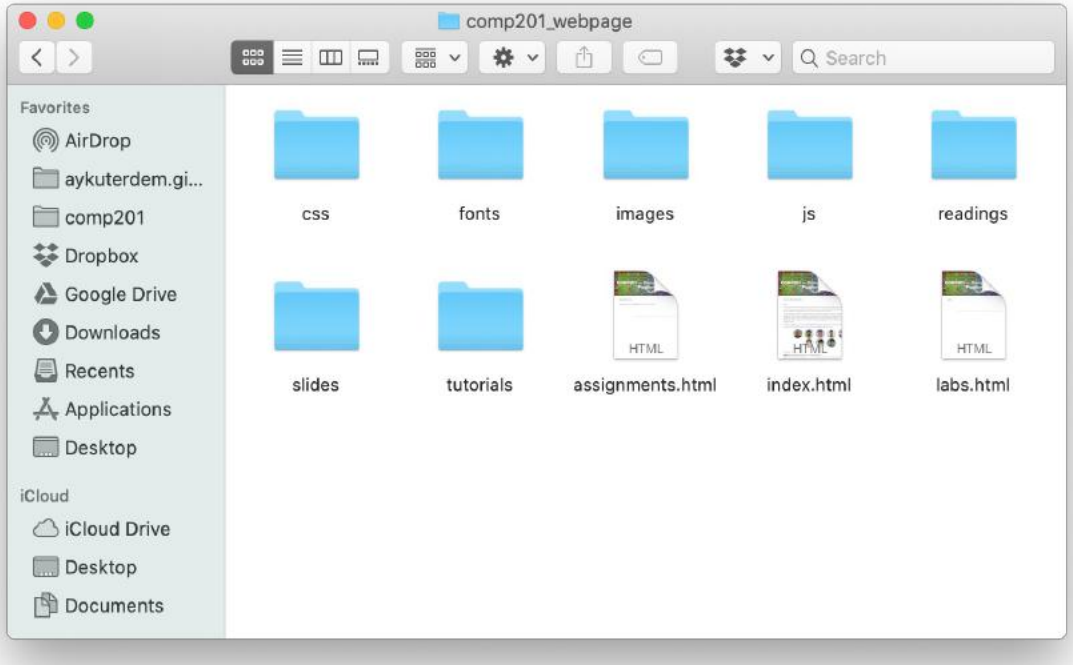
- **Unix:** a set of standards and tools commonly used in software development.
  - **macOS** and **Linux** are operating systems built on top of Unix
- You can navigate a Unix system using the **command line** (“terminal”)
- Every Unix system works with the same tools and commands



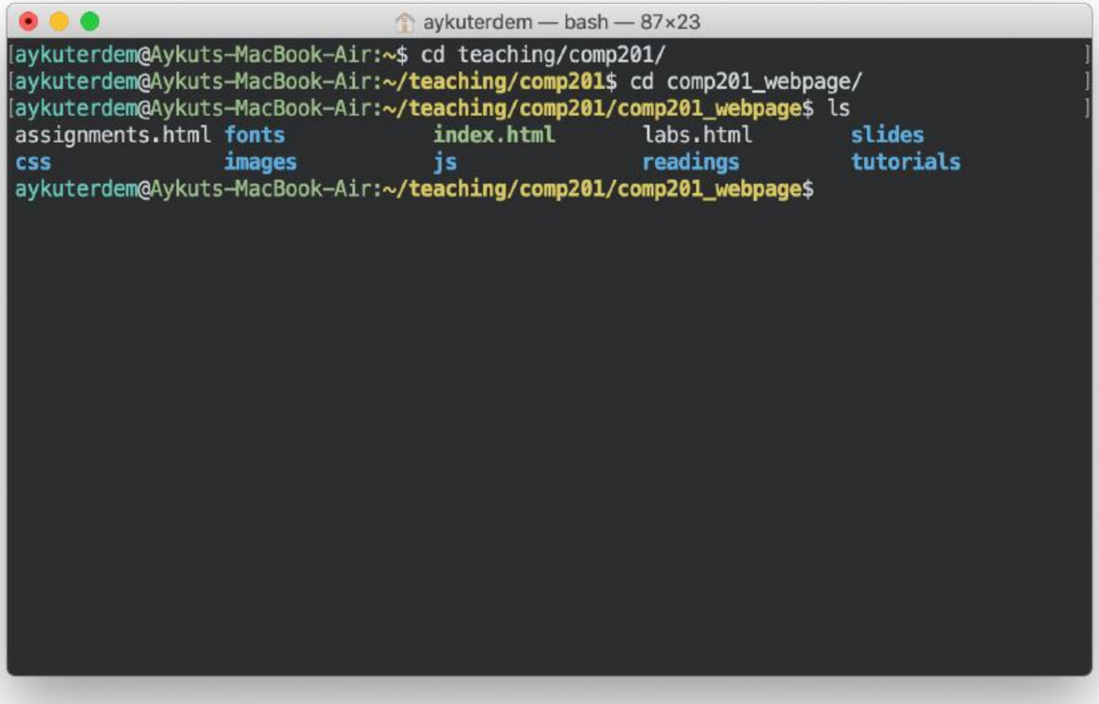
```
aykuterdem@Aykuts-MacBook-Air:~$ cd teaching/comp201/
aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201$ cd comp201_webpage/
aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201/comp201_webpage$ ls
assignments.html fonts            index.html       labs.html        slides
css              images           js               readings         tutorials
aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201/comp201_webpage$
```

What is the Command Line?

- The **command-line** is a text-based interface (i.e., **terminal** interface) to navigate a computer, instead of a Graphical User Interface (GUI).



Graphical User Interface



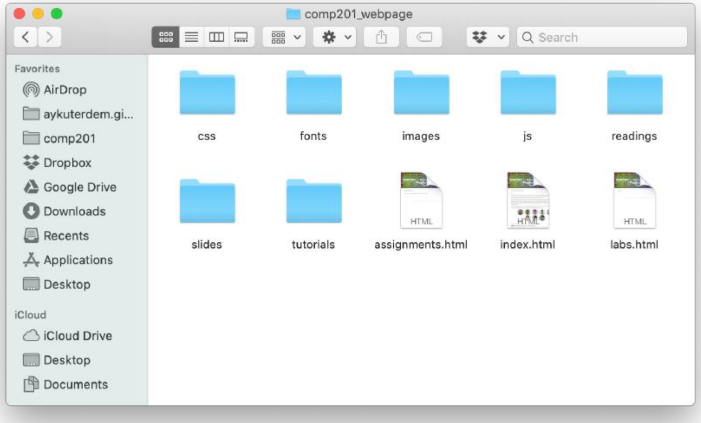
```
[aykuterdem@Aykuts-MacBook-Air:~$ cd teaching/comp201/
[aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201$ cd comp201_webpage/
[aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201/comp201_webpage$ ls
assignments.html fonts           index.html      labs.html       slides
css              images          js              readings        tutorials
aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201/comp201_webpage$
```

Text-based interface

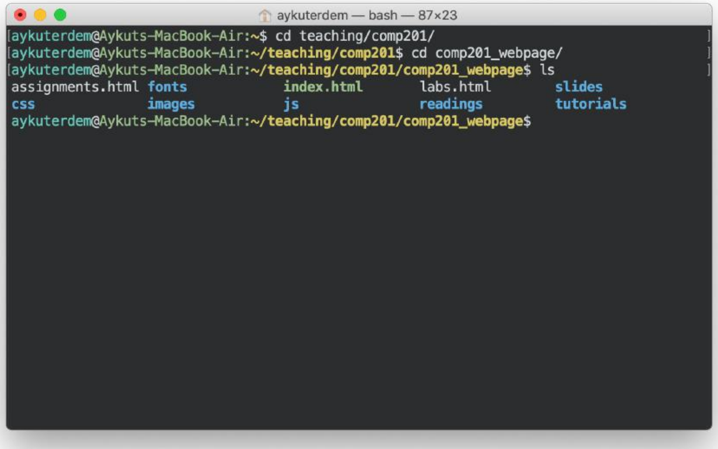
Command Line vs. GUI

Just like a GUI file explorer interface, a terminal interface:

- shows you a **specific place** on your computer at any given time.
- lets you go **into folders** and **out of folders**.
- lets you **create new** files and **edit** files.
- lets you **execute programs**.



Graphical User Interface



```
aykuterdem@Aykuts-MacBook-Air:~$ cd teaching/comp201/
aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201$ cd comp201_webpage/
aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201/comp201_webpage$ ls
assignments.html fonts            index.html       labs.html        slides
css              images           js               readings         tutorials
aykuterdem@Aykuts-MacBook-Air:~/teaching/comp201/comp201_webpage$
```

Command-line interface

Why Use Unix / the Command Line?

- You can navigate almost any device using the same tools and commands:
  - Servers
  - Laptops and desktops
  - Embedded devices (Raspberry Pi, etc.)
  - Mobile Devices (Android, etc.)
- Used frequently by software engineers:
  - **Web development:** running servers and web tools on servers
  - **Machine learning:** processing data on servers, running algorithms
  - **Systems**: writing operating systems, networking code and embedded software
  - **Mobile Development**: running tools, managing libraries
  - And more...
- We'll use Unix and the command line to implement and execute our programs.

Demo: Using Unix and the Command Line

Unix Commands Recap

- **`cd`** – change directories (..)
- **`ls`** – list directory contents
- **`mkdir`** – make directory
- **`emacs`** – open text editor
- **`vi`** – open text editor
- **`rm`** – remove file or folder
- **`man`** – view manual pages

Lab 1:
The Linux Shell
(*next week*)

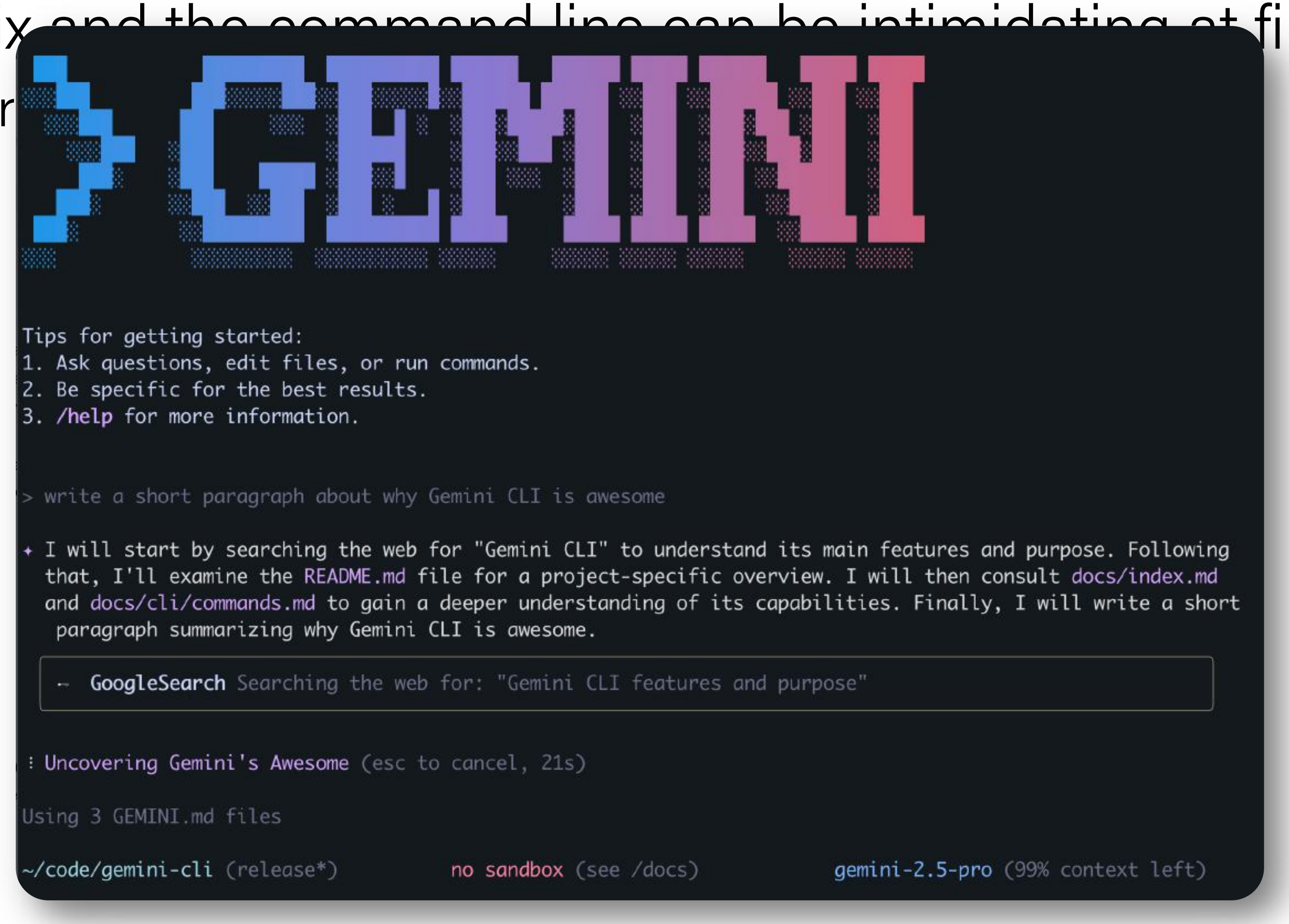
See the Resources page of the course website for more commands, and a complete reference.

Learning Unix and the Command Line

- Using Unix and the command line can be intimidating at first:
  - It looks retro!
  - How do I know what to type?

Learning Unix and the Command Line

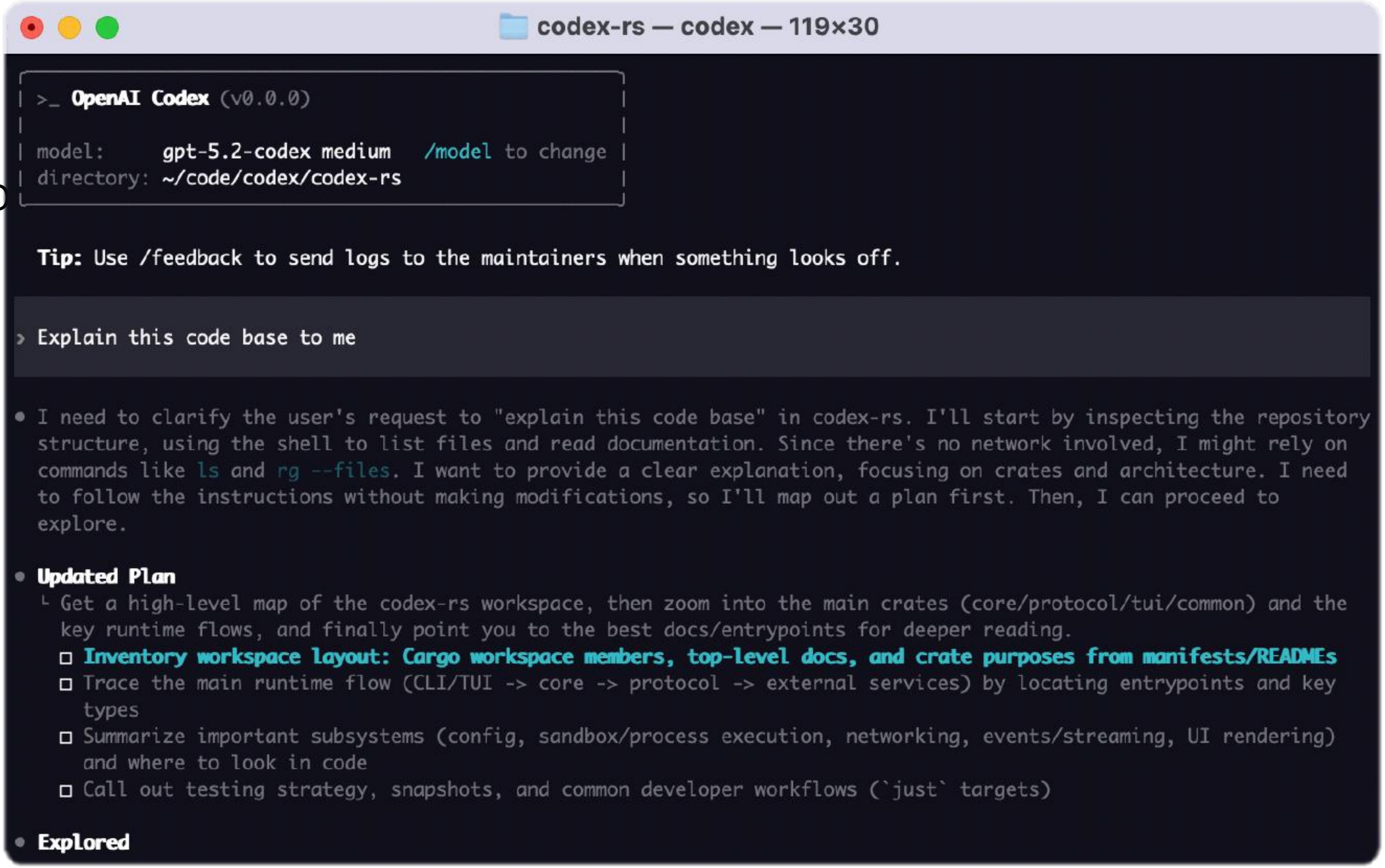
- Using Unix and the command line can be intimidating at first:
  - It looks r
  - How do



```
Tips for getting started:
1. Ask questions, edit files, or run commands.
2. Be specific for the best results.
3. /help for more information.


> write a short paragraph about why Gemini CLI is awesome

✦ I will start by searching the web for "Gemini CLI" to understand its main features and purpose. Following
  that, I'll examine the README.md file for a project-specific overview. I will then consult docs/index.md
  and docs/cli/commands.md to gain a deeper understanding of its capabilities. Finally, I will write a short
   paragraph summarizing why Gemini CLI is awesome.

 ┌──────────────────────────────────────────────────────────────────────────────────────────┐
 │ ⊶  GoogleSearch Searching the web for: "Gemini CLI features and purpose"                 │
 └──────────────────────────────────────────────────────────────────────────────────────────┘

⠇ Uncovering Gemini's Awesome (esc to cancel, 21s)

Using 3 GEMINI.md files

~/code/gemini-cli (release*)          no sandbox (see /docs)          gemini-2.5-pro (99% context left)
```

Learning Unix and the Command Line

- Using Un
  - It looks
  - How do



codex-rs — codex — 119×30

```
╭──────────────────────────────────────────────────╮
│ >_ OpenAI Codex (v0.0.0)                         │
│                                                  │
│ model:     gpt-5.2-codex medium   /model to change │
│ directory: ~/code/codex/codex-rs                 │
╰──────────────────────────────────────────────────╯

  Tip: Use /feedback to send logs to the maintainers when something looks off.

› Explain this code base to me

• I need to clarify the user's request to "explain this code base" in codex-rs. I'll start by inspecting the repository
  structure, using the shell to list files and read documentation. Since there's no network involved, I might rely on
  commands like ls and rg --files. I want to provide a clear explanation, focusing on crates and architecture. I need
  to follow the instructions without making modifications, so I'll map out a plan first. Then, I can proceed to
  explore.

• Updated Plan
  └ Get a high-level map of the codex-rs workspace, then zoom into the main crates (core/protocol/tui/common) and the
    key runtime flows, and finally point you to the best docs/entrypoints for deeper reading.
    □ Inventory workspace layout: Cargo workspace members, top-level docs, and crate purposes from manifests/READMEs
    □ Trace the main runtime flow (CLI/TUI -> core -> protocol -> external services) by locating entrypoints and key
      types
    □ Summarize important subsystems (config, sandbox/process execution, networking, events/streaming, UI rendering)
      and where to look in code
    □ Call out testing strategy, snapshots, and common developer workflows (`just` targets)

• Explored
```

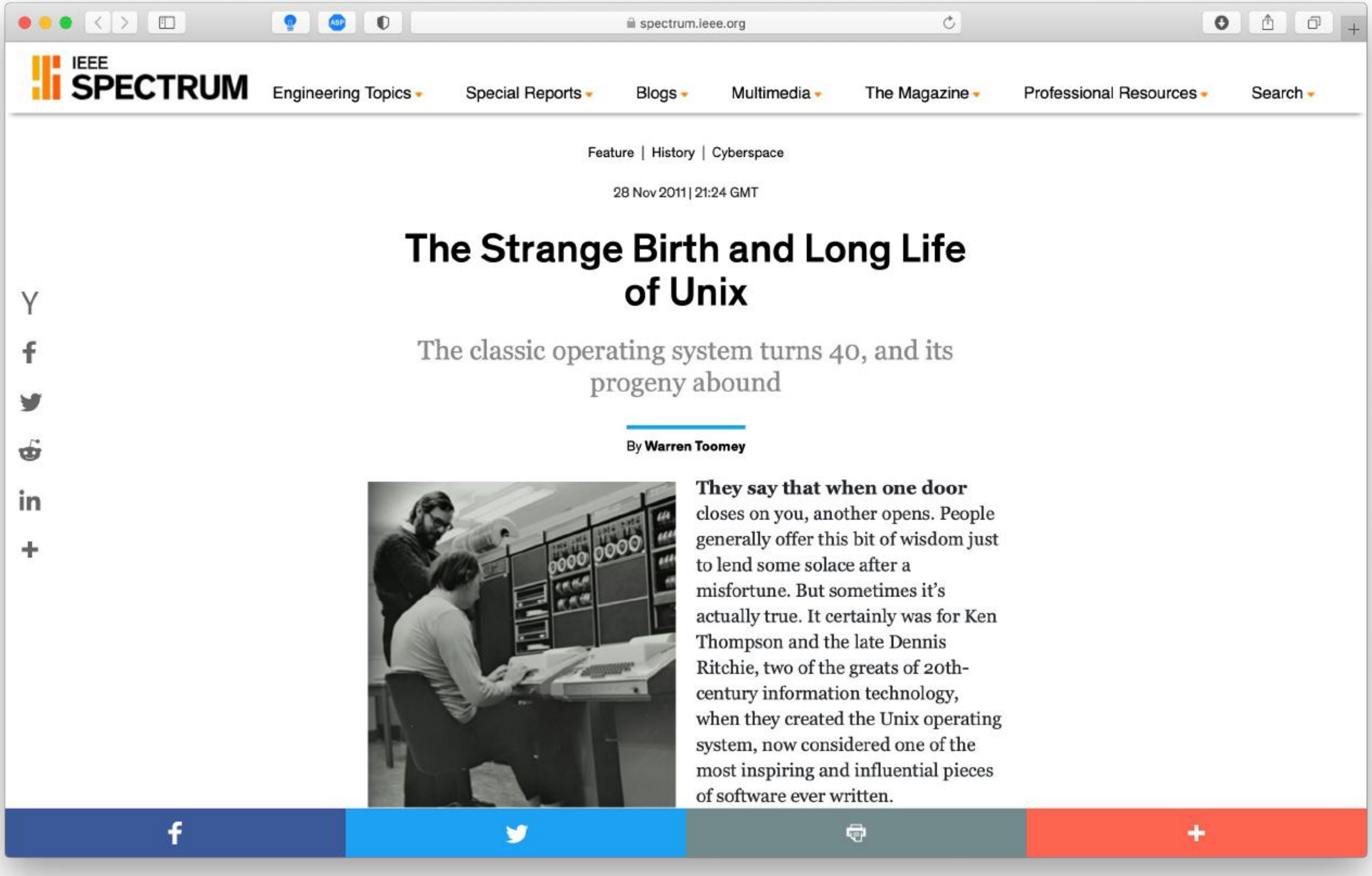
Learning Unix and the Command Line

- Using Unix and the command line can be intimidating at first:
  - It looks retro!
  - How do I know what to type?

- It's like learning a new language:
  - At first, you may have to constantly look things up (**Resources** page on course website!)
  - It's important to spend as much time as possible (during labs and assignments) building muscle memory with the tools

Question Break!

Additional Reading 1



Feature | History | Cyberspace

28 Nov 2011 | 21:24 GMT

The Strange Birth and Long Life of Unix

The classic operating system turns 40, and its progeny abound

By **Warren Toomey**

They say that when one door closes on you, another opens. People generally offer this bit of wisdom just to lend some solace after a misfortune. But sometimes it's actually true. It certainly was for Ken Thompson and the late Dennis Ritchie, two of the greats of 20th-century information technology, when they created the Unix operating system, now considered one of the most inspiring and influential pieces of software ever written.

https://spectrum.ieee.org/tech-history/cyberspace/the-strange-birth-and-long-life-of-unix

Lecture Plan

- Introduction
- COMP201 Course Policies
- Unix and the Command Line
- Getting Started With C

The C Language

C was created around 1970 to make writing Unix and Unix tools easier.

- Part of the C/C++/Java family of languages (C++ and Java were created later)
- Design principles:
  - Small, simple abstractions of hardware
  - Minimalist aesthetic
  - Prioritizes efficiency and minimalism over safety and high-level abstractions

C vs. C++ and Java

They all share:

- Syntax
- Basic data types
- Arithmetic, relational, and logical operators

C doesn't have:

- More advanced features like operator overloading, default arguments, pass by reference, classes and objects, ADTs, etc.
- Extensive libraries (no graphics, networking, etc.) – this means not much to learn C!
- many compiler and runtime checks (this may cause security vulnerabilities!)

Programming Language Philosophies

- **C is procedural:** you write functions, rather than define new variable types with classes and call methods on objects. C is small, fast and efficient.

- **C++ is procedural, with objects:** you write functions, and define new variable types with classes, and call methods on objects.

- **Python is also procedural, but dynamically typed:** you still write functions and call methods on objects, but the development process is very different.

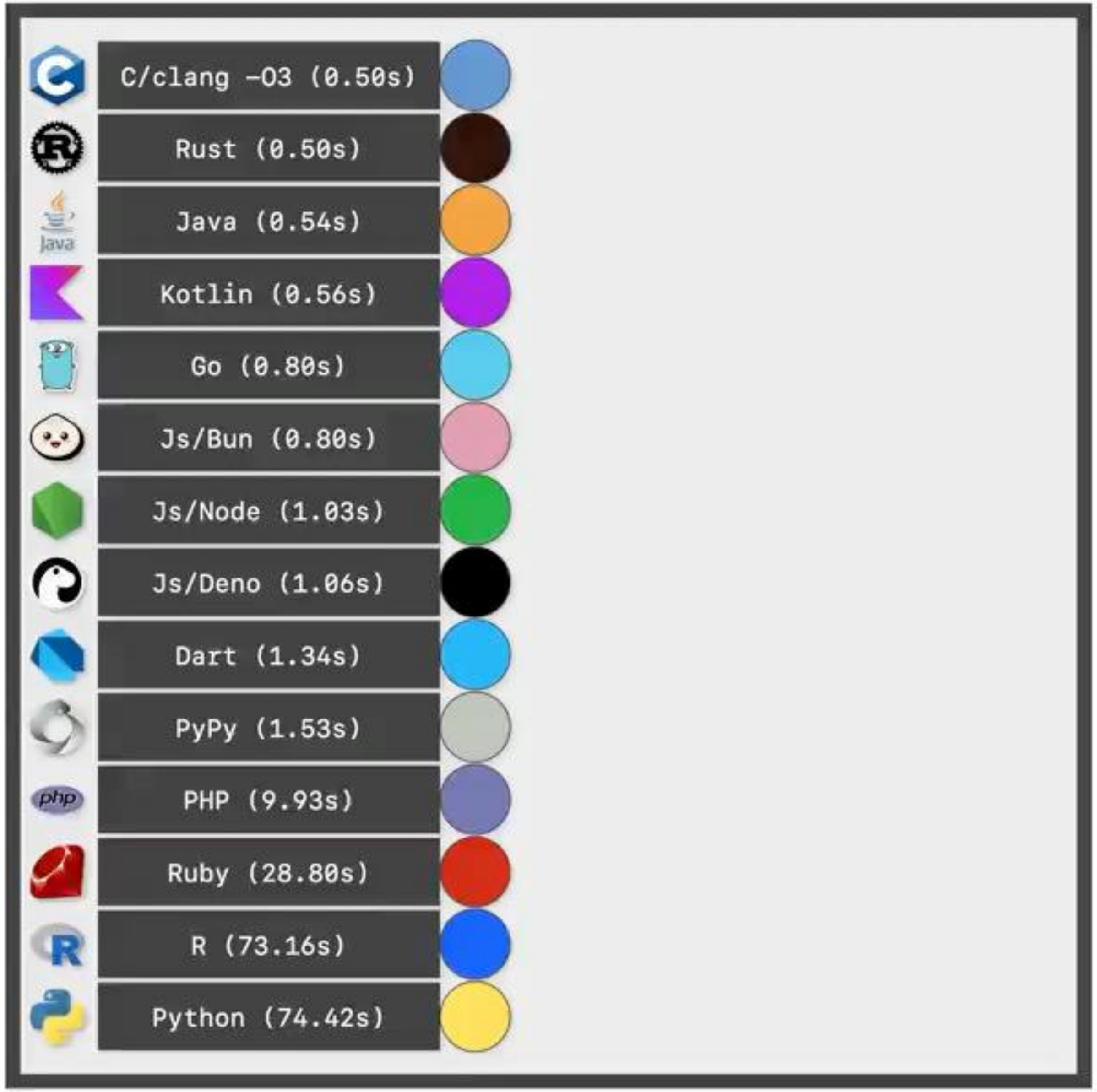
- **Java is object-oriented:** virtually everything is an object, and everything you write needs to conform to the object-oriented design pattern.

Why C?

- Many tools (and even other languages, like Python!) are built with C.
- C is the language of choice for fast, highly efficient programs.
- C is popular for systems programming (operating systems, networking, etc.)
- C lets you work at a lower level to manipulate and understand the underlying system.

Why C?

1 Billion nested loop iterations

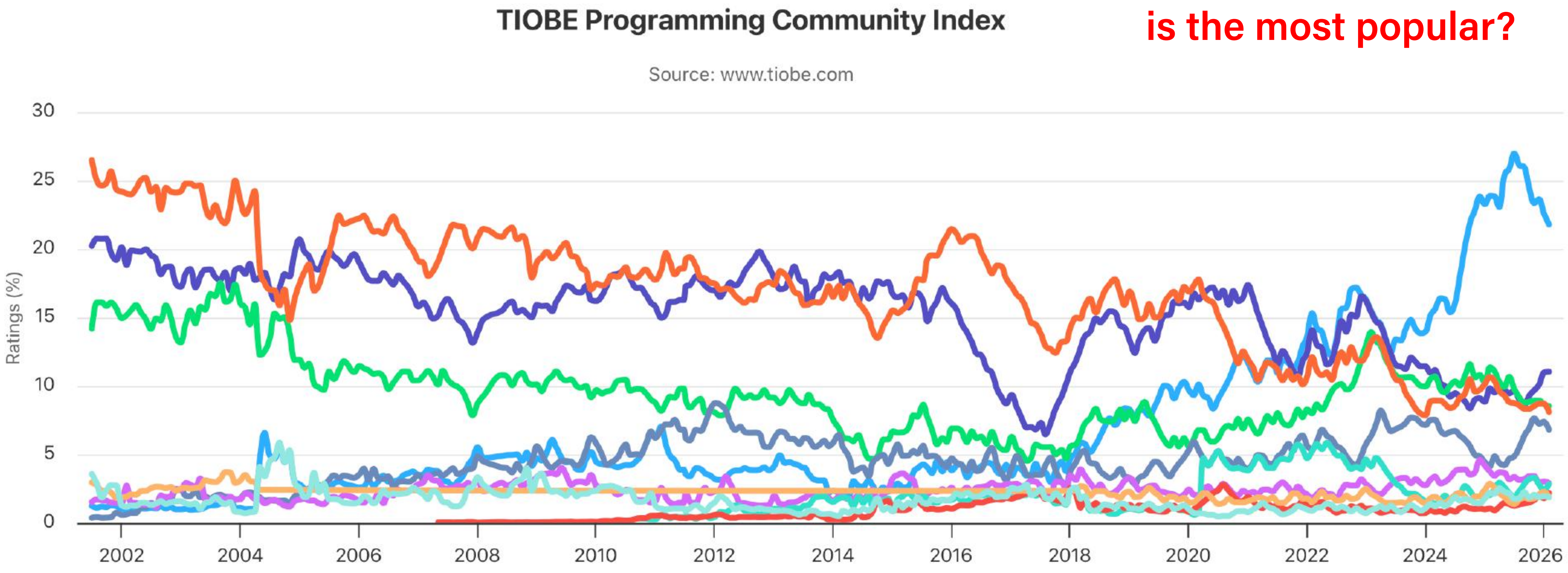


C/clang -O3 (0.50s)
Rust (0.50s)
Java (0.54s)
Kotlin (0.56s)
Go (0.80s)
Js/Bun (0.80s)
Js/Node (1.03s)
Js/Deno (1.06s)
Dart (1.34s)
PyPy (1.53s)
PHP (9.93s)
Ruby (28.80s)
R (73.16s)
Python (74.42s)

https://benjdd.com/languages/

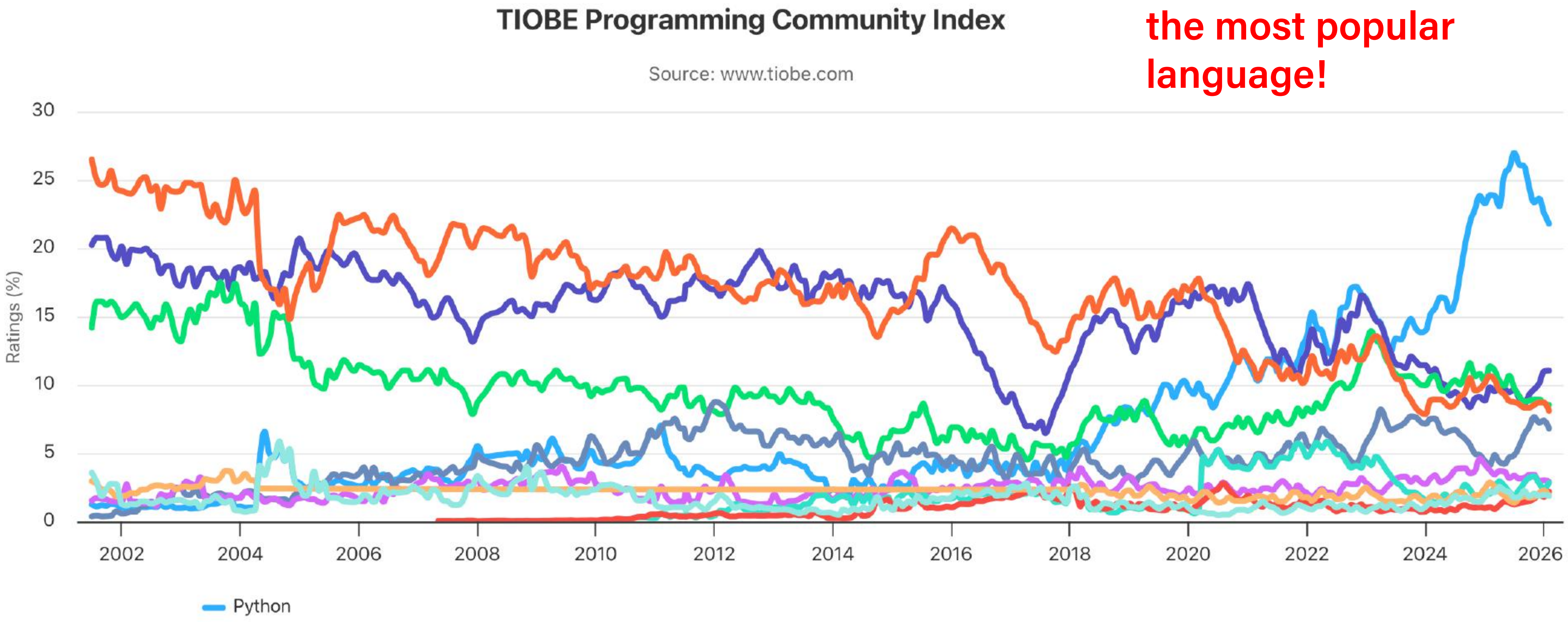
Programming Language Popularity

Guess which one is the most popular?



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

Programming Language Popularity

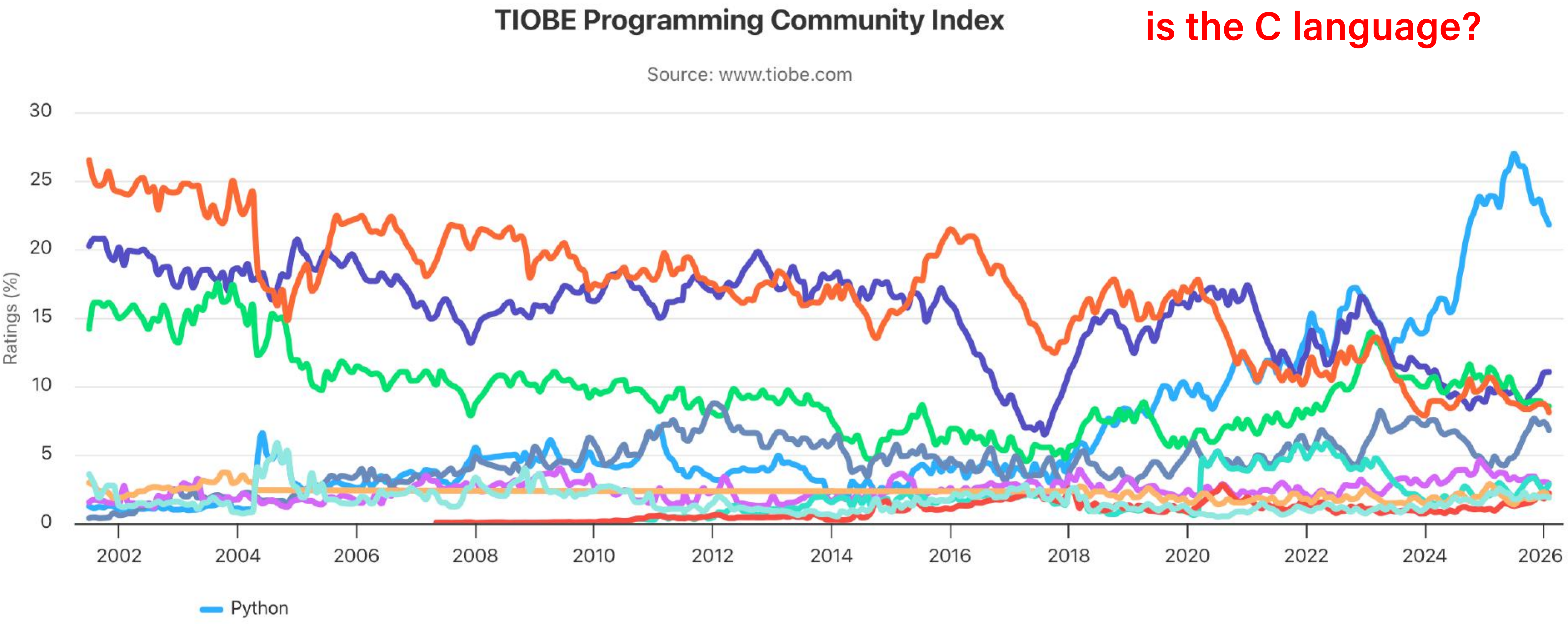


Currently, Python is the most popular language!

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

Programming Language Popularity

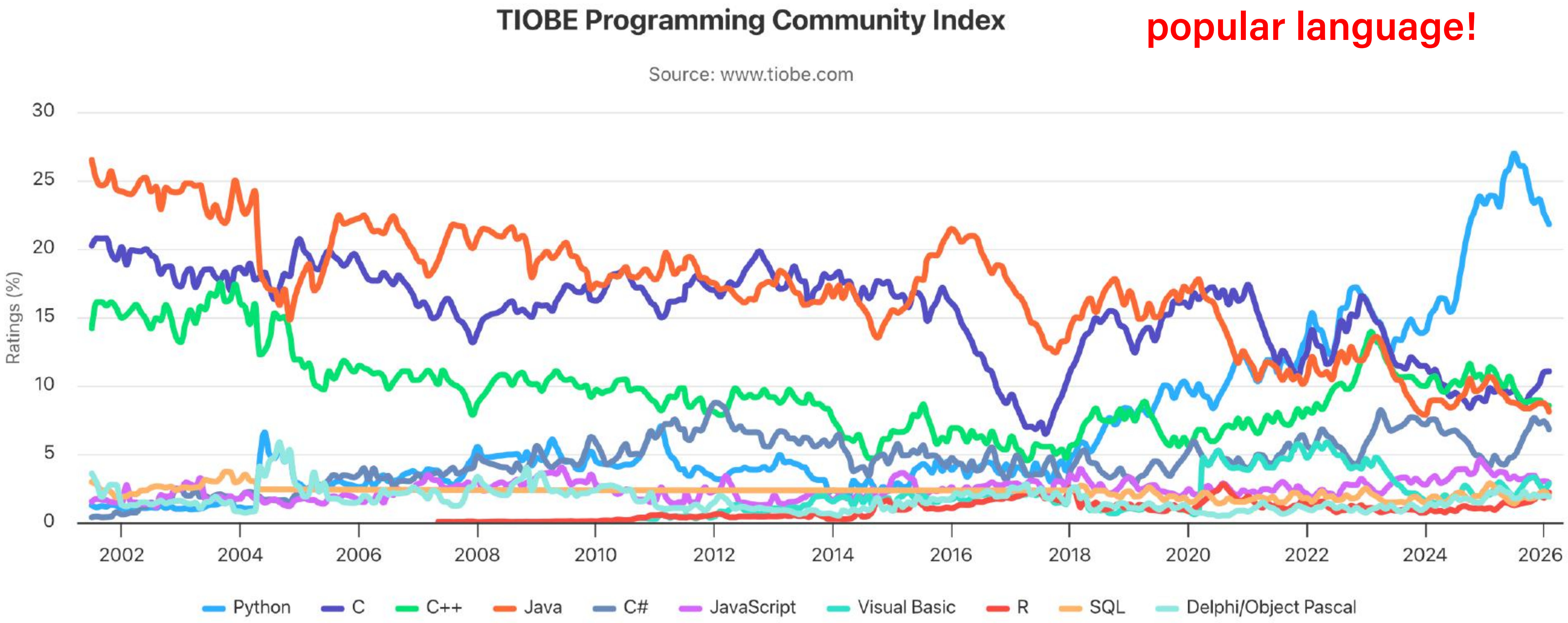
Guess which one is the C language?



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

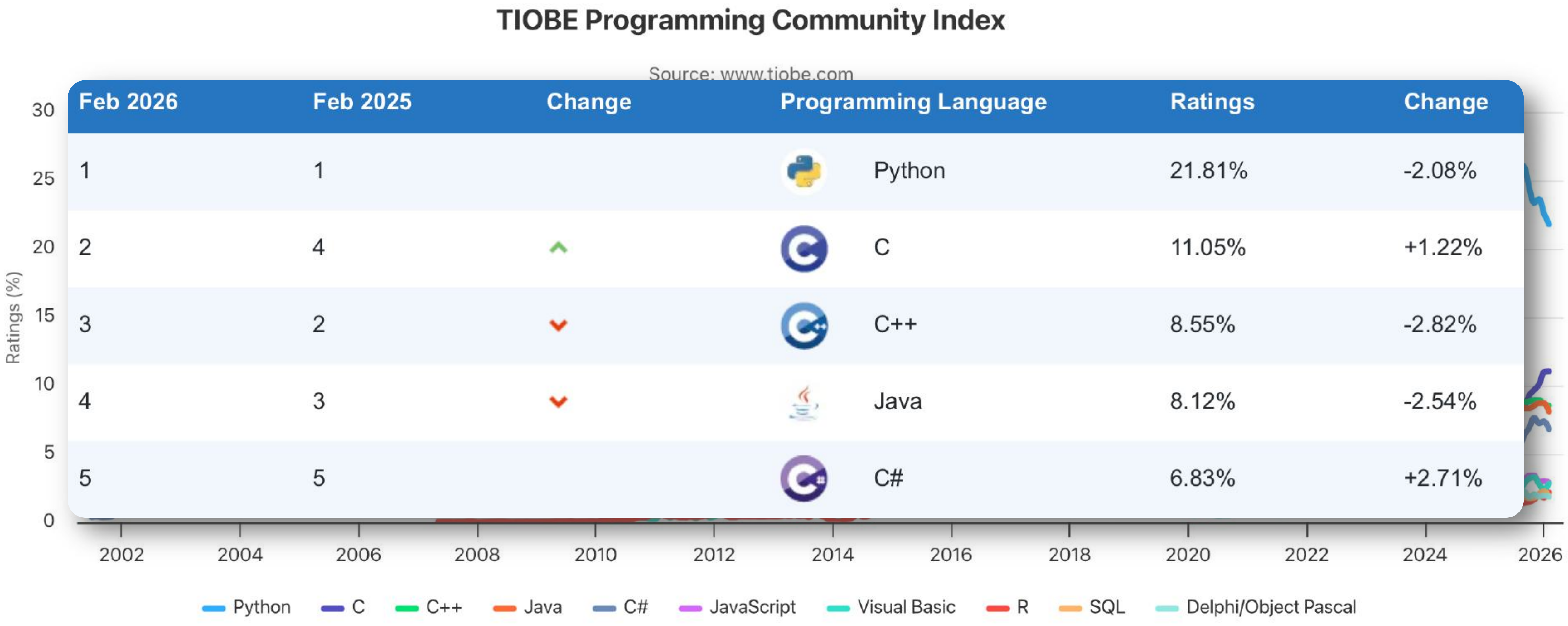
Programming Language Popularity

C is now the 2nd most popular language!



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

Programming Language Popularity



TIOBE Programming Community Index

Source: www.tiobe.com

Feb 2026	Feb 2025	Change	Programming Language	Ratings	Change
1	1		Python	21.81%	-2.08%
2	4	▲	C	11.05%	+1.22%
3	2	▼	C++	8.55%	-2.82%
4	3	▼	Java	8.12%	-2.54%
5	5		C#	6.83%	+2.71%

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

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>  // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

Program comments
You can write block or inline comments.

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>  // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

Import statements
C libraries are written with angle brackets.
Local libraries have quotes:
`#include "lib.h"`

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

main function – entry point for the program
Should always return an integer (0 = success)

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

Main parameters – `main` takes two parameters, both relating to the command line arguments used to execute the program.

`argc` is the number of arguments in `argv`
`argv` is an array of arguments (**`char *` is C string**)

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h>   // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

`printf` – prints output to the screen

Familiar Syntax

```
int x = 42 + 7 * -5;               // variables, types
double pi = 3.14159;
char c = 'Q';                      /* two comment styles */

for (int i = 0; i < 10; i++) {     // for loops
    if (i % 2 == 0) {              // if statements
        x += i;
    }
}

while (x > 0 && c == 'Q' || b) {   // while loops, logic
    x = x / 2;
    if (x == 42) { return 0; }
}

binky(x, 17, c);                  // function call
```

Boolean Variables

To declare Booleans, (e.g. **`bool b = ____`**), you must include **`stdbool.h`**:

```
#include <stdio.h>     // for printf
#include <stdbool.h>   // for bool

int main(int argc, char *argv[]) {
    bool x = 5 > 2 && binky(argc) > 0;
    if (x) {
        printf("Hello, world!\n");
    } else {
        printf("Howdy, world!\n");
    }
    return 0;
}
```

Boolean Expressions

C treats a nonzero value as true, and a zero value as false:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int x = 5;
    if (x) {   // true
       printf("Hello, world!\n");
    } else {
       printf("Howdy, world!\n");
    }
    return 0;
}
```

Console Output: `printf`

```
printf(text, arg1, arg2, arg3);
```

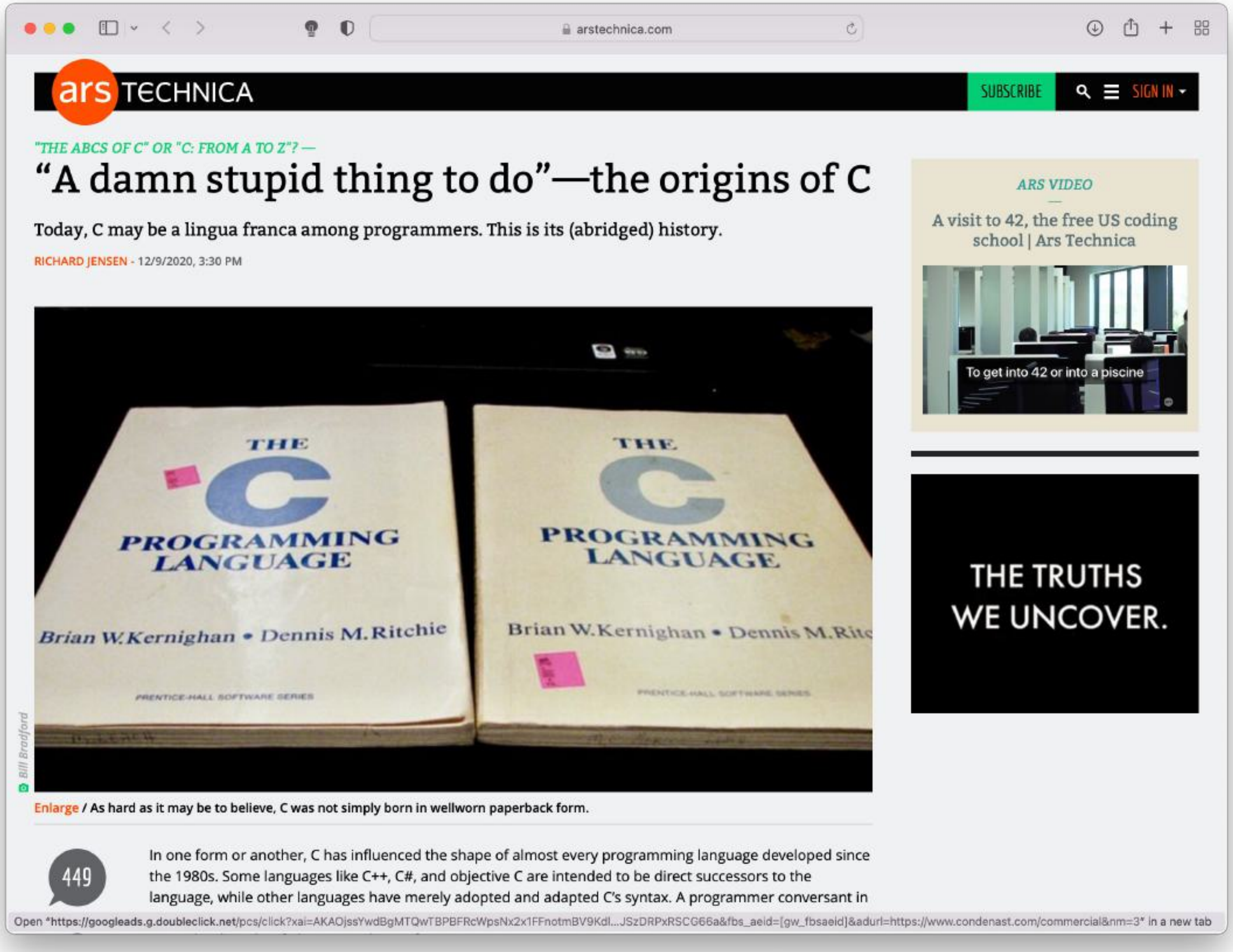
```
// Example
char *classPrefix = "COMP";
int classNumber = 201;
printf("You are in %s%d", classPrefix, classNumber);    // You are in COMP201
```

`printf` makes it easy to print out the values of variables or expressions.

If you include *placeholders* in your printed text, `printf` will replace each placeholder *in order* with the values of the parameters passed after the text.

`%s (string)`   `%d (integer)`   `%f (double)`

Additional Reading 2


arstechnica.com
ars TECHNICA
SUBSCRIBE
SIGN IN
Bill Bradford
THE C PROGRAMMING LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE-HALL SOFTWARE SERIES
449
THE TRUTHS WE UNCOVER.

"THE ABCS OF C" OR "C: FROM A TO Z"? —

“A damn stupid thing to do”—the origins of C

Today, C may be a lingua franca among programmers. This is its (abridged) history.

RICHARD JENSEN - 12/9/2020, 3:30 PM

Enlarge / As hard as it may be to believe, C was not simply born in wellworn paperback form.

In one form or another, C has influenced the shape of almost every programming language developed since the 1980s. Some languages like C++, C#, and objective C are intended to be direct successors to the language, while other languages have merely adopted and adapted C's syntax. A programmer conversant in

ARS VIDEO

A visit to 42, the free US coding school | Ars Technica

To get into 42 or into a piscine

Open "https://googleads.g.doubleclick.net/pcs/click?xai=AKAOjssYwdBgMTQwTBPBFRcWpsNx2x1FFnotmBV9Kdl...JSzDRPxRSCG66a&fbs_aeid=[gw_fbsaeid]&adurl=https://www.condenast.com/commercial&nm=3" in a new tab

https://arstechnica.com/features/2020/12/a-damn-stupid-thing-to-do-the-origins-of-c/

Question Break!

Writing, Debugging and Compiling

We will use:

- the **vi/emacs** text editor to write our C programs
- the **make** tool to compile our C programs
- the **gdb** debugger to debug our programs
- the **valgrind** tools to debug memory errors and measure program efficiency

Demo: Compiling And Running A C Program

`args.c`

Working On C Programs Recap

- **ssh** – remotely log in to `linuxpool` computers (*later*)
- **Vi/Emacs** – text editor to write and edit C programs
  - Use the mouse to position cursor, scroll, and highlight text
  - :w / Ctl-x Ctl-s to save, :q / Ctl-x Ctl-c to quit
- **make** – compile program using provided Makefile
- **./myprogram** – run executable program (optionally with arguments)
- **make clean** – remove executables and other compiler files
- Lecture codes are accessible at course webpage

Recap

- COMP201 is a programming class, which uses C to teach you about what goes on under the hood of programming languages and software.
- We'll use Unix and command line tools to write, debug and run our programs.
- Please regularly visit the course website, https://aykuterdem.github.io/classes/comp201.s26 and follow the announcements on Blackboard.
- We're looking forward to an exciting semester!

Next time: How a computer represents integer numbers? What are the limitations?