

# Data Representation

15-110 – Friday 01/16

# Announcements

- No office hours or lecture on Monday (MLK Day)
- Check1 is due **Tuesday at noon**

# Learning Objectives

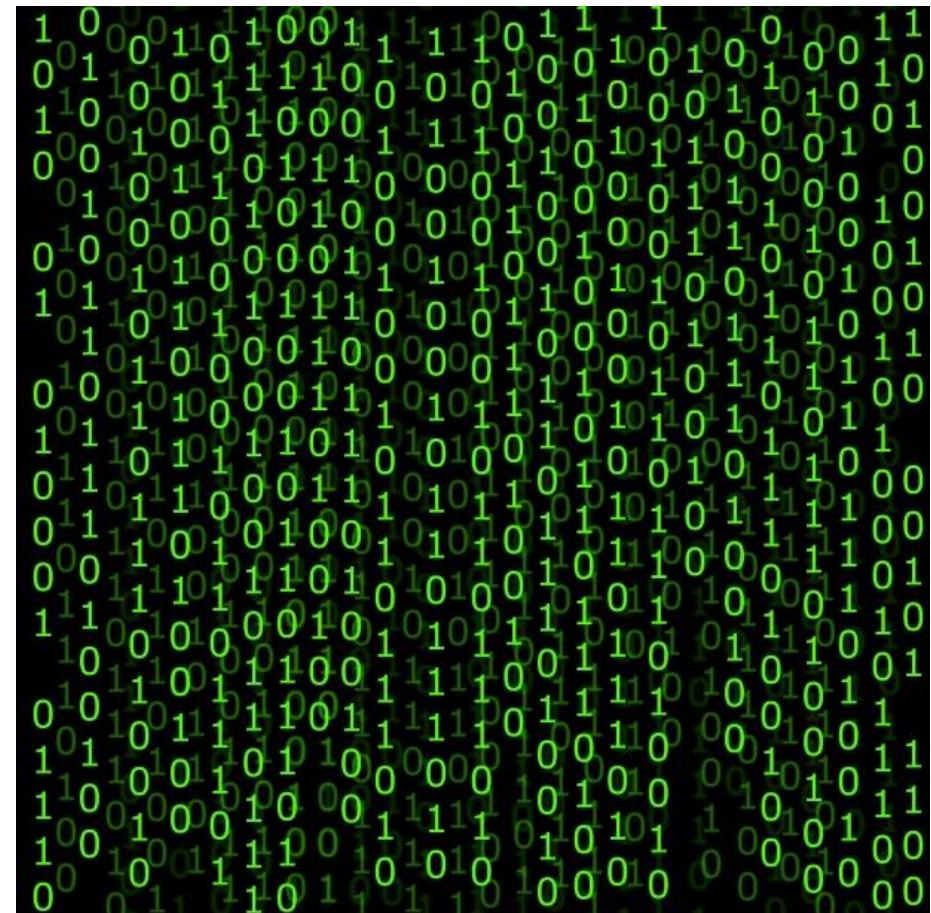
- Understand how different **number systems** can represent the same information
- Translate **binary numbers** to decimal, and vice versa
- Interpret binary numbers as abstracted types, including **colors** and **text**

# Number Systems

# Computers Run on 0s and 1s

Computers represent everything by using 0s and 1s. You've likely seen references to this before.

How can we represent text, or images, or sound with 0s and 1s? This brings us back to **abstraction**.



# Abstraction is About Representation

Recall our definition of abstraction from the first lecture:

**Abstraction** is a technique used to make complex systems manageable by changing the amount of detail used to **represent** or interact with the system.

We'll use abstraction to interpret 0s and 1s in a wide variety of ways. We'll start by translating them into decimal numbers.

# Number Systems – Coins

A **number system** is a way of representing numbers using symbols.

One example of a number system is currency. In the US currency system, how much is each of the following symbols worth?



Penny  
1 cent



Nickel  
5 cents



Dime  
10 cents



Quarter  
25 cents

# Number Systems – Dollars

Alternatively, we can represent money using **dollars and cents**, in decimal form.

For example, a medium coffee at La Prima Cafe is **\$2.50**.



# Converting Coins to Dollars

We can **convert between number systems** by translating a value from one system to the other.

For example, the coins on the left represent the same value as \$0.87

Using pictures is clunky. Let's make a new representation system for coins.



# Coin Number Representation

To represent coins, we'll make a number with four digits.

The first represents quarters, the second dimes, the third nickels, and the fourth pennies.

c.3.1.0.2 =

$$3 * \$0.25 + 1 * \$0.10 + 0 * \$0.05 + 2 * \$0.01 =$$

\$0.87



| Q | D | N | P |
|---|---|---|---|
| c | 3 | 1 | 0 |
| 2 |   |   |   |

# Converting Dollars to Coins

Think about an algorithm to convert from a dollar amount to coins, using as few coins as possible.

**You do:** How would you begin such an algorithm?

# Conversion Example

What is \$0.59 in coin representation?

$$\$0.59 = 2 * \$0.25 + 0 * \$0.10 + 1 * \$0.05 + 4 * \$0.01 = \text{c.2.0.1.4}$$

# Activity: Coin Conversion

**You do:** Now try the following calculations:

What is c.1.1.1.2 in dollars?

What is \$0.61 in coin representation?

# Number Systems - Decimal

When we work with ordinary numbers outside of any specific context, we usually use the **decimal** number system, which uses digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

Moving from the right, the first digit is the number of 1s, the second is 10s, the third is 100s, etc. Each digit represents a **power of 10**. For example, 1980 in decimal is  $1 * 1000 + 9 * 100 + 8 * 10 + 0 * 1$

But this isn't the only abstract number system we can use!

| $10^3$ | $10^2$ | $10^1$ | $10^0$ |
|--------|--------|--------|--------|
| 1000   | 100    | 10     | 1      |
| 1      | 9      | 8      | 0      |

# Number Systems – Binary

We can represent numbers using only the digits 0 and 1 with the **binary number system**.

Instead of counting the number of 1s, 5s, 10s, and 25s in coins, or 1s, 10s, 100s, and 1000s in abstract amounts, count the number of 1s, 2s, 4s, 8s, etc. For example, 1101 in binary is  $1 * 8 + 1 * 4 + 0 * 2 + 1 * 1 = 13$  in decimal.

Why these numbers? They're **powers of 2**. This is a number in **base 2**, which only needs the digits 0 and 1.

|       |   |       |   |       |   |       |   |
|-------|---|-------|---|-------|---|-------|---|
| $2^3$ | 8 | $2^2$ | 4 | $2^1$ | 2 | $2^0$ | 1 |
| 1     |   | 1     |   | 0     |   | 1     |   |

# Bits and Bytes

When working with binary and computers, we often refer to a set of binary values used together to represent a number.

A single binary value is called a **bit**.

A set of 8 bits is called a **byte**.

We commonly use some number of **bytes** to represent data values.

See more:

[https://www.advanced-ict.info/javascript/binary\\_counter.html](https://www.advanced-ict.info/javascript/binary_counter.html)

# Counting in Binary

0 =

| $2^7$<br>128 | $2^6$<br>64 | $2^5$<br>32 | $2^4$<br>16 | $2^3$<br>8 | $2^2$<br>4 | $2^1$<br>2 | $2^0$<br>1 |
|--------------|-------------|-------------|-------------|------------|------------|------------|------------|
| 0            | 0           | 0           | 0           | 0          | 0          | 0          | 0          |

2 =

| $2^7$<br>128 | $2^6$<br>64 | $2^5$<br>32 | $2^4$<br>16 | $2^3$<br>8 | $2^2$<br>4 | $2^1$<br>2 | $2^0$<br>1 |
|--------------|-------------|-------------|-------------|------------|------------|------------|------------|
| 0            | 0           | 0           | 0           | 0          | 0          | 1          | 0          |

4 =

| $2^7$<br>128 | $2^6$<br>64 | $2^5$<br>32 | $2^4$<br>16 | $2^3$<br>8 | $2^2$<br>4 | $2^1$<br>2 | $2^0$<br>1 |
|--------------|-------------|-------------|-------------|------------|------------|------------|------------|
| 0            | 0           | 0           | 0           | 0          | 1          | 0          | 0          |

6 =

| $2^7$<br>128 | $2^6$<br>64 | $2^5$<br>32 | $2^4$<br>16 | $2^3$<br>8 | $2^2$<br>4 | $2^1$<br>2 | $2^0$<br>1 |
|--------------|-------------|-------------|-------------|------------|------------|------------|------------|
| 0            | 0           | 0           | 0           | 0          | 1          | 1          | 0          |

1 =

| $2^7$<br>128 | $2^6$<br>64 | $2^5$<br>32 | $2^4$<br>16 | $2^3$<br>8 | $2^2$<br>4 | $2^1$<br>2 | $2^0$<br>1 |
|--------------|-------------|-------------|-------------|------------|------------|------------|------------|
| 0            | 0           | 0           | 0           | 0          | 0          | 0          | 1          |

3 =

| $2^7$<br>128 | $2^6$<br>64 | $2^5$<br>32 | $2^4$<br>16 | $2^3$<br>8 | $2^2$<br>4 | $2^1$<br>2 | $2^0$<br>1 |
|--------------|-------------|-------------|-------------|------------|------------|------------|------------|
| 0            | 0           | 0           | 0           | 0          | 0          | 1          | 1          |

5 =

| $2^7$<br>128 | $2^6$<br>64 | $2^5$<br>32 | $2^4$<br>16 | $2^3$<br>8 | $2^2$<br>4 | $2^1$<br>2 | $2^0$<br>1 |
|--------------|-------------|-------------|-------------|------------|------------|------------|------------|
| 0            | 0           | 0           | 0           | 0          | 1          | 0          | 1          |

7 =

| $2^7$<br>128 | $2^6$<br>64 | $2^5$<br>32 | $2^4$<br>16 | $2^3$<br>8 | $2^2$<br>4 | $2^1$<br>2 | $2^0$<br>1 |
|--------------|-------------|-------------|-------------|------------|------------|------------|------------|
| 0            | 0           | 0           | 0           | 0          | 1          | 1          | 1          |

# Converting Binary to Decimal

To convert a binary number to decimal, just add each power of 2 that is represented by a 1.

For example,  $00011000 = 16 + 8 = 24$

| $2^7$ | 128 | $2^6$ | 64 | $2^5$ | 32 | $2^4$ | 16 | $2^3$ | 8 | $2^2$ | 4 | $2^1$ | 2 | $2^0$ | 1 |
|-------|-----|-------|----|-------|----|-------|----|-------|---|-------|---|-------|---|-------|---|
|       | 0   |       | 0  |       | 0  |       | 1  |       | 1 |       | 0 |       | 0 |       | 0 |

Another example:

$10010001 = 128 + 16 + 1 = 145$

| $2^7$ | 128 | $2^6$ | 64 | $2^5$ | 32 | $2^4$ | 16 | $2^3$ | 8 | $2^2$ | 4 | $2^1$ | 2 | $2^0$ | 1 |
|-------|-----|-------|----|-------|----|-------|----|-------|---|-------|---|-------|---|-------|---|
|       | 1   |       | 0  |       | 0  |       | 1  |       | 0 |       | 0 |       | 0 |       | 1 |

# Converting Decimal to Binary

Converting decimal to binary uses the **same process** as converting dollars to coins.

Look for the largest power of 2 that can fit in the number and subtract it from the number. Repeat with the next power of 2, etc., until you reach 0.

For example,  $36 = 32 + 4 \rightarrow 00100100$

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| 128   | 64    | 32    | 16    | 8     | 4     | 2     | 1     |
| 0     | 0     | 1     | 0     | 0     | 1     | 0     | 0     |

Another example:

$$\begin{aligned} 103 &= 64 + 39 = 64 + 32 + 7 = \\ &64 + 32 + 4 + 3 = 64 + 32 + 4 + 2 + 1 \rightarrow \\ &01100111 \end{aligned}$$

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| 128   | 64    | 32    | 16    | 8     | 4     | 2     | 1     |
| 0     | 1     | 1     | 0     | 0     | 1     | 1     | 1     |

# Activity: Converting Binary

**You do:** Now try converting numbers on your own.

First: what is **01011011** in decimal?

Second: what is **75** in binary?

# Abstracted Types

# Binary and Abstraction

Now that we can represent numbers using binary, we can represent **everything** computers store using binary.

We just need to use **abstraction** to interpret bits or numbers in particular ways.

Let's consider dates, images, and text.

# Discussion: Representing Dates

It can be helpful to think logically about how to represent a value before learning how it's done in practice. Let's do that now.

**Discuss:** We can convert binary directly into numbers, but how could we use binary and number to represent a date (i.e., 01/17/2025)?

# Representing Dates – Simple Approach

Use 4 bits to represent the month (1-12), 5 bits to represent the day (1-31) and 12 bits to represent the year (1-4095). Convert the month, day, year normally from decimal to binary.

| Month   | Day | Year |
|---|-----|------|
| 01  | 17  | 2025 |
|  |     |      |

# Representing Dates – Actual Approach

In the commonly used Unix Timestamp approach, you count the seconds from a certain date (00:00:00 of 01/01/1970) and convert the number of seconds to binary. Any dates that occur before this time would be negative numbers, and any dates after would be positive numbers!

We use 32 bits to represent each date; the first bit is used to indicate if the number was positive (0) or negative (1), and the remaining 31 bits are used to represent the number of seconds elapsed. Thus, we restrict the number of bits to represent the date to 31 bits.

01/17/2025 2pm -> 1737140400 seconds

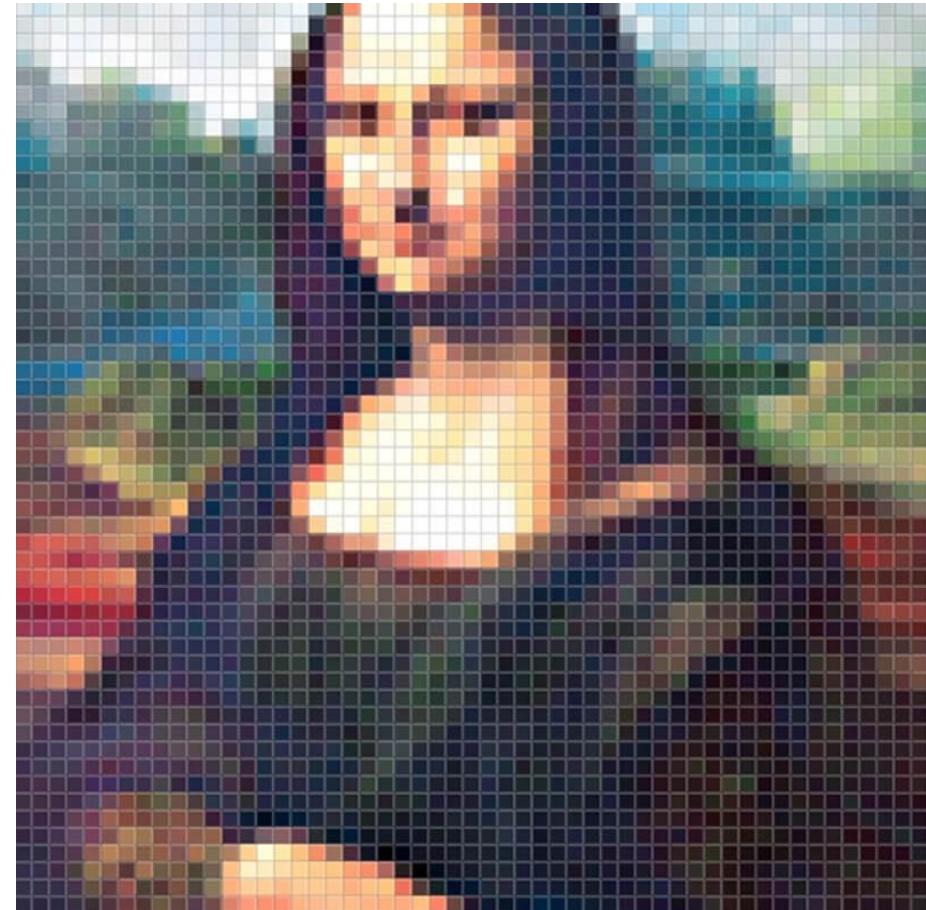
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

# Represent Images as Grids of Colors

What if we want to represent an image?  
How can we convert that to numbers?

First, break the image down into a grid of colors, where each square of color has a distinct hue. A square of color in this context is called a **pixel**.

If we can represent a pixel in binary, we can interpret a series of pixels as an image.

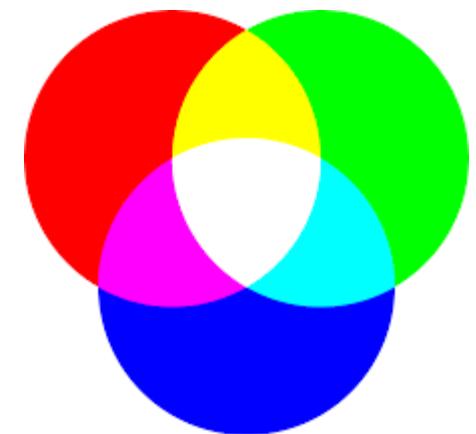


# Representing Colors in Binary

We need to represent a single color (a pixel) as a number.

There are a few ways to do this, but we'll focus on **RGB**. Any color can be represented as a combination of Red, Green, and Blue.

Red, green, and blue intensity can be represented using one **byte** each, where 00000000 (0) is none and 11111111 (255) is very intense. Each pixel will therefore require 3 bytes to encode.



Try it out here:

[https://www.w3schools.com/colors/colors\\_rgb.asp](https://www.w3schools.com/colors/colors_rgb.asp)

# Example: Representing Beige

To make the campus-building beige, we'd need:

**Red = 249 = 11111001**



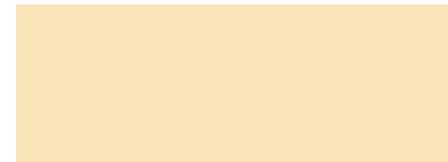
**Green = 228 = 11100100**



**Blue = 183 = 10110111**



**Which makes beige!**



# Represent Text as Individual Characters

Finally, how do we represent text?

First, we break it down into smaller parts, like with images. In this case, we can break text down into individual **characters**.

For example, the text "Hello World" becomes

|   |   |   |   |   |       |   |   |   |   |   |
|---|---|---|---|---|-------|---|---|---|---|---|
| H | e | I | I | o | space | W | o | r | I | d |
|---|---|---|---|---|-------|---|---|---|---|---|

# Use a Lookup Table to Convert Characters

Unlike colors, characters don't have a natural connection to numbers.

Instead, we can use a **lookup table** that maps each possible character to an integer.

As long as every computer uses the same lookup table, computers can always translate a set of numbers into the same set of characters.

# ASCII is a Simple Lookup Table

For a small set of characters, we can use the encoding system called ASCII. This maps the numbers 0 to 255 to characters. Therefore, one character is represented by one byte.

Check it out here:  
<https://www.asciiitable.com/>

| Dec   | Hex | Oct                 | Chr | Dec   | Hex | Oct    | HTML  | Chr   | Dec | Hex    | Oct | HTML   | Chr | Dec    | Hex | Oct | HTML | Chr |
|-------|-----|---------------------|-----|-------|-----|--------|-------|-------|-----|--------|-----|--------|-----|--------|-----|-----|------|-----|
| 0 0   | 000 | NULL                |     | 32 20 | 040 | &#032; | Space | 64 40 | 100 | &#064; | @   | 96 60  | 140 | &#096; | `   |     |      |     |
| 1 1   | 001 | Start of Header     |     | 33 21 | 041 | &#033; | !     | 65 41 | 101 | &#065; | A   | 97 61  | 141 | &#097; | a   |     |      |     |
| 2 2   | 002 | Start of Text       |     | 34 22 | 042 | &#034; | "     | 66 42 | 102 | &#066; | B   | 98 62  | 142 | &#098; | b   |     |      |     |
| 3 3   | 003 | End of Text         |     | 35 23 | 043 | &#035; | #     | 67 43 | 103 | &#067; | C   | 99 63  | 143 | &#099; | c   |     |      |     |
| 4 4   | 004 | End of Transmission |     | 36 24 | 044 | &#036; | \$    | 68 44 | 104 | &#068; | D   | 100 64 | 144 | &#100; | d   |     |      |     |
| 5 5   | 005 | Enquiry             |     | 37 25 | 045 | &#037; | %     | 69 45 | 105 | &#069; | E   | 101 65 | 145 | &#101; | e   |     |      |     |
| 6 6   | 006 | Acknowledgment      |     | 38 26 | 046 | &#038; | &     | 70 46 | 106 | &#070; | F   | 102 66 | 146 | &#102; | f   |     |      |     |
| 7 7   | 007 | Bell                |     | 39 27 | 047 | &#039; | '     | 71 47 | 107 | &#071; | G   | 103 67 | 147 | &#103; | g   |     |      |     |
| 8 8   | 010 | Backspace           |     | 40 28 | 050 | &#040; | (     | 72 48 | 110 | &#072; | H   | 104 68 | 150 | &#104; | h   |     |      |     |
| 9 9   | 011 | Horizontal Tab      |     | 41 29 | 051 | &#041; | )     | 73 49 | 111 | &#073; | I   | 105 69 | 151 | &#105; | i   |     |      |     |
| 10 A  | 012 | Line feed           |     | 42 2A | 052 | &#042; | *     | 74 4A | 112 | &#074; | J   | 106 6A | 152 | &#106; | j   |     |      |     |
| 11 B  | 013 | Vertical Tab        |     | 43 2B | 053 | &#043; | +     | 75 4B | 113 | &#075; | K   | 107 6B | 153 | &#107; | k   |     |      |     |
| 12 C  | 014 | Form feed           |     | 44 2C | 054 | &#044; | ,     | 76 4C | 114 | &#076; | L   | 108 6C | 154 | &#108; | l   |     |      |     |
| 13 D  | 015 | Carriage return     |     | 45 2D | 055 | &#045; | -     | 77 4D | 115 | &#077; | M   | 109 6D | 155 | &#109; | m   |     |      |     |
| 14 E  | 016 | Shift Out           |     | 46 2E | 056 | &#046; | .     | 78 4E | 116 | &#078; | N   | 110 6E | 156 | &#110; | n   |     |      |     |
| 15 F  | 017 | Shift In            |     | 47 2F | 057 | &#047; | /     | 79 4F | 117 | &#079; | O   | 111 6F | 157 | &#111; | o   |     |      |     |
| 16 10 | 020 | Data Link Escape    |     | 48 30 | 060 | &#048; | 0     | 80 50 | 120 | &#080; | P   | 112 70 | 160 | &#112; | p   |     |      |     |
| 17 11 | 021 | Device Control 1    |     | 49 31 | 061 | &#049; | 1     | 81 51 | 121 | &#081; | Q   | 113 71 | 161 | &#113; | q   |     |      |     |
| 18 12 | 022 | Device Control 2    |     | 50 32 | 062 | &#050; | 2     | 82 52 | 122 | &#082; | R   | 114 72 | 162 | &#114; | r   |     |      |     |
| 19 13 | 023 | Device Control 3    |     | 51 33 | 063 | &#051; | 3     | 83 53 | 123 | &#083; | S   | 115 73 | 163 | &#115; | s   |     |      |     |
| 20 14 | 024 | Device Control 4    |     | 52 34 | 064 | &#052; | 4     | 84 54 | 124 | &#084; | T   | 116 74 | 164 | &#116; | t   |     |      |     |
| 21 15 | 025 | Negative Ack.       |     | 53 35 | 065 | &#053; | 5     | 85 55 | 125 | &#085; | U   | 117 75 | 165 | &#117; | u   |     |      |     |
| 22 16 | 026 | Synchronous idle    |     | 54 36 | 066 | &#054; | 6     | 86 56 | 126 | &#086; | V   | 118 76 | 166 | &#118; | v   |     |      |     |
| 23 17 | 027 | End of Trans. Block |     | 55 37 | 067 | &#055; | 7     | 87 57 | 127 | &#087; | W   | 119 77 | 167 | &#119; | w   |     |      |     |
| 24 18 | 030 | Cancel              |     | 56 38 | 070 | &#056; | 8     | 88 58 | 130 | &#088; | X   | 120 78 | 170 | &#120; | x   |     |      |     |
| 25 19 | 031 | End of Medium       |     | 57 39 | 071 | &#057; | 9     | 89 59 | 131 | &#089; | Y   | 121 79 | 171 | &#121; | y   |     |      |     |
| 26 1A | 032 | Substitute          |     | 58 3A | 072 | &#058; | :     | 90 5A | 132 | &#090; | Z   | 122 7A | 172 | &#122; | z   |     |      |     |
| 27 1B | 033 | Escape              |     | 59 3B | 073 | &#059; | ;     | 91 5B | 133 | &#091; | [   | 123 7B | 173 | &#123; | {   |     |      |     |
| 28 1C | 034 | File Separator      |     | 60 3C | 074 | &#060; | <     | 92 5C | 134 | &#092; | \   | 124 7C | 174 | &#124; |     |     |      |     |
| 29 1D | 035 | Group Separator     |     | 61 3D | 075 | &#061; | =     | 93 5D | 135 | &#093; | ]   | 125 7D | 175 | &#125; | }   |     |      |     |
| 30 1E | 036 | Record Separator    |     | 62 3E | 076 | &#062; | >     | 94 5E | 136 | &#094; | ^   | 126 7E | 176 | &#126; | ~   |     |      |     |
| 31 1F | 037 | Unit Separator      |     | 63 3F | 077 | &#063; | ?     | 95 5F | 137 | &#095; | _   | 127 7F | 177 | &#127; | Del |     |      |     |

asciichars.com

# Translating Text to Numbers

"Yay" ->

|   |   |   |
|---|---|---|
| Y | a | y |
|---|---|---|

->

89 97 121 ->

01011001 01100001  
01111001

| Dec | Hex | Oct | Chr                 | Dec | Hex | Oct | HTML   | Chr   | Dec | Hex | Oct | HTML   | Chr | Dec | Hex | Oct | HTML   | Chr |
|-----|-----|-----|---------------------|-----|-----|-----|--------|-------|-----|-----|-----|--------|-----|-----|-----|-----|--------|-----|
| 0   | 00  | 000 | NULL                | 32  | 20  | 040 | &#032; | Space | 64  | 40  | 100 | &#064; | @   | 96  | 60  | 140 | &#096; | '   |
| 1   | 1   | 001 | Start of Header     | 33  | 21  | 041 | &#033; | !     | 65  | 41  | 101 | &#065; | A   | 97  | 61  | 141 | &#097; | a   |
| 2   | 2   | 002 | Start of Text       | 34  | 22  | 042 | &#034; | "     | 66  | 42  | 102 | &#066; | B   | 98  | 62  | 142 | &#098; | b   |
| 3   | 3   | 003 | End of Text         | 35  | 23  | 043 | &#035; | #     | 67  | 43  | 103 | &#067; | C   | 99  | 63  | 143 | &#099; | c   |
| 4   | 4   | 004 | End of Transmission | 36  | 24  | 044 | &#036; | \$    | 68  | 44  | 104 | &#068; | D   | 100 | 64  | 144 | &#100; | d   |
| 5   | 5   | 005 | Enquiry             | 37  | 25  | 045 | &#037; | %     | 69  | 45  | 105 | &#069; | E   | 101 | 65  | 145 | &#101; | e   |
| 6   | 6   | 006 | Acknowledgment      | 38  | 26  | 046 | &#038; | &     | 70  | 46  | 106 | &#070; | F   | 102 | 66  | 146 | &#102; | f   |
| 7   | 7   | 007 | Bell                | 39  | 27  | 047 | &#039; | '     | 71  | 47  | 107 | &#071; | G   | 103 | 67  | 147 | &#103; | g   |
| 8   | 8   | 010 | Backspace           | 40  | 28  | 050 | &#040; | (     | 72  | 48  | 110 | &#072; | H   | 104 | 68  | 150 | &#104; | h   |
| 9   | 9   | 011 | Horizontal Tab      | 41  | 29  | 051 | &#041; | )     | 73  | 49  | 111 | &#073; | I   | 105 | 69  | 151 | &#105; | i   |
| 10  | A   | 012 | Line feed           | 42  | 2A  | 052 | &#042; | *     | 74  | 4A  | 112 | &#074; | J   | 106 | 6A  | 152 | &#106; | j   |
| 11  | B   | 013 | Vertical Tab        | 43  | 2B  | 053 | &#043; | +     | 75  | 4B  | 113 | &#075; | K   | 107 | 6B  | 153 | &#107; | k   |
| 12  | C   | 014 | Form feed           | 44  | 2C  | 054 | &#044; | ,     | 76  | 4C  | 114 | &#076; | L   | 108 | 6C  | 154 | &#108; | l   |
| 13  | D   | 015 | Carriage return     | 45  | 2D  | 055 | &#045; | -     | 77  | 4D  | 115 | &#077; | M   | 109 | 6D  | 155 | &#109; | m   |
| 14  | E   | 016 | Shift Out           | 46  | 2E  | 056 | &#046; | .     | 78  | 4E  | 116 | &#078; | N   | 110 | 6E  | 156 | &#110; | n   |
| 15  | F   | 017 | Shift In            | 47  | 2F  | 057 | &#047; | /     | 79  | 4F  | 117 | &#079; | O   | 111 | 6F  | 157 | &#111; | o   |
| 16  | 10  | 020 | Data Link Escape    | 48  | 30  | 060 | &#048; | 0     | 80  | 50  | 120 | &#080; | P   | 112 | 70  | 160 | &#112; | p   |
| 17  | 11  | 021 | Device Control 1    | 49  | 31  | 061 | &#049; | 1     | 81  | 51  | 121 | &#081; | Q   | 113 | 71  | 161 | &#113; | q   |
| 18  | 12  | 022 | Device Control 2    | 50  | 32  | 062 | &#050; | 2     | 82  | 52  | 122 | &#082; | R   | 114 | 72  | 162 | &#114; | r   |
| 19  | 13  | 023 | Device Control 3    | 51  | 33  | 063 | &#051; | 3     | 83  | 53  | 123 | &#083; | S   | 115 | 73  | 163 | &#115; | s   |
| 20  | 14  | 024 | Device Control 4    | 52  | 34  | 064 | &#052; | 4     | 84  | 54  | 124 | &#084; | T   | 116 | 74  | 164 | &#116; | t   |
| 21  | 15  | 025 | Negative Ack.       | 53  | 35  | 065 | &#053; | 5     | 85  | 55  | 125 | &#085; | U   | 117 | 75  | 165 | &#117; | u   |
| 22  | 16  | 026 | Synchronous idle    | 54  | 36  | 066 | &#054; | 6     | 86  | 56  | 126 | &#086; | V   | 118 | 76  | 166 | &#118; | v   |
| 23  | 17  | 027 | End of Trans. Block | 55  | 37  | 067 | &#055; | 7     | 87  | 57  | 127 | &#087; | W   | 119 | 77  | 167 | &#119; | w   |
| 24  | 18  | 030 | Cancel              | 56  | 38  | 070 | &#056; | 8     | 88  | 58  | 130 | &#088; | X   | 120 | 78  | 170 | &#120; | x   |
| 25  | 19  | 031 | End of Medium       | 57  | 39  | 071 | &#057; | 9     | 89  | 59  | 131 | &#089; | Y   | 121 | 79  | 171 | &#121; | y   |
| 26  | 1A  | 032 | Substitute          | 58  | 3A  | 072 | &#058; | :     | 90  | 5A  | 132 | &#090; | Z   | 122 | 7A  | 172 | &#122; | z   |
| 27  | 1B  | 033 | Escape              | 59  | 3B  | 073 | &#059; | ;     | 91  | 5B  | 133 | &#091; | [   | 123 | 7B  | 173 | &#123; | {   |
| 28  | 1C  | 034 | File Separator      | 60  | 3C  | 074 | &#060; | <     | 92  | 5C  | 134 | &#092; | \   | 124 | 7C  | 174 | &#124; |     |
| 29  | 1D  | 035 | Group Separator     | 61  | 3D  | 075 | &#061; | =     | 93  | 5D  | 135 | &#093; | ]   | 125 | 7D  | 175 | &#125; | }   |
| 30  | 1E  | 036 | Record Separator    | 62  | 3E  | 076 | &#062; | >     | 94  | 5E  | 136 | &#094; | ^   | 126 | 7E  | 176 | &#126; | ~   |
| 31  | 1F  | 037 | Unit Separator      | 63  | 3F  | 077 | &#063; | ?     | 95  | 5F  | 137 | &#095; | _   | 127 | 7F  | 177 | &#127; | Del |

# For More Characters, Use Unicode

There are many characters that aren't available in ASCII (characters from non-English languages, advanced symbols, emoji...) due to the limited size.

The Unicode system represents every character that can be typed into a computer. It uses up to 5 bytes, which can represent up to 1 trillion characters! Find all the Unicode characters here: [www.unicode-table.com](http://www.unicode-table.com)

The Unicode system is also **actively under development**. The Unicode Consortium regularly updates the standard to add new types of characters and emoji.

**Discuss:** what are the potential repercussions of using a single standard for all text on computers?

# Learning Objectives

- Understand how different **number systems** can represent the same information
- Translate **binary numbers** to decimal, and vice versa
- Interpret binary numbers as abstracted types, including **colors** and **text**

# Bonus Slides

In case you want to learn even more about data representation

# Size of Integers

Your machine is either classified as 32-bit or 64-bit. This refers to the **size of integers** used by your computer's operating system.

The largest signed integer that can be represented with  $N$  bits is  $2^N-1$  (why?). This means that...

Largest int for 32 bits: 4,294,967,295 (or 2,147,483,647 with negative numbers)

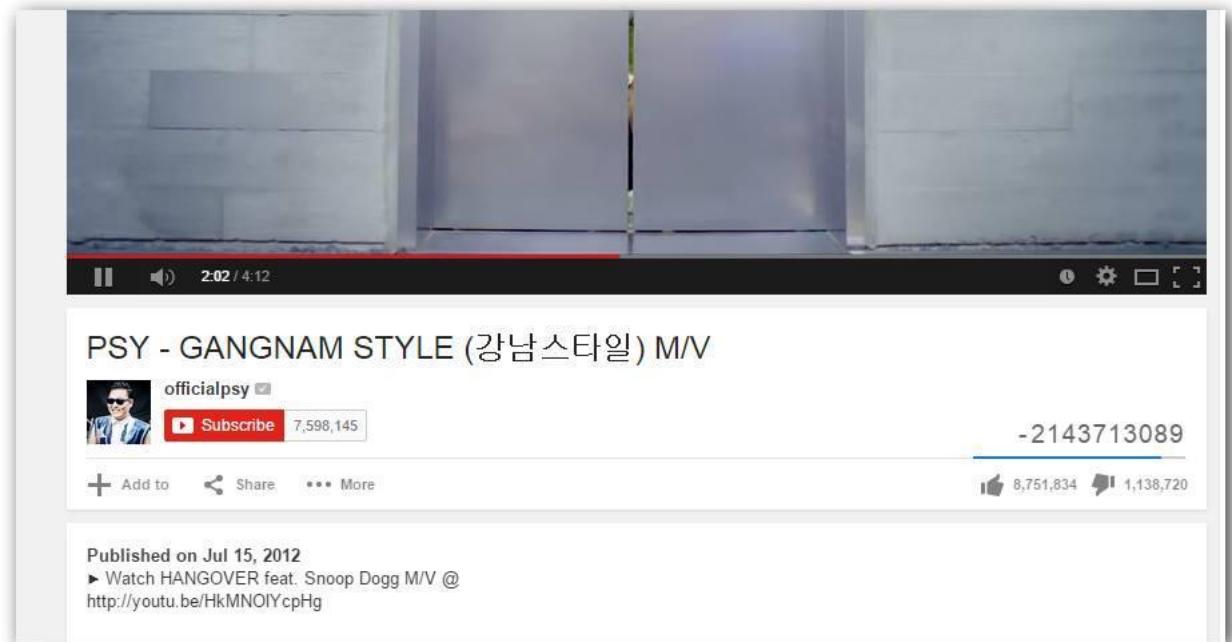
Largest int for 64 bits: 18,446,744,073,709,551,615 (18.4 quintillion)

# Integer Overflow

Why does this matter?

By late 2014, the music video *Gangnam Style* received more than **2 billion** views. When it passed the largest positive number that could be represented with 32 bits, YouTube showed the number of views as **negative** instead!

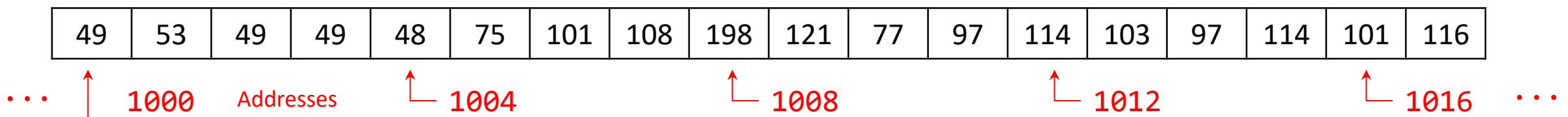
Now YouTube uses a 64-bit counter instead.



# Computer Memory is Stored as Binary

Your computer keeps track of saved data and all the information it needs to run in its **memory**, which is represented as binary. You can think about your computer's memory as a really long list of bits, where each bit can be set to 0 or 1. But usually we think in terms of bytes, groups of 8 bits.

Every byte in your computer has an **address**, which the computer uses to look up its value.



# Binary Values Depend on Interpretation

When you open a file on your computer, the application goes to the appropriate address, reads the associated binary, and **interprets** the binary values based on the file encoding it expects. That interpretation depends on the **application** you use when opening the file, and the **filetype**.

You can attempt to open **any file** using **any program**, if you convince your computer to let you try. Some programs may crash, and others will show nonsense because the binary isn't being interpreted correctly.

**Example:** try changing a .docx filetype to .txt, then open it in a plain text editor. .docx files have extra encoding, whereas .txt files use plain ASCII.

# We Use Lots of Bytes!

In modern computing, we use a **lot** of bytes to represent information.

**Smartphone Memory:** 64 gigabytes = 64 **billion** bytes

**Google databases:** Over 100 million gigabytes = 100 **quadrillion** bytes!

**CMU Wifi:** 15 **million** bytes per second