Representing Letters: ASCII
Representing numbers is great -- but what about words? Can we make sentences with binary data?

- **Key Idea:** Every letter is _____ binary bits.* in ASCII
  (This means that every letter is _____ hex digits.)

- Global standard called the **American Standard Code for Information Interchange (ASCII)** is a ___________ ____________ for translating numbers to characters.

### ASCII Character Encoding Examples:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b 0100 0001</td>
<td>0x41</td>
<td>A</td>
</tr>
<tr>
<td>0b 0100 0010</td>
<td>0x42</td>
<td>B</td>
</tr>
<tr>
<td>0b 0100 0011</td>
<td>0x43</td>
<td>C</td>
</tr>
<tr>
<td>0b 0101 0000</td>
<td>0x61</td>
<td>a</td>
</tr>
<tr>
<td>0b 0110 0001</td>
<td>0x62</td>
<td>b</td>
</tr>
<tr>
<td>0b 0110 0010</td>
<td>0x63</td>
<td>c</td>
</tr>
<tr>
<td>0b 0110 0011</td>
<td>0x64</td>
<td>d</td>
</tr>
<tr>
<td>0b0010 0100</td>
<td>0x24</td>
<td>$</td>
</tr>
<tr>
<td>0b0111 1011</td>
<td>0x7b</td>
<td>{</td>
</tr>
</tbody>
</table>

...and now we can form sentences!

Q: Are there going to be any issues with ASCII?

---

Technical Details of UTF-8 Encoding
UTF-8 uses a ___________ -bit design where each character by be any of the following:

<table>
<thead>
<tr>
<th>Length</th>
<th>Byte #1</th>
<th>Byte #2</th>
<th>Byte #3</th>
<th>Byte #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-byte</td>
<td>0___ ___</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-bytes:</td>
<td>110_ ___</td>
<td>10__ ___</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-bytes:</td>
<td>1110 ___</td>
<td>10__ ___</td>
<td>10__ ___</td>
<td></td>
</tr>
<tr>
<td>4-bytes:</td>
<td>1111 0___</td>
<td>10__ ___</td>
<td>10__ ___</td>
<td>10__ ___</td>
</tr>
</tbody>
</table>

Unicode characters are represented by U+### (where ### is the hex value of the character encoding data) and all 1-byte characters match the ASCII character encoding:

- ‘a’ is ASCII ______, or _______

**Example:** ε (epsilon) is defined as U+03b5. How do we encode this?

**Example:** I received the following binary message encoded in UTF-8:

\[
\begin{array}{cccccccccccc}
0100 & 1000 & 0110 & 1001 & 1111 & 0000 & 1011 & 1111 & 1000 & 1100 & 1000 & 1001 \\
\end{array}
\]

1. What is the hexadecimal representation of this message?

2. What is the **byte length** of this message? ______

3. What is the **character length** of this message? ______

4. What does the message say?

---

Representing Letters: Other Character Encodings
Since ASCII uses only 8 bits, we are limited to only 256 unique characters. There’s far more than 256 characters -- and what about EMOJIs?? 🎉

- **Many** other character encodings exist other than ASCII.
- The most widely used character encoding is known as Unicode Transformation Format (8-bit) or _______.
- Standard is ISO/IEC 10646 (Updated annually!).

---

Example: I received the following binary message encoded in UTF-8:

```
0b0100 0100 0110 1001 1111 0000 1001 1111 1000 1110 1000 1001
```

1. What is the hexadecimal representation of this message?

2. What is the **byte length** of this message? ______

3. What is the **character length** of this message? ______

4. What does the message say?

---

Example: ε (epsilon) is defined as U+03B5. How do we encode this?

---

Example: I received the following binary message encoded in UTF-8:

```
0b0100 0100 0110 1001 1111 0000 1001 1111 1000 1110 1000 1001
```

1. What is the hexadecimal representation of this message?

2. What is the **byte length** of this message? ______

3. What is the **character length** of this message? ______

4. What does the message say?

---

Example: I received the following binary message encoded in UTF-8:

```
0b0100 0100 0110 1001 1111 0000 1001 1111 1000 1110 1000 1001
```

1. What is the hexadecimal representation of this message?

2. What is the **byte length** of this message? ______

3. What is the **character length** of this message? ______

4. What does the message say?
Bit Manipulation: Binary Addition
For the past two lectures we have focused on the first foundation: DATA. Today, we are going to begin the transition away from data and into how data applies to the CPU. Binary addition work just like decimal addition, but with only 0s and 1s:

\[
\begin{align*}
0b \ 010011 & \quad + \quad 0b \ 001001 \\
+ \ 0b \ 001001 & \quad + \quad 0b \ 0111
\end{align*}
\]

\[
0b \ 0011 \\
\quad - \quad 0b \ 0111
\]

Negative Numbers: __________________________

\[
\begin{align*}
0b \ 010011 & \quad - \quad 0b \ 001001 \\
- \ 0b \ 001001 & \quad - \quad 0b \ 0111
\end{align*}
\]

Two’s Complement
The Two’s Complement is a way to represent signed (ex: positive vs. negative) numbers in a way __________________________!

For simplicity, let’s imagine running on an 7-bit machine:

-17 =

-4 =

-1 =

Overflow Detection in Two’s Complement:

Towards Multiplication
With Two’s Complement, we can add and subtract numbers! What about more complex operations?

\[
\begin{align*}
10 \times 2 &= \\
10 \times 4 &= \\
10 \times 9 &=
\end{align*}
\]

Bit Shift Operations:

1. [Left Shift]:

2. [Right Shift]:

42
18
-18
-42
31
-32
+42