CS107, Lecture 1

Welcome to CS107!

Reading:

Course Syllabus

Bryant & O'Hallaron, Ch. 1 (skim – available on Canvas)

Honor Code and Collaboration Page

Based on slides created by Nick Troccoli, Chris Gregg

Plan For Today

- Introduction + Syllabus
- Unix
- C 101 (please review the skipped slides)
- Bits and Bytes

Interactive Classes

- Please feel free to raise your hand at any time with a question!
- Questions and comments are encouraged and recommended!

Asynchronous Questions

- Have non-assignment related questions outside of class?
  - Reach out on Ed
- NOTE: We will use Ed for class announcements so please check Ed regularly.
  Anonymous posts on ED are only Anonymous to other students

Visit Ed through the link on Canvas

Asynchronous Questions

- Have assignment related questions outside of class?
  - Reach out on IntelliCopilot

- NOTE: We will be using IntelliCopilot as our main form of communication for assignments

Visit through the link on Canvas

What is CS 107?

- The CS 106 series teaches you how to solve problems as a programmer

- Many times CS 106 instructors had to say “just don’t worry about that” or “it probably doesn’t make sense why that happens, but ignore it for now” or “just type this to fix it”

- **CS 107 *finally* takes you behind the scenes**

- **How do things really work in there?**

  > It’s not quite down to hardware or physics/ electromagnetism (those will have to stay even further behind the scenes for now!)

  > It’s how things work inside Python/C++ (we will explore from C), and how your programs map onto the components of computer systems

CS107 Learning

- The goals for CS107 are for students to gain **mastery** of
  - writing C programs with complex use of memory and pointers
  - an accurate model of the address space
  - strong understanding of the compile/runtime behavior of C programs

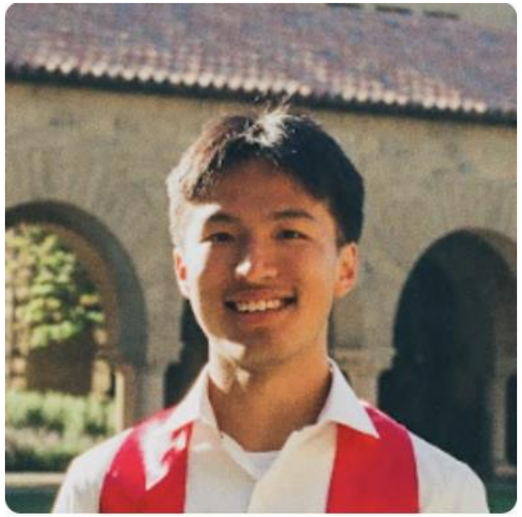
- to achieve **competence** in
  - translating C to/from assembly
  - writing programs that respect the limitations of computer arithmetic
  - identifying bottlenecks and improving runtime performance
  - writing code that correctly ports to other architectures
  - working effectively in UNIX development environment

- and have **exposure** to
  - a working understanding of the basics of computer architecture
  - understanding compilers and disassemblers
  - understand the semantics of assembly with respect to stack layout

Meet the Instructors

Adam Keppler
(Co-Instructor)
akeppler@stanford.edu

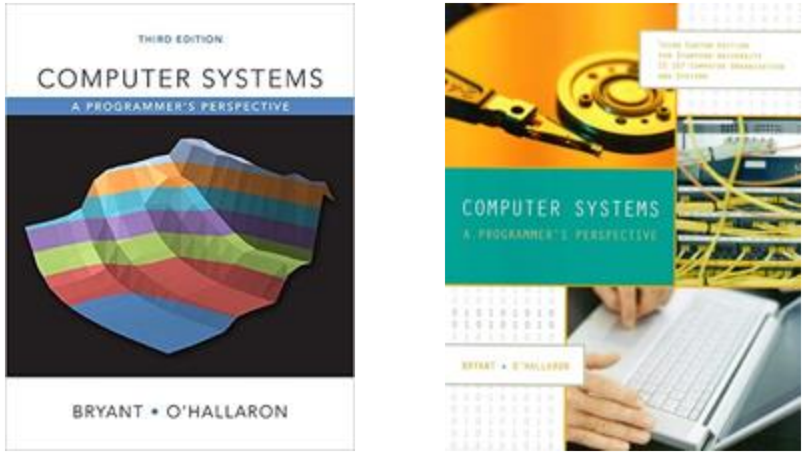
Olayinka Adekola (Ola)
(Co-Instructor)
oadekola@stanford.edu



Ben Yan
(Course Assistant)
bbyan@stanford.edu

Textbook(s)

- *Computer Systems: A Programmer's Perspective* by Bryant & O'Hallaron, **3rd Edition**
  - **3rd edition matters** – important updates to content
  - Stanford Library has generously scanned **all** readings for CS107 under "fair use" (private study, scholarship, research). [**Canvas -> Files**]. Please do not distribute.
  - If you want more context, you may want to purchase a full copy
- A C programming reference of your choice
  - *The C Programming Language* by Kernighan and Ritchie (free link on course website Resources page)
  - Other C programming books, websites, or reference sheets



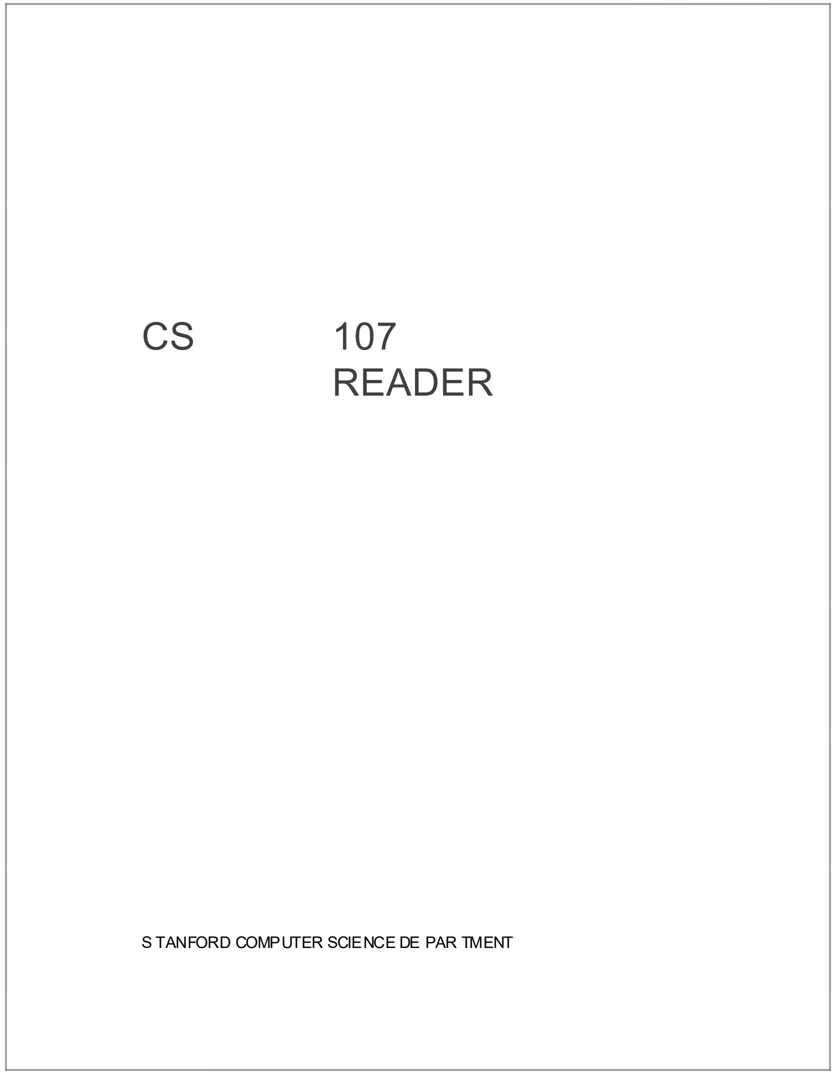
Full textbook

For CS107-specific readings

The textbooks are a **very** good resources in this course!

C Programming Language

Course Reader



CS 107 READER

STANFORD COMPUTER SCIENCE DEPARTMENT

There is a course reader, which condenses much of the material for the course:

https://stanford.edu/~cgregg/cgi-bin/107-reader

- If you find typos, let us know!

Course Structure

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

Great preview of homework!

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. **Heap Allocators -** *How do core memory-allocation operations like malloc and free work?*

Grading

****	40%	Assignments
*	10%	Lab Participation
**	15%	Midterm Exam (7/16)
*	10%	Heap Allocator(Final Project)
**	15%	Final Exam (8/15)
*	10%	Lecture Participation

Grading

****	40%	Assignments
*	10%	Lab Participation
**	15%	Midterm Exam (7/16)
*	10%	Heap Allocator(Final Project)
**	15%	Final Exam (8/15)
*	10%	Lecture Participation

Extra Credit

Late Days

- Due to the fast pace of this course, there are no late days
- Instead, you can submit late with an assignment cap
    - assignment turned in 1 day late will receive a 95% cap
    - assignment turned in 2 days late will receive a 90% cap
    - we will not work that is more than 2 days late (unless OAE)

Exceptions:

- assign0 no cap on late submission
- assign6 no late submissions accepted

OAE Accommodations

If you have OAE accommodations, please fill out this form:
https://forms.gle/dDx2EdhaVYDz2ihq6

This is also on the course website and Ed!

Non-OAE accommodations – reach out via Email

Stanford Honor Code

- The **Honor Code** is an undertaking of the students, individually and collectively:
  - that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
  - that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

- The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

- While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

see also: http://honorcode.stanford.edu/

It is your responsibility to ensure you have read and are familiar with the honor code guidelines posted on the main page of the CS107 course website. Please read them and come talk to us if you have any questions or concerns.

Honor Code and CS107

- Please help us ensure academic integrity:
  - Indicate any assistance received on HW (books, friends, internet, ChatGPT, 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 our ED forum.
  - Report any inappropriate activity you see performed by others.
- Assignments are checked regularly for similarity with help of software tools.
- 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.
  - If you realize that you have made a mistake, you may retract your submission to any assignment at any time, no questions asked.

https://cs107.stanford.edu/collaboration

Always Cite Your Sources

- The world has an increasing number of resources available from:
  - Documentation
  - Stack Overflow
  - ChatGPT
  - Medium
  - IntelliCopilot
  - And More

- Increasingly Important to Cite Sources!

- Please tag sources that you used either in the README or top of the relevant file.

Lecture Participation

In-Class Attendance is **required** for all non-CGOE students

- During the lectures, there will be interactive questions
- Some of the questions may encourage you to work with your neighbor
- Lecture Attendance begins tomorrow but let’s do a trial run

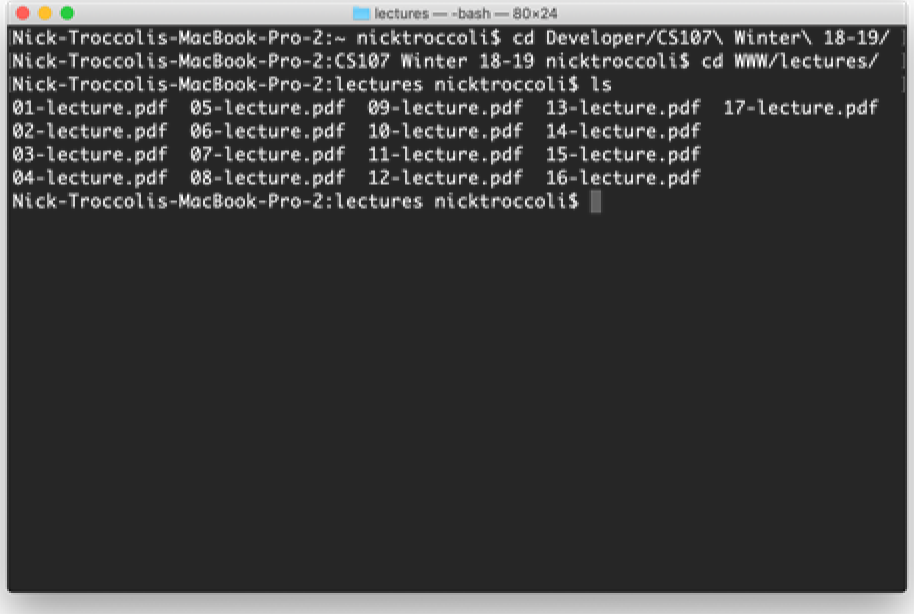
Remote Participation (Non-CGOE)

- If you need to participate in CS 107 remotely, please fill out this form: https://forms.gle/QmhtKd57moe3GQuz7

- Even if you have already reached out to the course staff, you MUST submit this form

What is Unix?

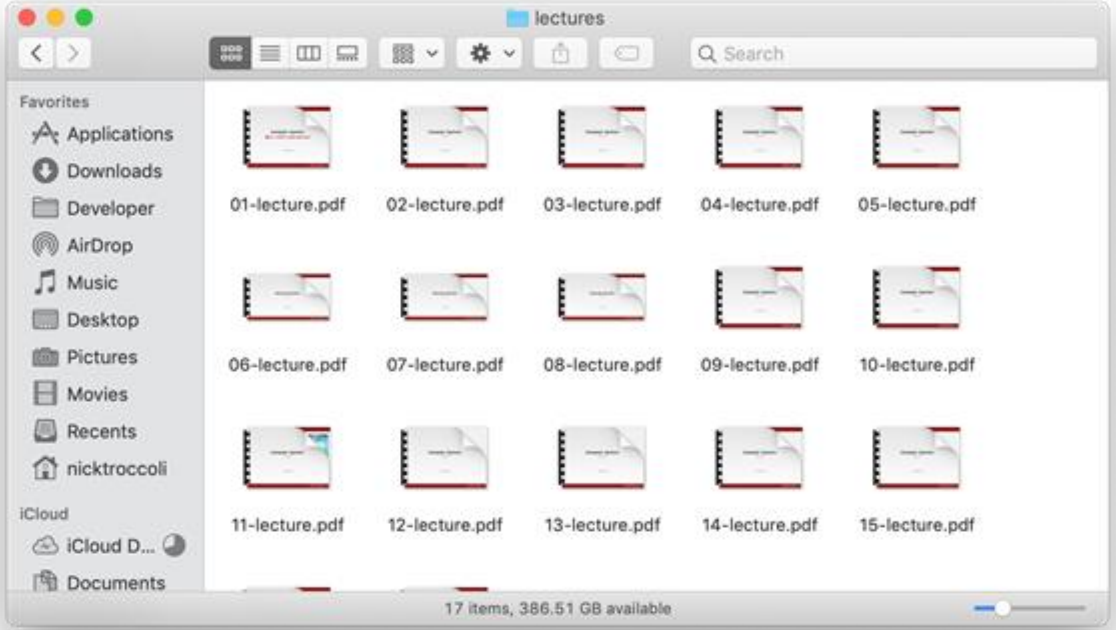
- **Unix:** a set of standards and tools commonly used in software development.
  - **macOS** and **Linux** are operating systems built on top of Unix
- You can navigate a Unix system using the **command line** ("terminal")
- Every Unix system works with the same tools and commands



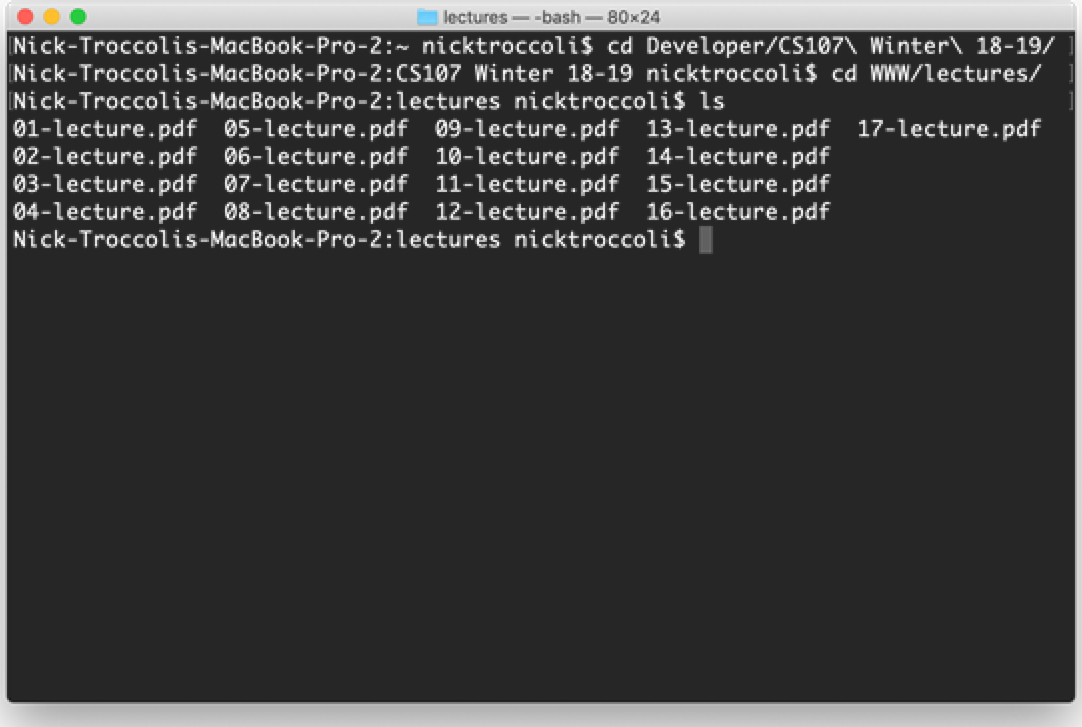
```
Nick-Troccolis-MacBook-Pro-2:~ nicktroccoli$ cd Developer/CS107\ Winter\ 18-19/
Nick-Troccolis-MacBook-Pro-2:CS107 Winter 18-19 nicktroccoli$ cd WWW/lectures/
Nick-Troccolis-MacBook-Pro-2:lectures nicktroccoli$ ls
01-lecture.pdf  05-lecture.pdf  09-lecture.pdf  13-lecture.pdf  17-lecture.pdf
02-lecture.pdf  06-lecture.pdf  10-lecture.pdf  14-lecture.pdf
03-lecture.pdf  07-lecture.pdf  11-lecture.pdf  15-lecture.pdf
04-lecture.pdf  08-lecture.pdf  12-lecture.pdf  16-lecture.pdf
Nick-Troccolis-MacBook-Pro-2:lectures nicktroccoli$
```

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



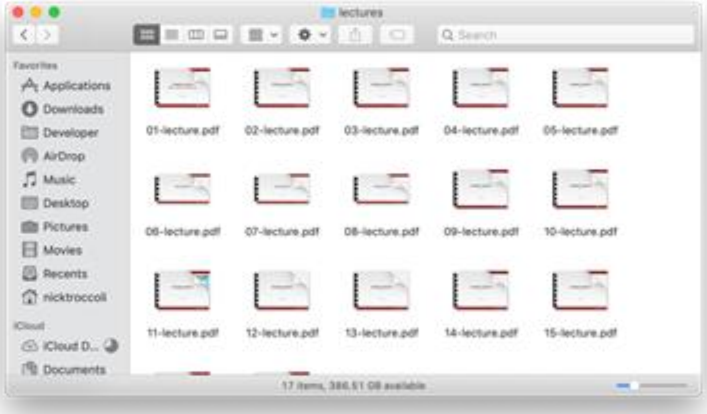
```
Nick-Troccolis-MacBook-Pro-2:~ nicktroccoli$ cd Developer/CS107\ Winter\ 18-19/
Nick-Troccolis-MacBook-Pro-2:CS107 Winter 18-19 nicktroccoli$ cd WWW/lectures/
Nick-Troccolis-MacBook-Pro-2:lectures nicktroccoli$ ls
01-lecture.pdf  05-lecture.pdf  09-lecture.pdf  13-lecture.pdf  17-lecture.pdf
02-lecture.pdf  06-lecture.pdf  10-lecture.pdf  14-lecture.pdf
03-lecture.pdf  07-lecture.pdf  11-lecture.pdf  15-lecture.pdf
04-lecture.pdf  08-lecture.pdf  12-lecture.pdf  16-lecture.pdf
Nick-Troccolis-MacBook-Pro-2:lectures nicktroccoli$ 
```

Text-based
interface

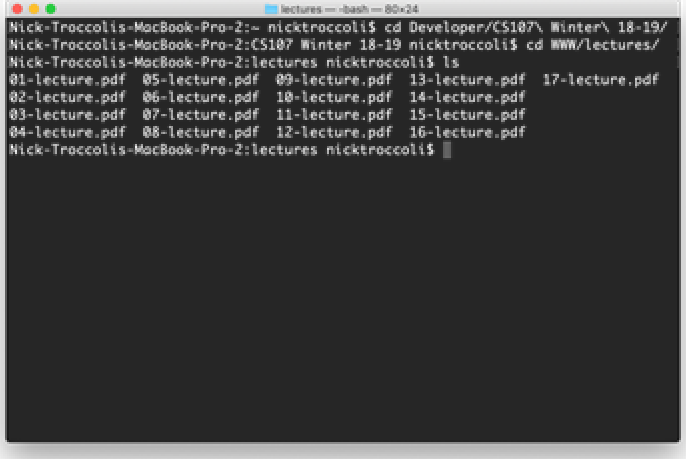
Command Line Vs. GUI

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



```
Nick-Troccolis-MacBook-Pro-2:~ nicktroccoli$ cd Developer/CS107\ Winter\ 18-19/
Nick-Troccolis-MacBook-Pro-2:CS107 Winter 18-19 nicktroccoli$ cd WWW/lectures/
Nick-Troccolis-MacBook-Pro-2:lectures nicktroccoli$ ls
01-lecture.pdf  05-lecture.pdf  09-lecture.pdf  13-lecture.pdf  17-lecture.pdf
02-lecture.pdf  06-lecture.pdf  10-lecture.pdf  14-lecture.pdf
03-lecture.pdf  07-lecture.pdf  11-lecture.pdf  15-lecture.pdf
04-lecture.pdf  08-lecture.pdf  12-lecture.pdf  16-lecture.pdf
Nick-Troccolis-MacBook-Pro-2:lectures nicktroccoli$
```

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.

Unix Commands To Try

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

See the course website for more commands and a complete reference.

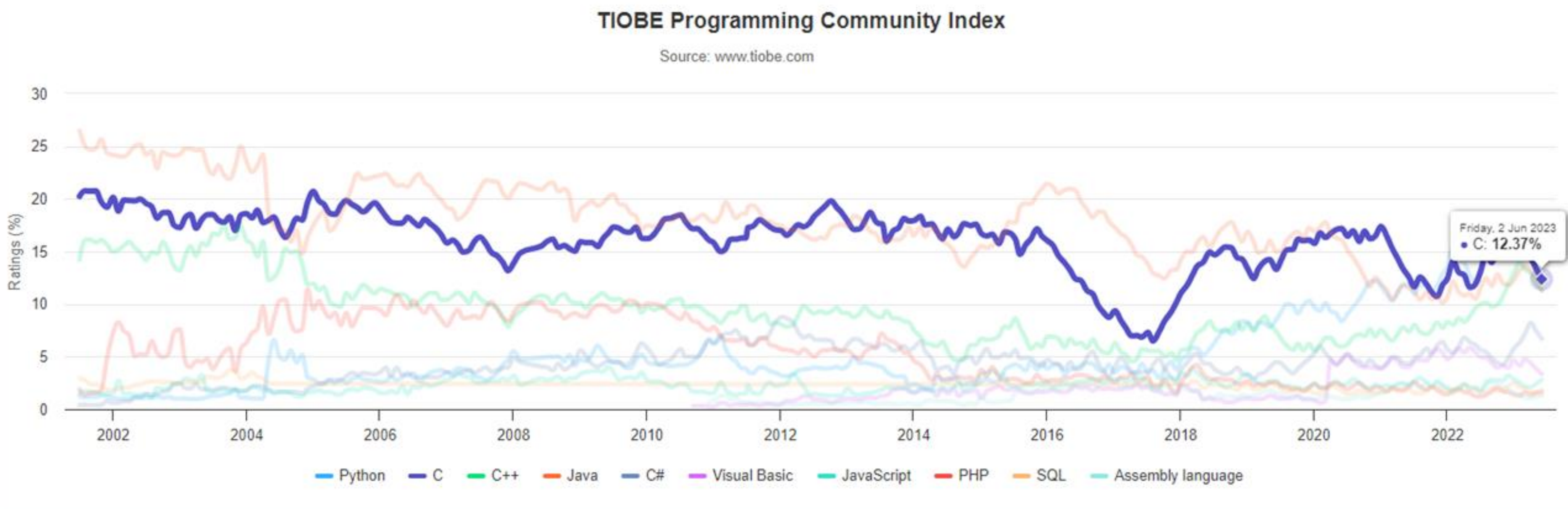
Learning Unix and the Command Line

- Using Unix and the command line can be intimidating at first:
  - It looks retro!
  - How do I know what to type?
- It’s like learning a new language:
  - At first, you may have to constantly look things up (**resources** on course website!)
  - It’s important to spend as much time as possible (during labs and assignments) building muscle memory with the tools

Plan For Today

- Introduction + Syllabus
- Unix?
- C 101 (please review the skipped slides)
- Bits and Bytes

Programming Language Popularity



- C has consistently been the most or 2nd most popular language since 1988!

The C Language: History

- Birthdate around 1970
- Created to make writing Unix (the OS itself) and tools for Unix easier
- Common Ancestor to most Programming Languages
  - Especially C++/Java family of languages
- Design principles:
  - Small, simple abstractions of hardware
  - Minimalist aesthetic
  - C Focuses on:
    - Efficiency and minimalism
  - C Sacrifices:
    - Safety (unlike Java/Python)
    - Convenient high-level services and abstractions (which are commonly found Java, Python, C++)
- As the common ancestor, it inspired safer systems, increased abstraction, and higher level features

C vs. C++ and Java

They all share:

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

C Limitations:

- No advanced features like operator overloading, default arguments, pass by reference, classes and objects, Abstract Data Types, etc.
- Standard Libraries offer core functionality rather than nice wrappers or syntactic sugar.
- Small language means small footprint on a system.
- No runtime checks (this may cause severe security vulnerabilities and bugs !)

Programming Philosophies

Functional Programming (FP) – Programming Philosophy that seeks to avoid using state, preferring functions with no side effect, data immutability, and thread-safe code.

C supports some functional elements, but is not a functional programming language (FPL).

Procedural Programming (PP) – About creating procedures or ‘scripts’ using functions to setup a series of tasks or steps to complete.

C is consider the quintessential Procedural Programming Language.

Object Oriented Programing (OOP) – The idea that we can create objects that contain and maintain both data and code in the form of fields (attributes/methods). There are also interactions between the objects such as inheritance, encapsulation, abstraction, and polymorphism.

Unlike Python/C++/Java, C is ***not*** an Object Oriented Language and does not have a notion of objects.

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.**

Programming Philosophy is a Spectrum: Most major languages have elements of multiple philosophies, while some occasionally epitomize a specific philosophy.

Quintessential Examples:

LISP - Functional Programming
Java - Object Oriented Programming
C – Procedural Programming

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.
- Modern alternatives to C are emerging (e.g., Rust), but they’re more complicated and not quite ready for those new to systems programming (but find me outside of class and I’m happy to talk about them :)

The Heart of C

- C helps those who help themselves
- C is meant to give fundamental tools that expose the computer’s internal workings as much as possible, without becoming Assembly
- As such C is fundamentally about data, its storage, and its manipulation
  - Every program is technically data itself, with C it is possible to write self-editing programs
- There are no objects in C only data
- Types in C is an illusion meant to provide convenience to the user and help with organization

Types in C: Static & Strong Typing

- A type in C defines how much memory is stored with the associated data

- It also defines whether the value is raw data or a ‘pointer’ to another location in memory
  - We will talk extensively more about pointers in the coming weeks.

- For now, lets take a look at the fundamental non-pointer types:
  - int – records an integer value (…, -3, -2, -1, 0, 1, 2, 3 , …) in binary
  - float – records a decimal value (-0.3, 0, 0.3 , 1.0, 3.14, etc.) in binary
    - If you are wondering how this happens, it is a great question! We will cover it in a later section
  - char – Any English letter (uppercase, lowercase, numbers, and some control codes)
  - bool – Technically as small as a single bit 0 or 1, compilers sometimes store it as a char due to restrictions on byte-alignment of memory

- While the above are the core types, they come in different flavors. Flavors are arranged by:
  - **How much room,** you have to store the information (it takes more room to store 1024 than 4)
  - **Storage Format**, is the value always positive? Do you need a negative? A Decimal?
  - **Does it hold data or point to data?** (Pointers)

Your First C Program

Assign 0: out today and due Friday 6/27
Note: Short turn-around to make the rest of the quarter easier
Need more time?

Need help on C syntax or writing C code?

- Slides on website
- Lecture code on myth machines
- Office Hours
- IntelliCopilot

Your First C Program

```c
/*
 * 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

```c
/*
 * 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

```c
/*
 * 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

Console Output: printf

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

`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)`

```
// Example
char *classPrefix = "CS";
int    classNumber = 107;
printf("You are in %s%d", classPrefix, classNumber);    // You are in CS107
```

Familiar Syntax

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

for (int i = 0;    i < i++) {      // for loops
    10; 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;
```

Writing, Debugging and Compiling

We will use:

- the **vim** text editor to write our C programs
- the **make** tool to compile our C programs

Now

- the **gdb** debugger to debug our programs
- the **valgrind** tools to debug memory errors and measure program efficiency

Next week

Working On C Programs

- **ssh** – remotely log in to Myth computers
- **Vim** – text editor to write and edit C programs
  - Use the mouse to position cursor, scroll, and highlight text
  - Ctl-x Ctl-s to save, 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 code is accessible at **/afs/ir/class/cs107/lecture-code/lect[N]**
  - Make your own copy: **cp -r /afs/ir/class/cs107/lecture-code/lect[N] lect[N]**
  - See the website for even more commands, and a complete reference.

Assignment 0:

Assignment page:

https://web.stanford.edu/class/cs107/assign0/

Assignment already released, due Friday, 6/27

Lab Sign Up

Preference form is now open

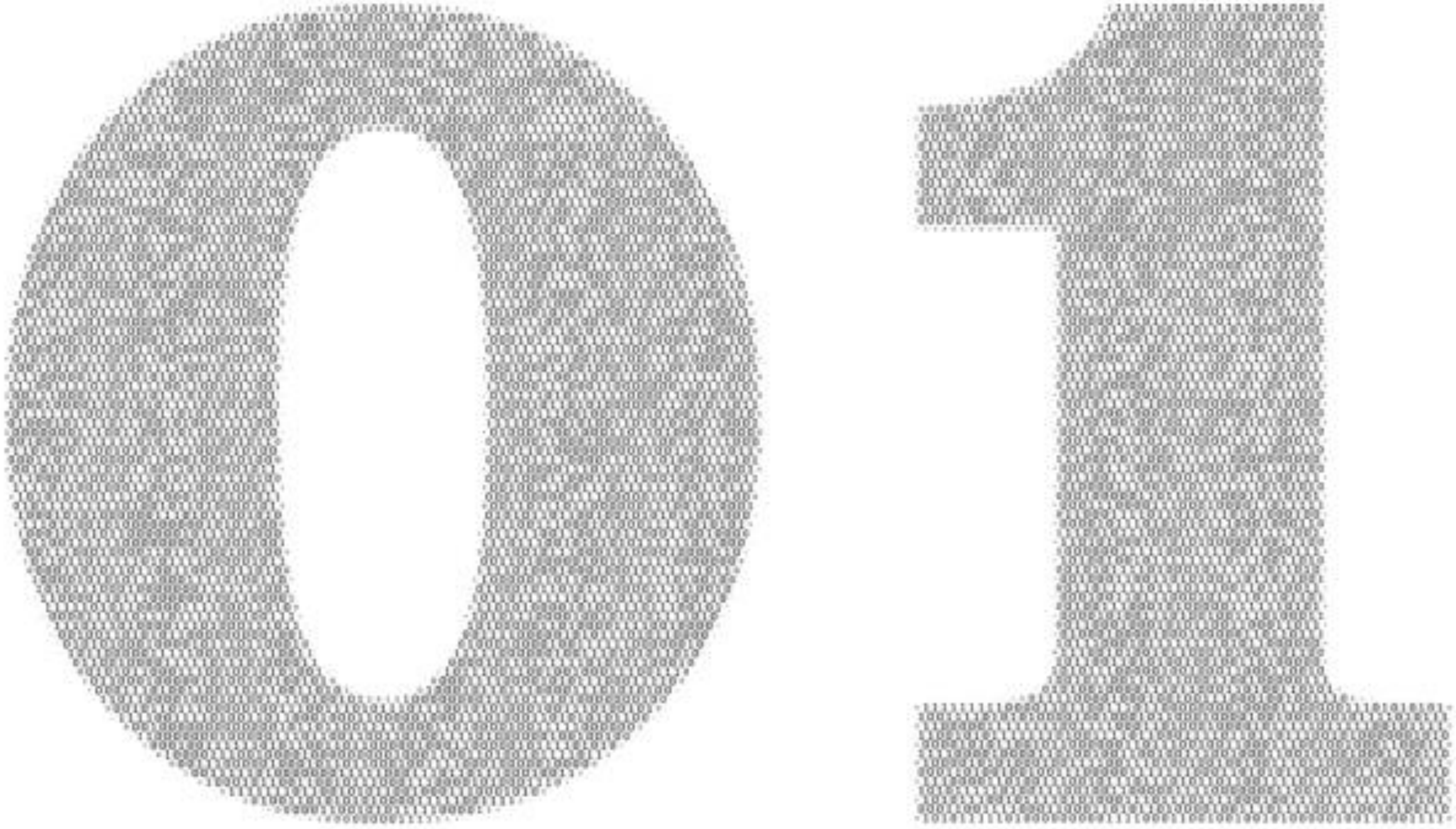
Labs will begin week this week!

Lab signup is based on submitted preferences, otherwise you’ll be assigned. Please submit by this tomorrow.

Plan For Today

- Introduction + Syllabus
- Unix?
- C 101 (please review the skipped slides)
- Bits and Bytes

Bits



Computers are good at detecting "off" or "on"

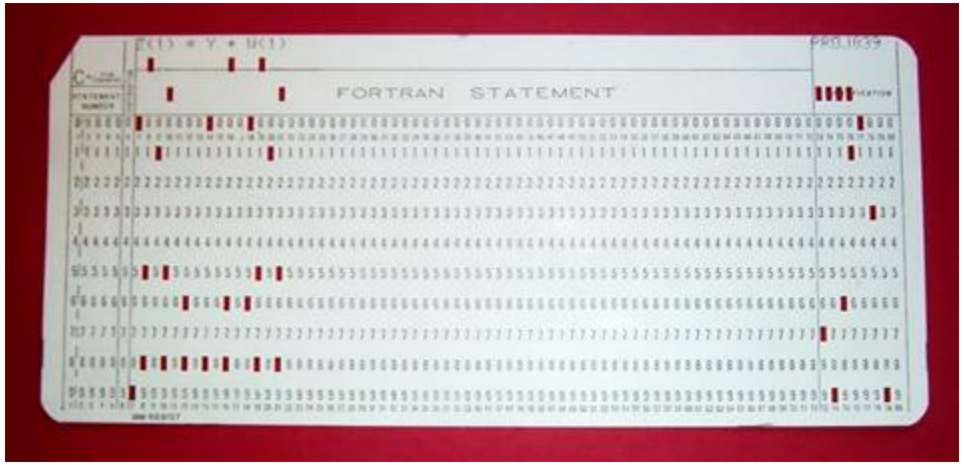
We have lots of ways to tell the difference between two different states:

Clockwise / Counterclockwise

True or False

Lightbulb off / on

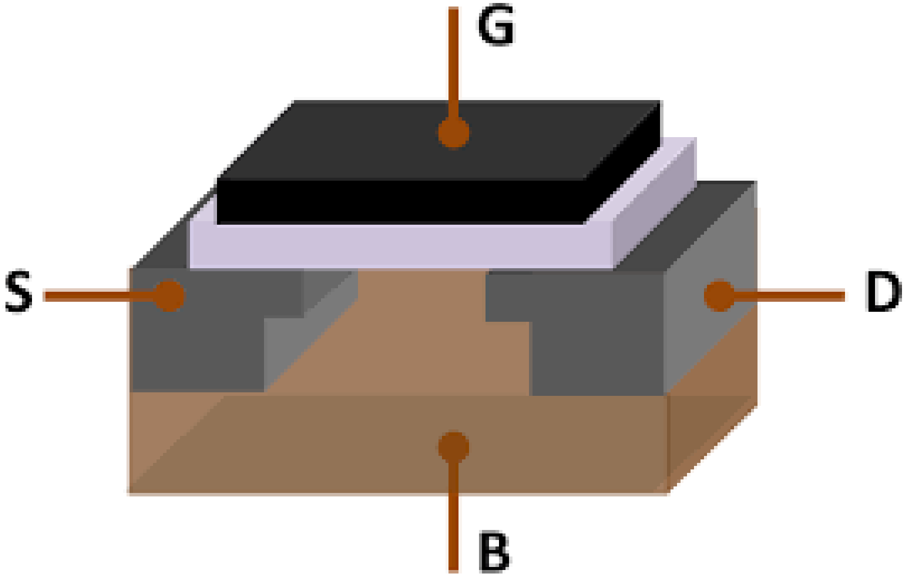
Yes or No



Punchcard hole / no hole

Computers are good at detecting "off" or "on"

Electronic computers are built using transistors. A transistor can be set up to either be "off" or "on" -- this gives us our 0 and 1!



Computers are built around the idea of two states: "on" and "off". Transistors implement this in hardware, and bits represent this in software.

One Bit At A Time

- We can combine bits, like with base-10 numbers, to represent more data.

8 bits = 1 byte.

- Computer memory is just a large array of bytes! It is *byte-addressable*; you can't address (store location of) a bit; only a byte.

- Computers still fundamentally operate on bits; we have just gotten more creative about how to represent different data as bits!
  - Images
  - Audio
  - Video
  - Text
  - And more…

How does a bit do so much?

- Information can be reshaped

- Numbers can have the same value but in different representations

- Typically, we use base 10 in everyday life (most people attribute this to humans having 10 fingers, but humans have used other # systems)

- Base 10 has ten digits: 0 1 2 3 4 5 6 7 8 9

- Base 2 has two digits: 0 1

- We can represent up to ten numbers with one digit in base 10

- We can represent up to two numbers with one digit in base 2

- If we want to represent more numbers, we add more digits regardless of the base.

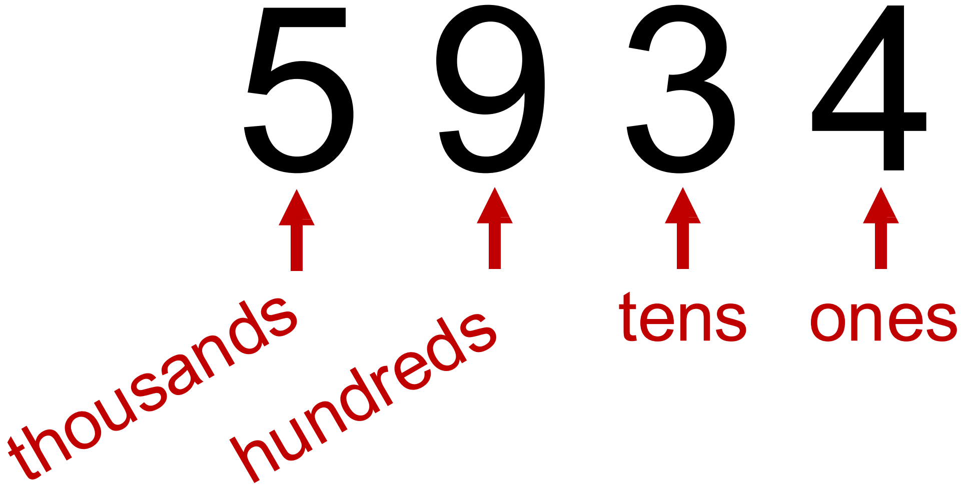
Base 10

5 9 3 4

Digits 0-9 (*base-10*)

4 Columns

Base 10



5 9 3 4

thousands

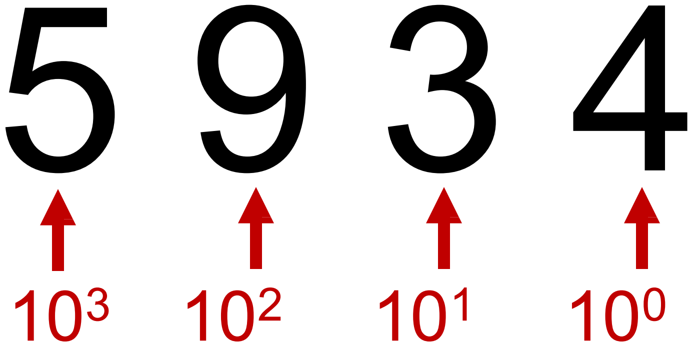
hundreds

tens

ones

= **5***1000 + **9***100 + **3***10 + **4***1

Base 10



5 9 3 4

10^3 10^2 10^1 10^0

$= \mathbf{5}*10^3 + \mathbf{9}*10^{2} + \mathbf{3}*10^{1} + \mathbf{4}*10^{0}$

Base 10

5 9 3 4

10^X: 3 2 1 0

Base 2

1 0 1 1

2^x: 3 2 1 0

Digits 0-1 (*base-2*)

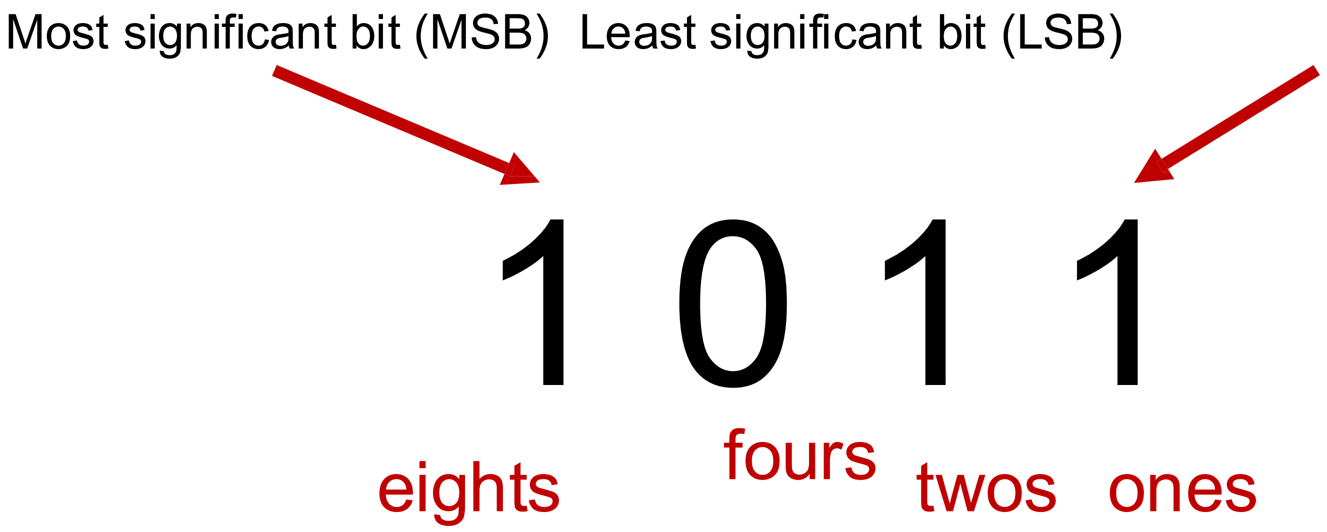
Base 2

1 0 1 1

2^3 2^2 2^1 2^0

$= \mathbf{1}*2^3 + \mathbf{0}*2^2 + \mathbf{1}*2^1 + \mathbf{1}*2^0 = 11_{10}$

Base 2



Most significant bit (MSB) Least significant bit (LSB)

1 0 1 1

eights fours twos ones

$$= \mathbf{1}*8 + \mathbf{0}*4 + \mathbf{1}*2 + \mathbf{1}*1 = 11_{10}$$

Base 10 to Base 2

Question: What is 6 in base 2?

- 2 Strategies:
    1. Build the number from the left (Find the most significant bit first)

    2. Build the number from the left (Find the least significant bit first)

Practice: Base 2 to Base 10

What is the base-2 value 1010 in base-10?

a) **20**

b) **101**

c) **10**

d) **5**

e) **Other**

Practice: Base 10 to Base 2

What is the base-10 value 14 in base 2?

a) **1111**

b) **1110**

c) **1010**

d) **Other**

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store?

- Please answer minimum first

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store? **minimum = 0** **maximum = ?**

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store? **minimum = 0** **maximum = ?**

1 1 1 1 1 1 1 1

2^x:

7 6 5 4 3 2 1 0

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store? **minimum = 0** **maximum = ?**

	1	1	1	1	1	1	1	1
2^x:	7	6	5	4	3	2	1	0

- **Strategy 1:** $1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 255$

Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store? **minimum = 0** **maximum = 255**

	1	1	1	1	1	1	1	1
2^x:	7	6	5	4	3	2	1	0

- **Strategy 1:** $1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 255$
- **Strategy 2:** $2^8 – 1 = 255$

Byte Values

- How about minimum and maximum base-10 value for 16 bits?

  minimum = 0   maximum = ?

Multiplying by Base

$$1450 \text{ x } 10 = 1450\underline{\mathbf{0}}$$

$$1100_2 \text{ x } 2 = 1100\underline{\mathbf{0}}$$

Key Idea: inserting 0 at the end multiplies by the base!

NOTE: Inverse is also true, multiplying by the base adds a 0

Dividing by Base

$$1450 / 10 = 145$$

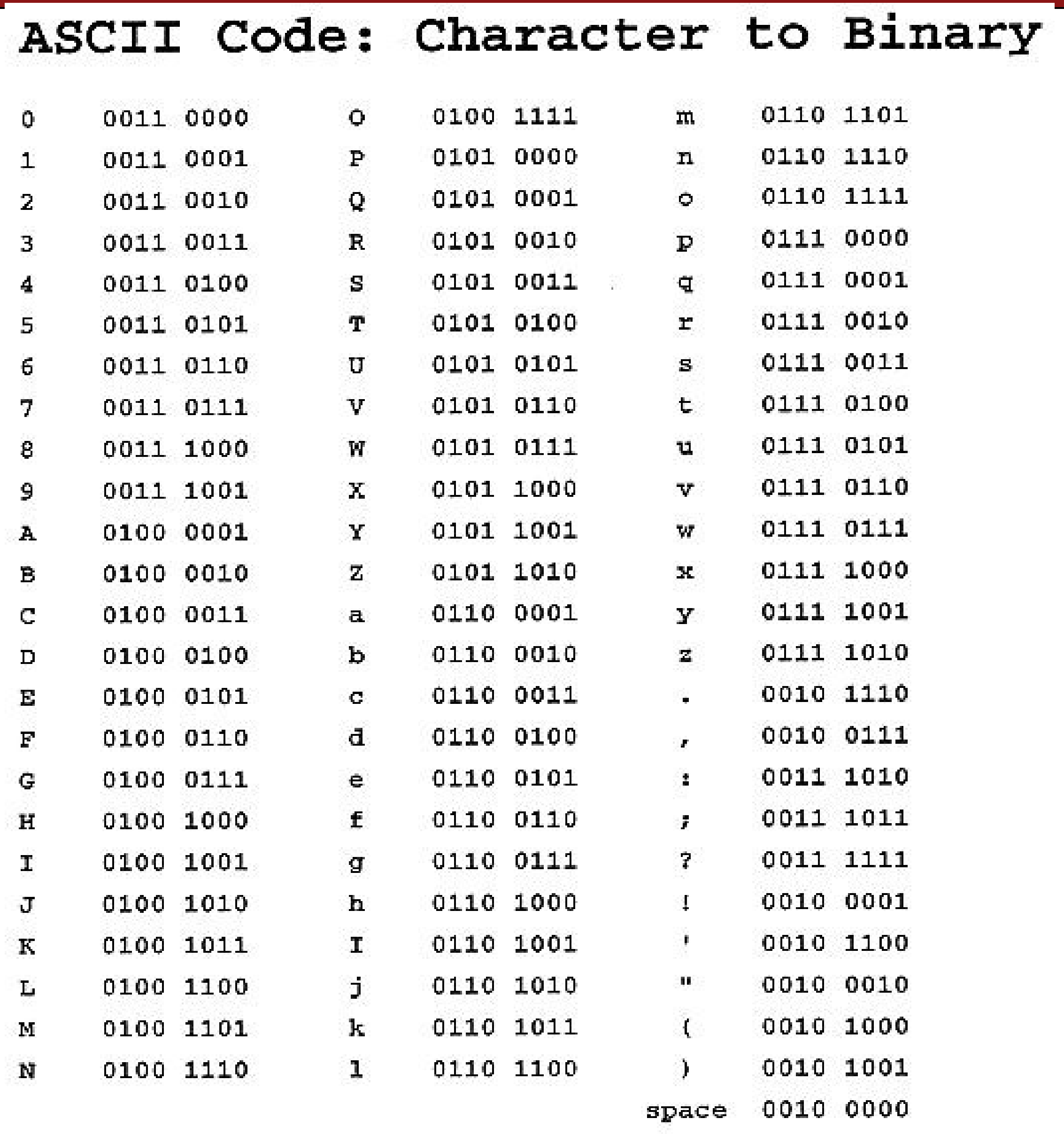
$$1100_2 / 2 = 110$$

Key Idea: removing 0 at the end divides by the base!

NOTE: Inverse is also true, dividing by the base removes a column

Combinations of bits can encode anything and represent everything

We can encode anything we want with bits. E.g., the ASCII character set.



ASCII Code: Character to Binary

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	I	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	(	0010 1000
N	0100 1110	l	0110 1100	)	0010 1001
				space	0010 0000

Questions?