Mutability
Objects

(Demo)
Objects
Objects

- Objects represent information
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
Objects

• Objects represent information
• They consist of data and behavior, bundled together to create abstractions
• Objects can represent things, but also properties, interactions, & processes
• A type of object is called a class; classes are first-class values in Python
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
  - A metaphor for organizing large programs
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
  - A metaphor for organizing large programs
  - Special syntax that can improve the composition of programs
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
  - A metaphor for organizing large programs
  - Special syntax that can improve the composition of programs
- In Python, every value is an object
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
  - A metaphor for organizing large programs
  - Special syntax that can improve the composition of programs
- In Python, every value is an object
  - All **objects** have **attributes**
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; classes are first-class values in Python
- Object-oriented programming:
  - A metaphor for organizing large programs
  - Special syntax that can improve the composition of programs
- In Python, every value is an object
  - All objects have attributes
  - A lot of data manipulation happens through object methods
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
  - A metaphor for organizing large programs
  - Special syntax that can improve the composition of programs
- In Python, every value is an object
  - All **objects** have **attributes**
  - A lot of data manipulation happens through object **methods**
  - Functions do one thing; objects do many related things
Example: Strings

(Demo)
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>~</td>
</tr>
</tbody>
</table>
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
</tbody>
</table>

8 rows: 3 bits

8 rows: 3 bits

0 1 2 3 4 5 6 7 8 9 A B C D E F

@ A B C D E F G H I J K L M N O

P Q R S T U V W X Y Z [ \ ] ^ _ ~

` a b c d e f g h i j k l m n o

p q r s t u v w x y z { | } ~ DEL
### Representing Strings: the ASCII Standard

The American Standard Code for Information Interchange (ASCII) is a standard that represents text using 8-bit sequences. The ASCII standard uses 7-bit codes for most characters, with an additional bit reserved for parity, making it 8 bits in total. This allows for the representation of 256 different characters, including letters, numbers, punctuation, and control characters.

#### ASCII Code Chart

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
</tr>
<tr>
<td>1</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
</tr>
<tr>
<td>2</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
</tr>
<tr>
<td>4</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
</tr>
<tr>
<td>5</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>7</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
</tr>
</tbody>
</table>

The chart above shows the ASCII codes for the first 128 characters, which are the standard characters used in English. Each character is represented by an 8-bit code, where the first three bits determine the sextuple block, and the last five bits determine the character within that block.

- **8 rows:** 3 bits
- **4 rows:** 5 bits
- **2 rows:** 6 bits
- **1 row:** 7 bits

This representation allows for a wide range of characters to be encoded, including numbers, letters, punctuation, and control characters that are used for formatting and communication purposes.
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0 0 0</th>
<th>0 0 1</th>
<th>0 1 0</th>
<th>0 1 1</th>
<th>1 0 0</th>
<th>1 0 1</th>
<th>1 1 0</th>
<th>1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
</tr>
<tr>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
</tr>
<tr>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
</tbody>
</table>

8 rows: 3 bits
16 columns: 4 bits

ASCII Code Chart
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

### ASCII Code Chart

<table>
<thead>
<tr>
<th>Symbol</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
</tr>
<tr>
<td>A B C D E F</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
<td>DLE</td>
<td>DC1</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>! &quot; # $ % &amp; ' ( ) * + , - . /</td>
<td>: ;</td>
<td>&lt; = &gt; ?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A B C D E F G H I J K L M N O</td>
<td>P Q R S T U V W X Y Z [ \ ] ^ _</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a b c d e f g h i j k l m n o</td>
<td>p q r s t u v w x y z {</td>
<td>} ~ DEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Layout was chosen to support sorting by character code
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>ASCII Code Chart</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
</tr>
<tr>
<td>NUL</td>
</tr>
<tr>
<td>DLE</td>
</tr>
<tr>
<td>!</td>
</tr>
<tr>
<td>0 1 2</td>
</tr>
<tr>
<td>@</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>p</td>
</tr>
</tbody>
</table>

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

| 0 0 0 | 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS  | HT  | LF  | VT  | FF  | CR  | SO  | SI  |
|-------|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 0 1 | 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM  | SUB | ESC | FS  | GS  | RS  | US  |
| 0 1 0 | 2 | ID ! | " | # | $ | % | & | ( | ) | * | + | – | – | ./ |     |
| 0 1 1 | 3 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O  |
| 1 0 0 | 4 | P | Q | R | S | T | U | V | W | X | Y | Z | [  | \  | ]  | ^ | _  |
| 1 0 1 | 5 | \ | ^ | _ | | | | | | | | | | | | | |   |
| 1 1 0 | 6 | ASCII characters | | | | | | | | | | | | | | |   |
| 1 1 1 | 7 | ASCII characters | | | | | | | | | | | | | | |   |

16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission
Representing Strings: the ASCII Standard

## American Standard Code for Information Interchange

### ASCII Code Chart

<table>
<thead>
<tr>
<th>8 rows: 3 bits</th>
<th>16 columns: 4 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>NUL</td>
</tr>
<tr>
<td>0 0 1</td>
<td>SOH</td>
</tr>
<tr>
<td>0 1 0</td>
<td>ETX</td>
</tr>
<tr>
<td>0 1 1</td>
<td>EOT</td>
</tr>
<tr>
<td>0 1 2</td>
<td>ENQ</td>
</tr>
<tr>
<td>0 1 3</td>
<td>ACK</td>
</tr>
<tr>
<td>0 1 4</td>
<td>BEL</td>
</tr>
<tr>
<td>0 1 5</td>
<td>BS</td>
</tr>
<tr>
<td>0 1 6</td>
<td>HT</td>
</tr>
<tr>
<td>0 1 7</td>
<td>LF</td>
</tr>
<tr>
<td>0 1 8</td>
<td>VT</td>
</tr>
<tr>
<td>0 1 9</td>
<td>FF</td>
</tr>
<tr>
<td>0 1 A</td>
<td>CR</td>
</tr>
<tr>
<td>0 1 B</td>
<td>SO</td>
</tr>
<tr>
<td>0 1 C</td>
<td>SI</td>
</tr>
<tr>
<td>1 0 0</td>
<td>DLE</td>
</tr>
<tr>
<td>1 0 1</td>
<td>DC1</td>
</tr>
<tr>
<td>1 0 2</td>
<td>DC2</td>
</tr>
<tr>
<td>1 0 3</td>
<td>DC3</td>
</tr>
<tr>
<td>1 0 4</td>
<td>DC4</td>
</tr>
<tr>
<td>1 0 5</td>
<td>NAK</td>
</tr>
<tr>
<td>1 0 6</td>
<td>SYN</td>
</tr>
<tr>
<td>1 0 7</td>
<td>ETB</td>
</tr>
<tr>
<td>1 0 8</td>
<td>CAN</td>
</tr>
<tr>
<td>1 0 9</td>
<td>EM</td>
</tr>
<tr>
<td>1 0 A</td>
<td>SUB</td>
</tr>
<tr>
<td>1 0 B</td>
<td>ESC</td>
</tr>
<tr>
<td>1 0 C</td>
<td>FS</td>
</tr>
<tr>
<td>1 0 D</td>
<td>GS</td>
</tr>
<tr>
<td>1 0 E</td>
<td>RS</td>
</tr>
<tr>
<td>1 0 F</td>
<td>US</td>
</tr>
<tr>
<td>1 1 0</td>
<td>@</td>
</tr>
<tr>
<td>1 1 1</td>
<td>#</td>
</tr>
<tr>
<td>1 1 2</td>
<td>$</td>
</tr>
<tr>
<td>1 1 3</td>
<td>%</td>
</tr>
<tr>
<td>1 1 4</td>
<td>&amp;</td>
</tr>
<tr>
<td>1 1 5</td>
<td>(</td>
</tr>
<tr>
<td>1 1 6</td>
<td>)</td>
</tr>
<tr>
<td>1 1 7</td>
<td>*</td>
</tr>
<tr>
<td>1 1 8</td>
<td>+</td>
</tr>
<tr>
<td>1 1 9</td>
<td>,</td>
</tr>
<tr>
<td>1 1 A</td>
<td>-</td>
</tr>
<tr>
<td>1 1 B</td>
<td>.</td>
</tr>
<tr>
<td>1 1 C</td>
<td>/</td>
</tr>
<tr>
<td>1 1 D</td>
<td>0</td>
</tr>
<tr>
<td>1 1 E</td>
<td>1</td>
</tr>
<tr>
<td>1 1 F</td>
<td>2</td>
</tr>
<tr>
<td>2 0 0</td>
<td>3</td>
</tr>
<tr>
<td>2 0 1</td>
<td>4</td>
</tr>
<tr>
<td>2 0 2</td>
<td>5</td>
</tr>
<tr>
<td>2 0 3</td>
<td>6</td>
</tr>
<tr>
<td>2 0 4</td>
<td>7</td>
</tr>
<tr>
<td>2 0 5</td>
<td>8</td>
</tr>
<tr>
<td>2 0 6</td>
<td>9</td>
</tr>
<tr>
<td>2 0 7</td>
<td>:</td>
</tr>
<tr>
<td>2 0 8</td>
<td>;</td>
</tr>
<tr>
<td>2 0 9</td>
<td>&lt;</td>
</tr>
<tr>
<td>2 0 A</td>
<td>=</td>
</tr>
<tr>
<td>2 0 B</td>
<td>&gt;</td>
</tr>
<tr>
<td>2 0 C</td>
<td>?</td>
</tr>
<tr>
<td>2 1 0</td>
<td>A</td>
</tr>
<tr>
<td>2 1 1</td>
<td>B</td>
</tr>
<tr>
<td>2 1 2</td>
<td>C</td>
</tr>
<tr>
<td>2 1 3</td>
<td>D</td>
</tr>
<tr>
<td>2 1 4</td>
<td>E</td>
</tr>
<tr>
<td>2 1 5</td>
<td>F</td>
</tr>
<tr>
<td>2 1 6</td>
<td>G</td>
</tr>
<tr>
<td>2 1 7</td>
<td>H</td>
</tr>
<tr>
<td>2 1 8</td>
<td>I</td>
</tr>
<tr>
<td>2 1 9</td>
<td>J</td>
</tr>
<tr>
<td>2 1 A</td>
<td>K</td>
</tr>
<tr>
<td>2 1 B</td>
<td>L</td>
</tr>
<tr>
<td>2 1 C</td>
<td>M</td>
</tr>
<tr>
<td>2 1 D</td>
<td>N</td>
</tr>
<tr>
<td>2 1 E</td>
<td>O</td>
</tr>
<tr>
<td>2 1 F</td>
<td>P</td>
</tr>
<tr>
<td>2 2 0</td>
<td>Q</td>
</tr>
<tr>
<td>2 2 1</td>
<td>R</td>
</tr>
<tr>
<td>2 2 2</td>
<td>S</td>
</tr>
<tr>
<td>2 2 3</td>
<td>T</td>
</tr>
<tr>
<td>2 2 4</td>
<td>U</td>
</tr>
<tr>
<td>2 2 5</td>
<td>V</td>
</tr>
<tr>
<td>2 2 6</td>
<td>W</td>
</tr>
<tr>
<td>2 2 7</td>
<td>X</td>
</tr>
<tr>
<td>2 2 8</td>
<td>Y</td>
</tr>
<tr>
<td>2 2 9</td>
<td>Z</td>
</tr>
<tr>
<td>2 2 A</td>
<td>[</td>
</tr>
<tr>
<td>2 2 B</td>
<td>\</td>
</tr>
<tr>
<td>2 2 C</td>
<td>]</td>
</tr>
<tr>
<td>2 2 D</td>
<td>^</td>
</tr>
<tr>
<td>2 2 E</td>
<td>_</td>
</tr>
<tr>
<td>2 3 0</td>
<td>`</td>
</tr>
<tr>
<td>2 3 1</td>
<td>a</td>
</tr>
<tr>
<td>2 3 2</td>
<td>b</td>
</tr>
<tr>
<td>2 3 3</td>
<td>c</td>
</tr>
<tr>
<td>2 3 4</td>
<td>d</td>
</tr>
<tr>
<td>2 3 5</td>
<td>e</td>
</tr>
<tr>
<td>2 3 6</td>
<td>f</td>
</tr>
<tr>
<td>2 3 7</td>
<td>g</td>
</tr>
<tr>
<td>2 3 8</td>
<td>h</td>
</tr>
<tr>
<td>2 3 9</td>
<td>i</td>
</tr>
<tr>
<td>2 3 A</td>
<td>j</td>
</tr>
<tr>
<td>2 3 B</td>
<td>k</td>
</tr>
<tr>
<td>2 3 C</td>
<td>l</td>
</tr>
<tr>
<td>2 3 D</td>
<td>m</td>
</tr>
<tr>
<td>2 3 E</td>
<td>n</td>
</tr>
<tr>
<td>2 3 F</td>
<td>o</td>
</tr>
<tr>
<td>2 4 0</td>
<td>p</td>
</tr>
<tr>
<td>2 4 1</td>
<td>q</td>
</tr>
<tr>
<td>2 4 2</td>
<td>r</td>
</tr>
<tr>
<td>2 4 3</td>
<td>s</td>
</tr>
<tr>
<td>2 4 4</td>
<td>t</td>
</tr>
<tr>
<td>2 4 5</td>
<td>u</td>
</tr>
<tr>
<td>2 4 6</td>
<td>v</td>
</tr>
<tr>
<td>2 4 7</td>
<td>w</td>
</tr>
<tr>
<td>2 4 8</td>
<td>x</td>
</tr>
<tr>
<td>2 4 9</td>
<td>y</td>
</tr>
<tr>
<td>2 4 A</td>
<td>z</td>
</tr>
<tr>
<td>2 4 B</td>
<td>{</td>
</tr>
<tr>
<td>2 4 C</td>
<td></td>
</tr>
<tr>
<td>2 4 D</td>
<td>}</td>
</tr>
<tr>
<td>2 4 E</td>
<td>~</td>
</tr>
<tr>
<td>2 4 F</td>
<td>DEL</td>
</tr>
</tbody>
</table>

- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
</tr>
<tr>
<td>16</td>
<td>DLE</td>
</tr>
<tr>
<td>17</td>
<td>DC1</td>
</tr>
<tr>
<td>18</td>
<td>DC2</td>
</tr>
<tr>
<td>19</td>
<td>DC3</td>
</tr>
<tr>
<td>20</td>
<td>DC4</td>
</tr>
<tr>
<td>21</td>
<td>NAK</td>
</tr>
<tr>
<td>22</td>
<td>SYN</td>
</tr>
<tr>
<td>23</td>
<td>ETB</td>
</tr>
<tr>
<td>24</td>
<td>CAN</td>
</tr>
<tr>
<td>25</td>
<td>EM</td>
</tr>
<tr>
<td>26</td>
<td>SUB</td>
</tr>
<tr>
<td>27</td>
<td>ESC</td>
</tr>
<tr>
<td>28</td>
<td>FS</td>
</tr>
<tr>
<td>29</td>
<td>GS</td>
</tr>
<tr>
<td>30</td>
<td>RS</td>
</tr>
<tr>
<td>31</td>
<td>US</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>@</td>
</tr>
<tr>
<td>33</td>
<td>A</td>
</tr>
<tr>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>35</td>
<td>C</td>
</tr>
<tr>
<td>36</td>
<td>D</td>
</tr>
<tr>
<td>37</td>
<td>E</td>
</tr>
<tr>
<td>38</td>
<td>F</td>
</tr>
<tr>
<td>39</td>
<td>G</td>
</tr>
<tr>
<td>40</td>
<td>H</td>
</tr>
<tr>
<td>41</td>
<td>I</td>
</tr>
<tr>
<td>42</td>
<td>J</td>
</tr>
<tr>
<td>43</td>
<td>K</td>
</tr>
<tr>
<td>44</td>
<td>L</td>
</tr>
<tr>
<td>45</td>
<td>M</td>
</tr>
<tr>
<td>46</td>
<td>N</td>
</tr>
<tr>
<td>47</td>
<td>O</td>
</tr>
<tr>
<td>48</td>
<td>P</td>
</tr>
<tr>
<td>49</td>
<td>Q</td>
</tr>
<tr>
<td>50</td>
<td>R</td>
</tr>
<tr>
<td>51</td>
<td>S</td>
</tr>
<tr>
<td>52</td>
<td>T</td>
</tr>
<tr>
<td>53</td>
<td>U</td>
</tr>
<tr>
<td>54</td>
<td>V</td>
</tr>
<tr>
<td>55</td>
<td>W</td>
</tr>
<tr>
<td>56</td>
<td>X</td>
</tr>
<tr>
<td>57</td>
<td>Y</td>
</tr>
<tr>
<td>58</td>
<td>Z</td>
</tr>
<tr>
<td>59</td>
<td>[</td>
</tr>
<tr>
<td>60</td>
<td>\</td>
</tr>
<tr>
<td>61</td>
<td>]</td>
</tr>
<tr>
<td>62</td>
<td>^</td>
</tr>
<tr>
<td>63</td>
<td>_</td>
</tr>
</tbody>
</table>

16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission
### Representing Strings: the ASCII Standard

#### American Standard Code for Information Interchange

| 0 0 0 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS  | HT  | LF  | VT  | FF  | CR  | SO  | SI  |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 0 1 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM  | SUB | ESC | FS  | GS  | RS  | US  |
| 0 1 0 | !   | "   | #   | $   | %   | &   | '   | (   | )   | *   | +   | ,   | -   | .   | /   |
| 0 1 1 | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | :   | ;   | <   | =   | >   |
| 1 0 0 | @   | A   | B   | C   | D   | E   | F   | G   | H   | I   | J   | K   | L   | M   | N   | O   |
| 1 0 1 | P   | Q   | R   | S   | T   | U   | V   | W   | X   | Y   | Z   | [   | \   | ]   | ^   | _   |
| 1 1 0 | `   | a   | b   | c   | d   | e   | f   | g   | h   | i   | j   | k   | l   | m   | n   | o   |
| 1 1 1 | p   | q   | r   | s   | t   | u   | v   | w   | x   | y   | z   | {   | |   | }   | ~   | DEL |

| 8 rows: 3 bits |
| 16 columns: 4 bits |

- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

(Demo)
Representing Strings: the Unicode Standard
Representing Strings: the Unicode Standard

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1
- 150 scripts (organized)

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1
- 150 scripts (organized)
- Enumeration of character properties, such as case

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

• 137,994 characters in Unicode 12.1
• 150 scripts (organized)
• Enumeration of character properties, such as case
• Supports bidirectional display order
Representing Strings: the Unicode Standard

• 137,994 characters in Unicode 12.1
• 150 scripts (organized)
• Enumeration of character properties, such as case
• Supports bidirectional display order
• A canonical name for every character

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1
- 150 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

LATIN CAPITAL LETTER A
Representing Strings: the Unicode Standard

• 137,994 characters in Unicode 12.1
• 150 scripts (organized)
• Enumeration of character properties, such as case
• Supports bidirectional display order
• A canonical name for every character

Latin Capital Letter A

Die Face-6

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1
- 150 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

LATIN CAPITAL LETTER A

DIE FACE–6

EIGHTH NOTE
Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1
- 150 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

LATIN CAPITAL LETTER A

DIE FACE-6

EIGHTH NOTE
Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1
- 150 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

LATIN CAPITAL LETTER A

DIE FACE-6

EIGHTH NOTE
Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1
- 150 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

LATIN CAPITAL LETTER A

DIE FACE-6

EIGHTH NOTE

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

(Demo)
Mutation Operations
Some Objects Can Change

[Demo]
Some Objects Can Change

[Demo]

First example in the course of an object changing state
Some Objects Can Change

First example in the course of an object changing state

The same object can change in value throughout the course of computation
Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

same_person [__] → 🧄
Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

same_person ⬝ → BABY
**Some Objects Can Change**

First example in the course of an object changing state

The same object can change in value throughout the course of computation
Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

same_person \[\rightarrow\] GIRL

Unicode character name
Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

jessica
same_person

Unicode character name

GIRL
Some Objects Can Change

First example in the course of an object changing state

The same object can change in value throughout the course of computation

[Demo]

Unicode character name
Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

```
jessica ___________
same_person ___________
OLDER WOMAN
Unicode character name
```
Some Objects Can Change

First example in the course of an object changing state

The same object can change in value throughout the course of computation

```
jessica
same_person

OLDER WOMAN
```

All names that refer to the same object are affected by a mutation
Some Objects Can Change

First example in the course of an object changing state

The same object can change in value throughout the course of computation

All names that refer to the same object are affected by a mutation

Only objects of *mutable* types can change: lists & dictionaries
Some Objects Can Change

First example in the course of an object changing state

The same object can change in value throughout the course of computation

All names that refer to the same object are affected by a mutation

Only objects of mutable types can change: lists & dictionaries
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope
**Mutation Can Happen Within a Function Call**

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
```
Mutation Can HappenWithin a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
def mystery(s):
    s.pop()
    s.pop()
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
def mystery(s):
    s.pop()
    s.pop()
```

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
def mystery(s):
    or def mystery(s):
        s[2:] = []
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
>>> four = [1, 2, 3, 4]
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
def mystery(s):
    s.pop()
    s.pop()
```

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2

>>> four = [1, 2, 3, 4]
>>> len(four)
4
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```python
def mystery(s):
    s.pop()
    s.pop()
```

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery() # No arguments!
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
def mystery(s):
    s.pop()
    s.pop()

>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2

>>> another_mystery() # No arguments!
>>> len(four)
2
```
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```python
def mystery(s):
    s.pop()
    s.pop()

def another_mystery():
    four.pop()
    four.pop()
```

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery()  # No arguments!
>>> len(four)
2
```
Tuples

(Demo)
Tuples are Immutable Sequences
Tuples are Immutable Sequences

Immutable values are protected from mutation
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
```
```python
>>> ooze()
```
```python
>>> turtle
```
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)  # >>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
(1, 2, 3)
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)

>>> turtle = [1, 2, 3]
>>> ooze()
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

```python
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
[1, 2, 3]
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

```python
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

```python
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

Next lecture: ooze can change turtle's binding
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle = (1, 2, 3)
```

Next lecture: ooze can change turtle's binding

The value of an expression can change because of changes in names or objects
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
```  
Next lecture: ooze can change turtle's binding

```python
>>> turtle = [1, 2, 3]
>>> ooze()
```  
```python
>>> turtle
['Anything could be inside!']
```  
The value of an expression can change because of changes in names or objects

**Name change:**
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)

Next lecture: ooze can change turtle's binding

>>> ooze()

>>> turtle = [1, 2, 3]

['Anything could be inside!']

>>> ooze()

>>> turtle

The value of an expression can change because of changes in names or objects

```python
>>> x + x

Name change:

```python
>>> x + x

```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)  # Next lecture: ooze can change turtle's binding

Next lecture: ooze can change turtle's binding

>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

```
>>> x = 2
>>> x + x

Name change:

>>> x + x
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle = (1, 2, 3)
```

Next lecture: ooze can change turtle's binding

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
Name change:

>>> x + x
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
Name change:

>>> x = 3
>>> x + x
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
Next lecture: ooze can change turtle's binding
>>> turtle
(1, 2, 3)
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Name change:
Tuples are Immutable Sequences

Immutable values are protected from mutation

>>> turtle = (1, 2, 3)
>>> ooze()
Next lecture: ooze can change turtle's binding
>>> turtle
(1, 2, 3)

The value of an expression can change because of changes in names or objects

>>> x = 2
>>> x + x
4
Name change:

>>> x = 3
>>> x + x
6
Object mutation:
Tuples are Immutable Sequences

Immutable values are protected from mutation

>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)

Next lecture: ooze can change turtle's binding

The value of an expression can change because of changes in names or objects

>>> x = 2
>>> x + x
4

Name change:

>>> x = 3
>>> x + x
6

Object mutation:

>>> x = 2
>>> x + x
4

>>> x = 3
>>> x + x
6
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle (1, 2, 3)
Next lecture: ooze can change turtle's binding
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
Name change:
```

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
Object mutation:
```

Next lecture: ooze can change turtle's binding
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle = (1, 2, 3)
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4

Name change:

>>> x = 3
>>> x + x
6

Object mutation:

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Name change:

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 1, 2, 3, 3]
```

Next lecture: ooze can change turtle's binding
Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

The value of an expression can change because of changes in names or objects

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
Name change:

>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
Object mutation:
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)

The value of an expression can change because of changes in names or objects

Name change:

>>> x = 2
>>> x + x
4

Object mutation:

>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]

>>> x = 3
>>> x + x
6

>>> x = [1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]

An immutable sequence may still change if it contains a mutable value as an element

Next lecture: ooze can change turtle's binding

['Anything could be inside!']
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
Next lecture: ooze can change turtle's binding

>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
Name change:
>>> x = 3
>>> x + x
6
```

```python
>>> x = 1
>>> x + x
[1, 2, 1, 2]
Object mutation:
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it contains a mutable value as an element

```python
>>> s = ([1, 2], 3)
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Name change:

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
```

Object mutation:

```python
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it contains a mutable value as an element

```python
>>> s = ([1, 2], 3)
>>> s[0] = 4
```
**Tuples are Immutable Sequences**

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
Next lecture: ooze can change turtle's binding
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4

Name change:

>>> x = 3
>>> x + x
6

Object mutation:

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]

```python
>>> x = 3
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it contains a mutable value as an element

```python
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
Next lecture: ooze can change turtle's binding
>>> turtle
(1, 2, 3)
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Name change:

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

Object mutation:

An immutable sequence may still change if it contains a mutable value as an element

```python
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
>>> s = ([1, 2], 3)
```
**Tuples are Immutable Sequences**

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
```

**Name change:**

```python
>>> x = 3
>>> x + x
6
```

**Object mutation:**

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
```

An immutable sequence may still change if it contains a mutable value as an element

```python
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

Next lecture: ooze can change turtle's binding
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
Next lecture: ooze can change turtle's binding

>>>
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
Name change:
```

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
```  

Object mutation:

An immutable sequence may still change if it contains a mutable value as an element

```python
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)

The value of an expression can change because of changes in names or objects

Name change:

>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6

Object mutation:

>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]

An immutable sequence may still change if it contains a mutable value as an element

>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR

Next lecture: ooze can change turtle's binding
Mutation
Sameness and Change
Sameness and Change

As long as we never modify objects, a compound object is just the totality of its pieces
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
Sameness and Change

• As long as we never modify objects, a compound object is just the totality of its pieces
• A rational number is just its numerator and denominator
• This view is no longer valid in the presence of change
• A compound data object has an "identity" in addition to the pieces of which it is composed
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```python
>>> a = [10]
```
Sameness and Change

• As long as we never modify objects, a compound object is just the totality of its pieces
• A rational number is just its numerator and denominator
• This view is no longer valid in the presence of change
• A compound data object has an "identity" in addition to the pieces of which it is composed
• A list is still "the same" list even if we change its contents

```python
>>> a = [10]
>>> b = a
```
Sameness and Change

• As long as we never modify objects, a compound object is just the totality of its pieces
• A rational number is just its numerator and denominator
• This view is no longer valid in the presence of change
• A compound data object has an "identity" in addition to the pieces of which it is composed
• A list is still "the same" list even if we change its contents

```python
>>> a = [10]
>>> b = a
>>> a == b
True
```
Sameness and Change

• As long as we never modify objects, a compound object is just the totality of its pieces
• A rational number is just its numerator and denominator
• This view is no longer valid in the presence of change
• A compound data object has an "identity" in addition to the pieces of which it is composed
• A list is still "the same" list even if we change its contents

>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
```
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
```
**Sameness and Change**

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```
Sameness and Change

• As long as we never modify objects, a compound object is just the totality of its pieces
• A rational number is just its numerator and denominator
• This view is no longer valid in the presence of change
• A compound data object has an "identity" in addition to the pieces of which it is composed
• A list is still "the same" list even if we change its contents
• Conversely, we could have two lists that happen to have the same contents, but are different

```python
>>> a = [10]  
>>> b = a  
>>> a == b  
True  
>>> a.append(20)  
>>> a  
[10, 20]  
>>> b  
[10, 20]  
>>> a == b  
True
```
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```python
>>> a = [10]  
>>> b = a  
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```
Sameness and Change

As long as we never modify objects, a compound object is just the totality of its pieces

A rational number is just its numerator and denominator

This view is no longer valid in the presence of change

A compound data object has an "identity" in addition to the pieces of which it is composed

A list is still "the same" list even if we change its contents

Conversely, we could have two lists that happen to have the same contents, but are different

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces.
- A rational number is just its numerator and denominator.
- This view is no longer valid in the presence of change.
- A compound data object has an "identity" in addition to the pieces of which it is composed.
- A list is still "the same" list even if we change its contents.
- Conversely, we could have two lists that happen to have the same contents, but are different.

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```
Sameness and Change

• As long as we never modify objects, a compound object is just the totality of its pieces
• A rational number is just its numerator and denominator
• This view is no longer valid in the presence of change
• A compound data object has an "identity" in addition to the pieces of which it is composed
• A list is still "the same" list even if we change its contents
• Conversely, we could have two lists that happen to have the same contents, but are different

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
>>> b.append(20)
>>> b
[10, 20]
>>> a == b
True
```
Sameness and Change

• As long as we never modify objects, a compound object is just the totality of its pieces
• A rational number is just its numerator and denominator
• This view is no longer valid in the presence of change
• A compound data object has an "identity" in addition to the pieces of which it is composed
• A list is still "the same" list even if we change its contents
• Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
>>> a
[10, 20, 20]
```

```
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> a
[10]
>>> b.append(20)
>>> b
[10, 20]
>>> a == b
False
```
Identity Operators
Identity Operators

Identity

<exp0> is <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to the same object
Identity Operators

Identity

<exp0> is <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to the same object

Equality

<exp0> == <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to equal values
Identity Operators

**Identity**

<exp0> is <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to the same object

**Equality**

<exp0> == <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to equal values

Identical objects are always equal values
Identity Operators

Identity

<exp0> is <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to the same object

Equality

<exp0> == <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to equal values

Identical objects are always equal values

(Demo)
Mutable Default Arguments are Dangerous
Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call.
Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```python
def f(s=[]):
    ...  s.append(3)
    ...  return len(s)
...```

```bash
>>> def f(s=[]):
    ...  s.append(3)
    ...  return len(s)
...```
Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```python
>>> def f(s=[]):
    ...     s.append(3)
    ...     return len(s)
    ...
>>> f()
1
```
Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```python
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
>>> f()
2
```
Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```python
>>> def f(s=[]):
    ...   s.append(3)
    ...   return len(s)
    ...
>>> f()
1
>>> f()
2
>>> f()
3
```
Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```python
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
>>> f()
2
>>> f()
3
```
mutable default arguments are dangerous

A default argument value is part of a function value, not generated by a call.

```python
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
... >>> f()
1
>>> f()
2
>>> f()
3
```

Each time the function is called, `s` is bound to the same value!
Mutable Functions
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
```
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
75
```
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
75
```

Argument:
amount to withdraw
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value: remaining balance

>>> withdraw(25)
75

Argument: amount to withdraw
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Argument:
amount to withdraw

Return value:
remaining balance

>>> withdraw(25)
75

>>> withdraw(25)
50
Let's model a bank account that has a balance of $100.

Return value: remaining balance

```python
>>> withdraw(25)
75

>>> withdraw(25)
50
```

- Argument: amount to withdraw
- Second withdrawal of the same amount
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value: remaining balance

Argument: amount to withdraw

Different return value!

>>> withdraw(25) 75

>>> withdraw(25) 50

Second withdrawal of the same amount
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value: remaining balance

Argument: amount to withdraw

Different return value!

>>> withdraw(25)
75

>>> withdraw(25)
50

Second withdrawal of the same amount

>>> withdraw(60)
'Insufficient funds'
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value: remaining balance

Different return value!

>>> withdraw(25)
75

Argument: amount to withdraw

Second withdrawal of the same amount

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value: remaining balance

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

'Insufficient funds'

Where's this balance stored?
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```python
>>> withdraw = make_withdraw_list(100)

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35
```

Return value: remaining balance

Different return value!

Argument: amount to withdraw

Second withdrawal of the same amount

Where's this balance stored?
Let's model a bank account that has a balance of $100.

```python
>>> withdraw = make_withdraw_list(100)
```

- **Return value:** remaining balance
- **Different return value!**

```python
>>> withdraw(25)
75
```
- **Argument:** amount to withdraw

```python
>>> withdraw(25)
50
```
- **Second withdrawal of the same amount**

```python
>>> withdraw(60)
'Insufficient funds'
```
- Where's this balance stored?

```python
>>> withdraw(15)
35
```
Mutable Values & Persistent Local State
Mutable Values & Persistent Local State

def make_withdraw_list(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw

withdraw = make_withdraw_list(100)
withdraw(25)
Mutable Values & Persistent Local State

```python
def make_withdraw_list(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw

withdraw = make_withdraw_list(100)
withdraw(25)
```
Mutable Values & Persistent Local State

```python
def make_withdraw_list(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw

withdraw = make_withdraw_list(100)
withdraw(25)
```
Mutable Values & Persistent Local State

Global frame

func make_withdraw_list(balance) [parent=Global]

list

0

75

fn: make_withdraw_list [parent=Global]

balance

100

withdraw

Element assignment changes a list

amount

25

Return value

75

f2: withdraw [parent=f1]

def make_withdraw_list(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw

withdraw = make_withdraw_list(100)
withdraw(25)

Name bound outside of withdraw def
Mutable Values & Persistent Local State

withdraw doesn't reassign any name within the parent

Name bound outside of withdraw def

Element assignment changes a list
Mutable Values & Persistent Local State

withdraw doesn't reassign any name within the parent

It changes the contents of the b list

Name bound outside of withdraw def

Element assignment changes a list