What this course is about:

- Pulling back the curtain
- Begin to take nothing for granted, BE CURIOUS!
- How does it all work? It's a lot to handle, there's so much to consider, but you will be better for it
- We focus on how computers are built, at many layers, but...
- Do not apply this curiosity just to computers!
- You'll be learning some extremely valuable skills. This is where you'll set yourself apart from the webdev who learned Python in his spare time. Cultivate deep understanding of what's going on.
- Why study how computers are built? It's important for understanding *what they are.* It is also increasingly relevant as hardware is rapidly changing, as attacks on hardware models, are becoming more commonplace, and as software techniques are becoming increasingly important for performance.
- At the end, if you're stranded on an island, YOU will be the one to get technology up and running (with enough time and sweat)

Intro:

- see course webpage for updates, schedule
- I will post labs well ahead of time as we go in order to give you more than enough time to work on them
- We'll also be using Piazza (use it!)
- When I post labs, solutions, etc., I'll announce it on Piazza so you get a notification
- Book
- Office hours
- We're still figuring out course machine, but it will probably be fourier
- There will be (3) labs on Fridays 3:15-4:05. Get to know your TAs. Their information is on the webpage
- Ask questions!

A brief history of computers (how they were built)

- First was mechanical analog computers (all special-purpose)
- Antykithera mechanism is the first known example (205BC)
- used for calculating astronomical tables (for calendars, astrology, Olympiad cycles)
- used rotating gears with fine-tuned gear ratios (go find a recreation!)
- Su Song's Clock tower in China (1088)
- Babbage's Analytical -> Difference Engines (1840s)
- Later differential analyzers (late 1800s -> 1930s), solving differential equations
- Then moved to electromechanical devices (relays) 1930s
- Digital Devices -> Konrad Zuse Z2->3
- Moved to Vacuum Tubes (ENIAC) for US Army
- First semiconductor in 1947
- First Semiconductor Computer 1950 (NIST SEAC)
- First Transistor Computer 1953 (Univ. of Manchester)
- First IC 1958, Jack Kilby

    - IBM System/360 uses ICs and discrete components (1964)
    - Silicon Transistors in the CDC 6600 (first supercomputer, (1965)
    - First Microprocessor, Intel 4004 (1971)
    - Intel 8008 in 1972, used for Datapoint 2200, first microcomputer
    - Altair 8800 in 1974, used Intel 8080
    - MOS 6502 in 1975, still used today
    - Apple I in 1976, designed by Steve Wozniak
    - Intel 8086 in 1978, great great great grandfather of today's chips
    - First IBM PC, 1981, based on Intel 8088, ran MS-DOS
    - Intel Pentium, 1993
    - Pentium 4, 2000 (End of clock speed scaling)
    - IBM POWER 4, first commercial multicore (2001)

- What **is** a computer, anyway?
    - What do they do? (solve equations, transform information, manipulate symbols)
    - Originally, computer meant a person!
    - Take input, produce output
    - They do what you tell them! They are idiots
    - Is the brain a computer? (von Neumann)
    - Computers today move around electrons and photons. We interpret these manipulations of matter & energy as computations.

- **How we're going to understand computers (Abstraction)**
    - We'll start out with devices (transistors)
    - Then we'll treat those as black boxes to build gates
    - Then we'll use gates to build circuits
    - Then we'll use the circuits to build microarchitectural components
    - Then we'll see how machine language drives those components
    - And how the instruction set translates into machine language
    - Finally, we'll come full circle with how C programs turn into hardware operations
    - At each level, we abstract away details
    - Abstraction is incredibly important
    - We use it for Math (e.g., numbers!)
    - We use it in our brains to reason about our world (what is "me?")
    - We use it in Politics (What is the notion of "President?")
    - To make progress, you have to be able to cut through abstraction, you can't ignore that it exists
    - This class is going to teach you not only how to build up abstraction layers (from hardware to software) but also how to cut across them to reach understanding

- **What is the ultimate computer?**
    - Some computers clearly more powerful than others. Supercomputer of today can easily outperform those of the past. Apple watch is almost 1000x performance of lunar lander used in Apollo
    - If we get rid of speed/memory (abstract it away), is one computer more powerful than another?
    - The surprising answer is "NO." They all have the same power. If I wanted to emulate an

intel multicore chip on ENIAC, I could! It would be very, very slow however.

- **Turing Machines**
    - In 1928, David Hilbert asks if there is an algorithm to compute universal validity of a mathematical statement (stated in logic).
    - Turing (mathematician & cryptologist) develops his machine (Turing Machine). Answers “No.,” but showed some other incredible results!
    - It is an extremely simplified model of computation (NOT A COMPUTER) (understood differently those days), more precisely, an algorithm
    - Has infinite tape and read/write head
    - A control (if we are in state 72, and we see a $, move the head right one symbol, and go to state 53
    - Unsurprisingly, we can write code to simulate these things
    - Surprisingly, they can simulate modern computers!
    - Could even simulate your macbook with one
    - How? Well, the first TMs did something special purpose (e.g. compute digits of Pi, enumerate the natural numbers)
    - What if we build a TM that takes as input another TM’s control and input?
    - It can “simulate” that TM
    - This is a “Universal Turing Machine”

Turing’s Thesis:

1) Take 2 computers, x & y
2) For each, write a program that simulates a universal TM
3) Write a UTM program that simulates x, and one that simulates y
4) For (3) to work, you must assume that any computation that can be done, can be done by a TM

TURING’S THESIS: Any computation that can be done can be done by a TM
- Therefore, TMs are UNIVERSAL COMPUTATION DEVICES
- This thesis is not provable, but it is assumed to be true.
- If you make this assumption, then a general solution to entscheidung’s problem not possible

Takeaway: computers can do every known mechanical computation. Not necessarily in a “convenient” amount of time. There are sets of problems that are all equivalent in the number of steps it takes to solve (you’ll see these in CS 430). In 530, you’ll study problems that are known to not be efficiently solvable by a computer (or solvable at all)

Wacky Questions:
- How are brains different from computers?
- Are *we* computers? think about DNA
- How are we *different* though?
- Some very smart people think the *Universe* might be a computer (or a computer simulation). Will the same computational limits apply? What would it even mean for the universe to *compute*