02-201, Fall 2015, Carl Kingsford

Lecture 1: What is a computer?

0. Today's Topics

- Basic computer architecture
- How the computer represents data

1. What is a computer?

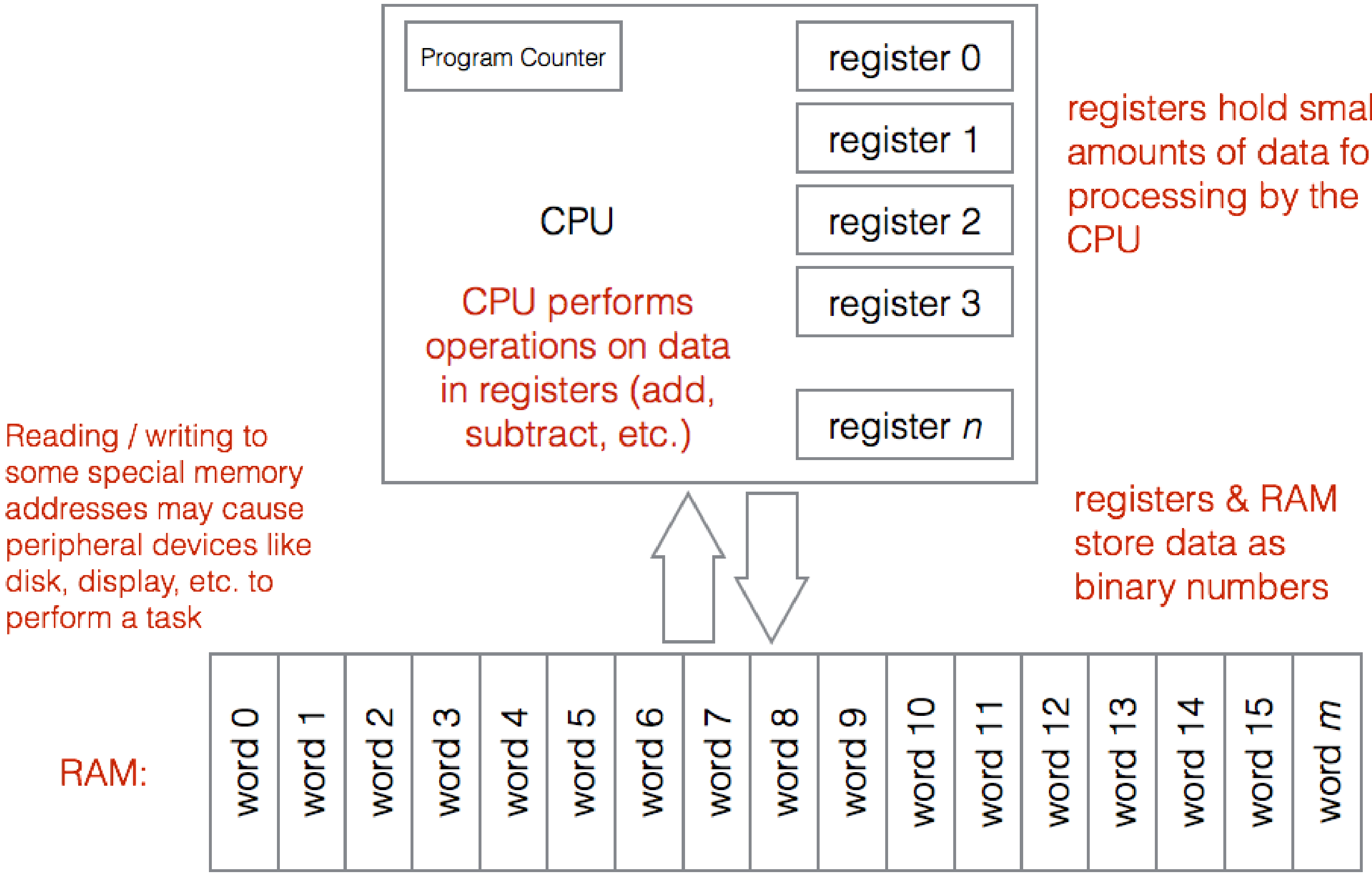
A modern computer is a collection of many components: the display, a keyboard, a disk drive, speakers, camera, microphone, network connection, etc.

The two most important components are:

- the **Central Processing Unit** (CPU): a compact electronic chip that executes simple instructions. This is the part that "computes".
- the **Random Access Memory** (RAM): this is the part that remembers the results of the computation.

It is these two components we will be thinking about most directly in this class.

1.a. A mental image of the computer



- The CPU has a number of *registers* that can each hold a number. The number of registers is small (modern "64-bit" Intel processors have 16 general purpose registers).
- RAM is broken down into *words*, each of which can hold a number. The number of words depends on how much memory your computer has. 16 gigabytes = about 2 billion words.

1.b. Some operations that the CPU can perform

The operations that the CPU can perform are generally very simple.

Examples:

- *Set* the value of a register to a given number x:

$$Ri \leftarrow x$$

- *Add, subtract, or multiply* two numbers in registers and put the result in a register:

$$Ri \leftarrow Rj + Rk$$

- Copy a number from a register into a memory location:

$$\text{RAM at location } i \leftarrow Rj$$

- Copy a number from a memory location into a register:

$$Ri \leftarrow \text{RAM at location } j$$

There are many other operations that a CPU can do, but most are these kind of simple operations on integers. We will see other operations soon.

1.c. Example: Computing the value of a polynomial

Suppose we want to evaluate this polynomial:

$$yx^2 + 4xy + 3x - 3$$

for $x = 10, y = 2$. We would have to ask the CPU to perform the following instructions:

1. $R1 \leftarrow 10$ Set the value of register 1 (R1) to 10 (x).
2. $R2 \leftarrow 2$ Set the value of register 2 (R2) to 2 (y).
3. $R3 \leftarrow R1 \times R1$ Multiply R1 by R1 and store the result in R3.
4. $R3 \leftarrow R2 \times R3$ Multiply R2 by R3 and store the result in R3.
5. $R4 \leftarrow R1 \times R2$ Multiply R1 by R2 and store the result in R4.
6. $R5 \leftarrow 4$ Set R5 to the value "4".
7. $R4 \leftarrow R4 \times R5$ Multiply R4 by R5 and store the result in R4.
8. $R3 \leftarrow R3 + R4$ Add R3 and R4 and store the result in R3.
9. $R5 \leftarrow 3$ Set R5 to the value "3".
10. $R1 \leftarrow R1 \times R5$ Multiply R1 by R5 and store the result in R1
11. $R1 \leftarrow R1 + R3$ Add R1 and R3 and store the result in R1
12. $R1 \leftarrow R1 - R5$ Subtract R5 from R1 and store the result in R1
13. $\text{RAM at location } 100 \leftarrow R1$ Store R1 into memory location 100

The answer is now in RAM at location 100.

We had to be very explict about each step and where the result of each step would be stored.

Note that we reused registers when we wanted to. When we reuse them, their contents are replaced by something new. The choice of registers we used (R1-R5) was arbitrary, and there are lots of other ways we could have writen the steps (e.g. by doing the operations in a different order).

> **Test yourself!** What are the contents of registers R1 through R5 before line 6 above?
>
> **Test yourself!** Write down a program in the style above to compute the value of the polynomial $2x^3y + 3x^2 - 7xy$.

1.d. How could we write the same thing in Go?

A programming language lets us avoid the tedium of writing down each instruction the computer must execute. We still have to be very explict about what we want the machine to do, but we can do it at a higher level:

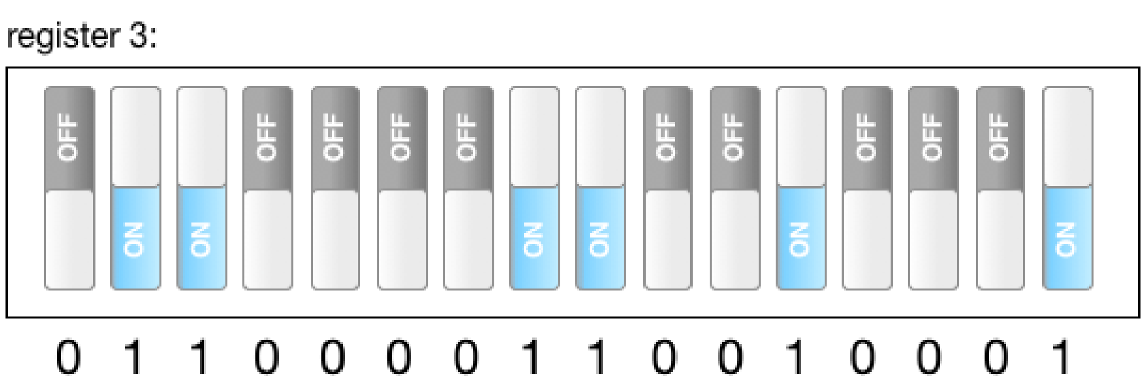
```go
package main
import "fmt"

func main() {
    var x = 10
    var y = 2
    var answer = y*x*x + 4*x*y + 3*x - 3
    fmt.Println(answer)
}
```

> **Exercise:** Run this program at http://play.golang.org. Change the values of `x` and `y` to see how the answer changes.
>
> **Looking ahead:** Guess how you might rewrite the above Go program to avoid using the variable `answer`, and only have variables `x` and `y`. Test to see if your solution works at http://play.golang.org.

2. How data is represented in the computer

The registers and RAM are make up of many tiny switches, each of which can either be on or off:



We use **0** to represent "off" and **1** to represent "on". Each one of the switches is a **bit**.

A computer is called a "64-bit" computer if its registers are 64-bits long (loosely speaking).

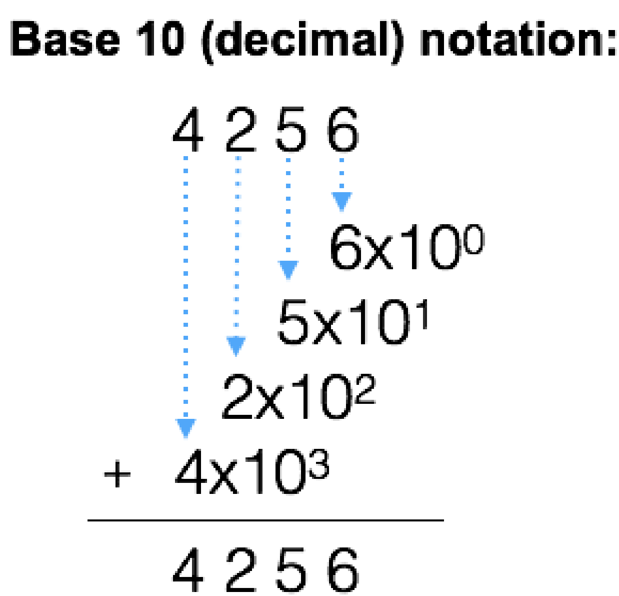
Eight switches in a row is a **byte**.

unit	size
bit	1 switch
byte	8 bits
kilobyte	2^{10} = 1024 bytes
megabyte	2^{20} = 1,048,576 bytes (approximately 1 million bytes)
gigabyte	2^{30} = 1,073,741,824 bytes (approximately 1 billion bytes)
terabyte	2^{40} = 1,099,511,627,776 bytes (approximately 1 trillion bytes)
petabyte	2^{50} = 1000 terabytes

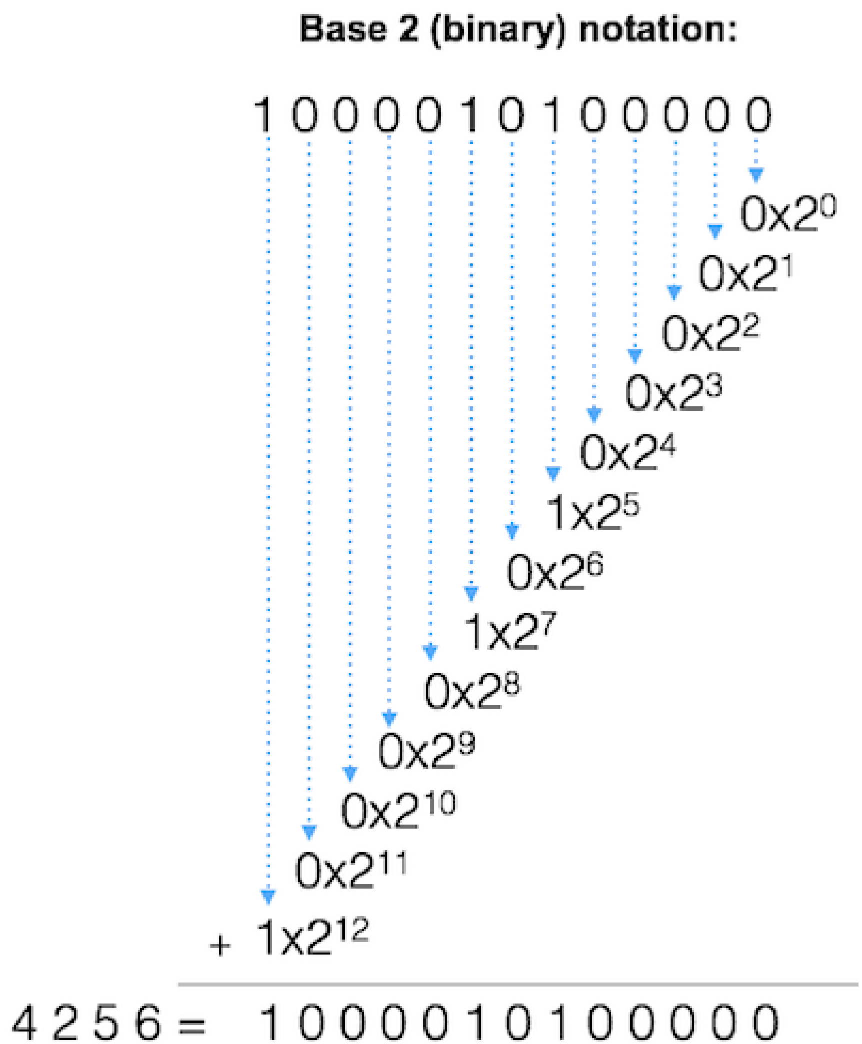
(Note that in some situations kilo-, mega-, giga-, tera- and petabytes are defined to be 1000, 1000000, 1 billion, 1 trillion, 1000 trillion exactly.)

2.a. Binary: how a computer represents numbers

With only the digits 0 and 1, the computer can't represent numbers in base 10. Instead, it must use base 2 (aka binary). You're all familar with how base-10 notation works:



Binary works the same way, with 10 replaced by 2:

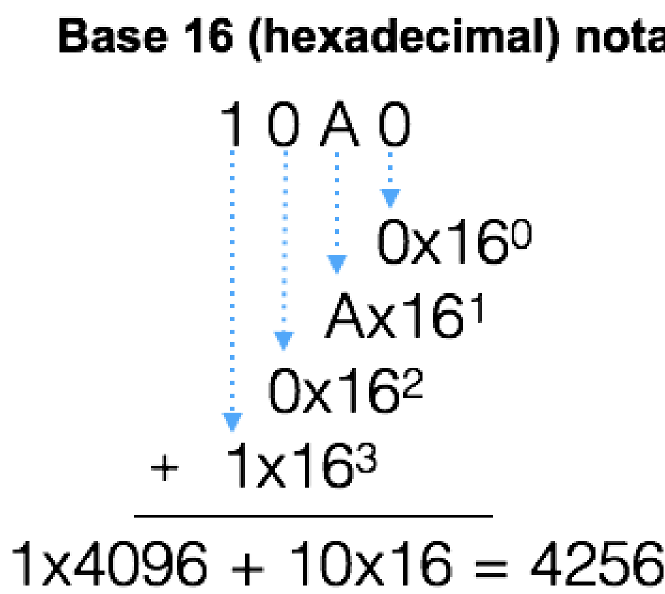


Using binary notation, we can represent any integer.

> **Test yourself!** How is the binary number 101010 represented in decimal notation?
>
> **Test yourself!** What is 327 in binary?
>
> **Test yourself!** What is the largest number you can represent with n bits?

2.b. Hexadecimal: a more convenient notation

As you can see from the above example, binary notation can get unwieldy fast. Instead, we often use base 16, also known as **hexadecimal**:



This works the same way as binary, and base 10, but using "16" as the base. The one additional trick is that since we need 16 different digits, we use

0,1,2,3,4,5,6,78,9,A,B,C,D,E,F

where A=10, B=11, C=12, D=13, E=14, and F=15.

Often, a hexadecimal number is written with `0x` before it to distinguish it from a decimal number.

> **Test yourself!** What is 0xFF in decimal?
>
> **Test yourself!** What is 0xF2 in *binary*?
>
> **Test yourself!** What is 256 in hexadecimal?
>
> **Test yourself!** What is 515 in base *8* (also known as octal)?
>
> **Test yourself!** Why is base 16 often used instead of (say) base 20 or 6?
>
> **Thinking ahead:** How might you represent negative numbers? What about a real number between 0 and 1.

2.c. Using numbers to represent text

Obviously, computers can do more than arithmetic: they can operate on text, graphics, music, and many other kinds of data. How do they do this?

A **string** is a sequence of **characters** in sequence:

H	e	l	l	o	,		W	o	r	l	d	!

Each item in a string is called a *character*.

If a computer can only deal with 0s and 1s, how does it represent an "H"?

The answer is we assign every character to integer, and everyone agrees on the integer for "H" (and every other letter). In other words, we **map** each character to an integer.

An early standard for doing this was ASCII ("American Standard Code for Information Interchange", pronounced *askey*):

Binary	Dec	Glyph	Binary	Dec	Glyph	Binary	Dec	Glyph
010 0000	32	(space)	100 0000	64	@	110 0000	96	`
010 0001	33	!	100 0001	65	A	110 0001	97	a
010 0010	34	"	100 0010	66	B	110 0010	98	b
010 0011	35	#	100 0011	67	C	110 0011	99	c
010 0100	36	$	100 0100	68	D	110 0100	100	d
010 0101	37	%	100 0101	69	E	110 0101	101	e
010 0110	38	&	100 0110	70	F	110 0110	102	f
010 0111	39	'	100 0111	71	G	110 0111	103	g
010 1000	40	(	100 1000	72	H	110 1000	104	h
010 1001	41	)	100 1001	73	I	110 1001	105	i
010 1010	42	*	100 1010	74	J	110 1010	106	j
010 1011	43	+	100 1011	75	K	110 1011	107	k
010 1100	44	,	100 1100	76	L	110 1100	108	l
010 1101	45	-	100 1101	77	M	110 1101	109	m
010 1110	46	.	100 1110	78	N	110 1110	110	n
010 1111	47	/	100 1111	79	O	110 1111	111	o
011 0000	48	0	101 0000	80	P	111 0000	112	p
011 0001	49	1	101 0001	81	Q	111 0001	113	q
011 0010	50	2	101 0010	82	R	111 0010	114	r
011 0011	51	3	101 0011	83	S	111 0011	115	s
011 0100	52	4	101 0100	84	T	111 0100	116	t
011 0101	53	5	101 0101	85	U	111 0101	117	u
011 0110	54	6	101 0110	86	V	111 0110	118	v
011 0111	55	7	101 0111	87	W	111 0111	119	w
011 1000	56	8	101 1000	88	X	111 1000	120	x
011 1001	57	9	101 1001	89	Y	111 1001	121	y
011 1010	58	:	101 1010	90	Z	111 1010	122	z
011 1011	59	;	101 1011	91	[	111 1011	123	{
011 1100	60	<	101 1100	92	\	111 1100	124	|
011 1101	61	=	101 1101	93	]	111 1101	125	}
011 1110	62	>	101 1110	94	^	111 1110	126	~
011 1111	63	?	101 1111	95	_			

You don't need to know those numbers.

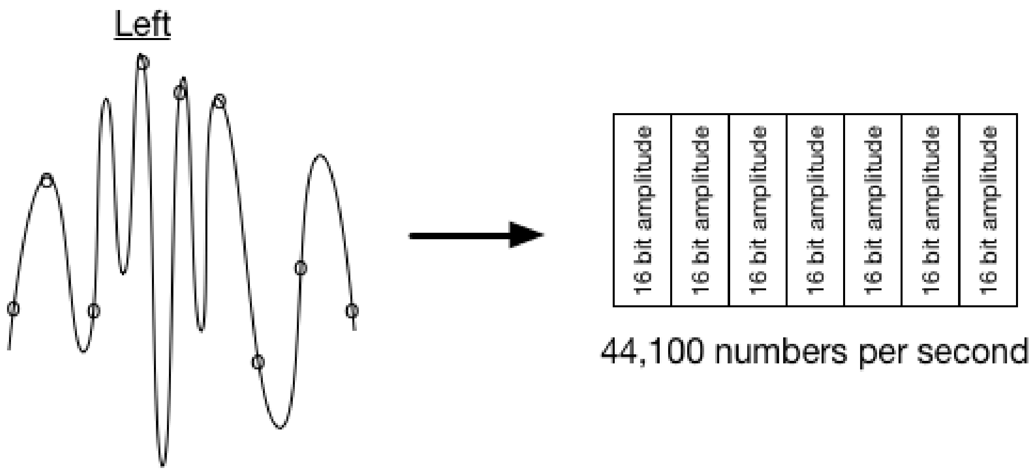
Why is 'a' mapped to '97'? No particular reason. It's just what everyone agreed on.

Nowadays, the much more extensive Unicode standard is used. It maps thousands of letters to integers. For example 0x1f601 = 😁 .

2.d. Digital representation of sound

Sound is a continuous motion of air. How can a computer record, playback, and manipulate it?

The sound wave is digitized, and recorded using a format that everyone agrees on. In this way, sound is turned into a stream of integers:



The number of samples per second limits the frequency that the signal can represent. The number of bits per sample limits the aplitudes that can be recorded. This digitization process introduces some error due to the inability to represent perfectly a continuous signal.

There are many different audio encoding standards, but each boils down to representing the wave as a sequence of numbers.

2.e Digital representation of images

We face th same problem with images: they are a continuum of light across a continuum of colors.

Representing colors:

One encoding for colors is to map them to 3 numbers:

- The amount of green in the color
- The amount of red in the color
- The amount of blue in the color

Each of the above 3 numbers might be represented by an 8 bit integer (i.e. a number between 0 and 255). So that:

PURPLE is 170 red, 0 green, 255 blue

This is called an RGB encoding (for "red, green, blue").

These colors are often written in hexadecimal, so that the above is 0xAA00FF, where the first 2 digits are the amount of red, the second 2 digits are the amount of green, and the last two digits are the amount of blue.

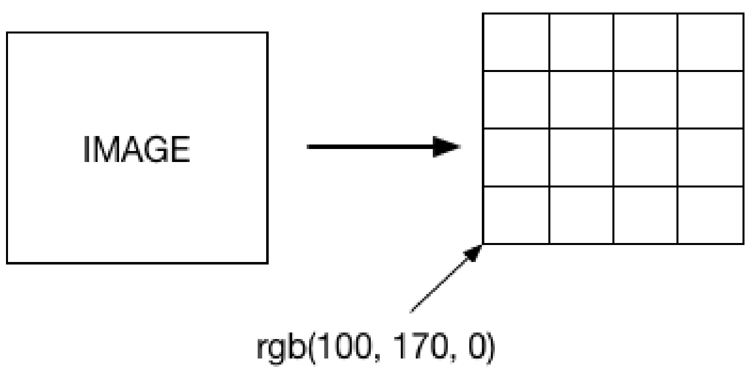
(In HTML, this would be written as `#AA00FF`, where the `#` denotes hexadecimal.)

There are many color encoding schemes, some that use other colors besides red, green, and blue.

> **Test yourself!** What color is `#000000`? Can you find out? What about `#FFFFFF`?
>
> **Test yourself!** What is the code for a gray color using this RGB encoding scheme?

Representing images:

An image is a 2-dimensional matrix of **pixels**, which are digitized units of light. The color of the light at each pixel is encoded using some color encoding such as RGB:



The **resolution** of the image is the dimension of the pixel matrix. A 10 megapixel camera has sensors for around 10 million pixels.

3. Summary

- Computers represent data in binary.
- The data can be stored in memory or in the CPU registers.
- The CPU can perform relatively simplistic operations on the data.
- Real impact of computers come from representing rich, continuous, real-world data as a sequence of bits.
- This is done by agreeing on a mapping from the real world to a sequence of bits.

4. Glossary

- CPU: the central processing unit that can execute instructions.
- RAM: random access memory that stores results of computation, can be accessed by address.
- register: a storage unit in the CPU that is comprised of several bits.
- word: a unit of RAM that can be accessed.
- address: the "name" or location of a word in RAM.
- bit: a single storage unit that can be either on or off.
- byte: 8 bits.
- base 2 or binary: representation of numbers using only the digits 0 and 1.
- hexadecimal: representation of numbers using the digits 0-9 and A-F.
- octal: representation of numbers using the digits 0-7.
- "0x": a prefix commonly written before hexadecimal numbers so they are not mistaken for decimal or octal numbers.
- string: a sequence of characters.
- character: a single letter or symbol within a string.
- ASCII: a standard for mapping characters to integers.
- Unicode: a modern standard for mapping characters, emoji, etc. to integers.
- RGB: stands for "red, green, blue" -- an encoding scheme for colors.
- pixel: a unit of a digitized image.