Mutable Values
Announcements
Objects

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

8 rows: 3 bits
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 0</td>
</tr>
<tr>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0 1 1 1</td>
</tr>
<tr>
<td>1 0 0 0</td>
</tr>
<tr>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>
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>0 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 1</td>
</tr>
</tbody>
</table>

8 rows: 3 bits
16 columns: 4 bits
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 NUL SOH STX ETX EOT ENQ ACK BEL BS HT LF VT FF CR SO SI</td>
</tr>
<tr>
<td>1 DLE DC1 DC2 DC3 DC4 NAK SYN ETB CAN EM SUB ESC FS GS RS US</td>
</tr>
<tr>
<td>2 ! &quot; # $ % &amp; ( ) * + , - . /</td>
</tr>
<tr>
<td>3 0 1 2 3 4 5 6 7 8 9 : ; &lt; = &gt; ?</td>
</tr>
<tr>
<td>4 @ A B C D E F G H I J K L M N O</td>
</tr>
<tr>
<td>5 P Q R S T U V W X Y Z [ \ ] ^ ~</td>
</tr>
<tr>
<td>6 ` a b c d e f g h i j k l m n o</td>
</tr>
<tr>
<td>7 p q r s t u v w x y z {</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>Character</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>NUL</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
</tr>
<tr>
<td>SOH</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
</tr>
<tr>
<td>STX</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
</tr>
<tr>
<td>ETX</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
</tr>
<tr>
<td>EOT</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
</tr>
<tr>
<td>ENQ</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
</tr>
<tr>
<td>ACK</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
</tr>
<tr>
<td>BEL</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
</tr>
<tr>
<td>BS</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
<td>( 100 )</td>
</tr>
<tr>
<td>HT</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
<td>( 101 )</td>
</tr>
<tr>
<td>LF</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
<td>( 110 )</td>
</tr>
<tr>
<td>VT</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
<td>( 111 )</td>
</tr>
<tr>
<td>FF</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
<td>( 000 )</td>
</tr>
<tr>
<td>CR</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
<td>( 001 )</td>
</tr>
<tr>
<td>SO</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
<td>( 010 )</td>
</tr>
<tr>
<td>SI</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</td>
<td>( 011 )</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

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

### ASCII Code Chart

<table>
<thead>
<tr>
<th>0 0 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>\n</td>
<td>0 0 0</td>
<td>0 0 1</td>
<td>0 1 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>SOH</td>
<td>`</td>
<td>0 1 1</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>STX</td>
<td>&quot;</td>
<td>1 0 0</td>
<td>1 0 1</td>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>ETX</td>
<td>#</td>
<td>1 1 0</td>
<td>1 1 1</td>
<td>0 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>ETX</td>
<td>$</td>
<td>0 0 1</td>
<td>0 0 0</td>
<td>0 1 1</td>
<td>0 1 0</td>
</tr>
<tr>
<td>EOT</td>
<td>%</td>
<td>0 1 0</td>
<td>0 1 1</td>
<td>1 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>ENQ</td>
<td>&amp;</td>
<td>0 1 1</td>
<td>0 1 0</td>
<td>1 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>ACK</td>
<td>(</td>
<td>1 0 0</td>
<td>1 0 1</td>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>BEL</td>
<td>*</td>
<td>1 0 1</td>
<td>1 1 0</td>
<td>0 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>BS</td>
<td>+</td>
<td>1 1 0</td>
<td>0 0 0</td>
<td>0 1 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>HT</td>
<td>,</td>
<td>0 0 0</td>
<td>0 0 1</td>
<td>0 1 1</td>
<td>0 1 0</td>
</tr>
<tr>
<td>LF</td>
<td>-</td>
<td>0 0 1</td>
<td>0 0 0</td>
<td>0 1 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>VT</td>
<td>/</td>
<td>0 1 0</td>
<td>0 1 1</td>
<td>1 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>FF</td>
<td>0 0 0</td>
<td>0 0 1</td>
<td>0 1 0</td>
<td>0 1 1</td>
<td>1 0 0</td>
</tr>
<tr>
<td>CR</td>
<td>0 1 0</td>
<td>0 1 1</td>
<td>1 0 0</td>
<td>1 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>SO</td>
<td>1 0 0</td>
<td>1 0 1</td>
<td>1 1 0</td>
<td>1 1 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>SI</td>
<td>1 1 0</td>
<td>1 1 1</td>
<td>0 0 0</td>
<td>0 0 1</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

### Layout

- **8 rows: 3 bits**
  - Layout was chosen to support sorting by character code
  - Rows indexed 2–5 are a useful 6-bit (64 element) subset
- **16 columns: 4 bits**
  - Control characters were designed for transmission

### "Line feed" (\n)

- 6 rows:
  - 0 0 0
  - 0 0 1
  - 0 1 0
  - 0 1 1
  - 1 0 0
  - 1 0 1
  - 1 1 0
  - 1 1 1

<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>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
</tr>
<tr>
<td>CR</td>
<td>SO</td>
<td>SI</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
</tr>
<tr>
<td>SI</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
<td>LF</td>
</tr>
</tbody>
</table>

- **8 rows: 3 bits**
- **16 columns: 4 bits**
- Control characters were designed for transmission
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>0 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 1</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**

<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>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</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>SYM</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Bell&quot; (\a)</td>
<td>&quot;Line feed&quot; (\n)</td>
<td>&amp;</td>
<td>*</td>
<td>+</td>
<td>-</td>
<td>/</td>
</tr>
<tr>
<td>3</td>
<td>!</td>
<td>`</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</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>
</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>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</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>
</tr>
</tbody>
</table>

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

(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)
Representing Strings: the Unicode Standard

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

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

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

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
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

(Demo)
Mutation Operations
Some Objects Can Change

[Demo]
Some Objects Can Change

First example in the course of an object changing state
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
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

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

Demol

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

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

[Demo]

same_person __ → GIRL

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

<table>
<thead>
<tr>
<th>jessica</th>
<th>same_person</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIRL</td>
<td>Unicode character name</td>
</tr>
</tbody>
</table>
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

WOMAN
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

![Diagram showing object states](Demo)
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
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 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
```
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
def mystery(s):
    s.pop()
    s.pop()
    s[2:] = []

>>> 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

>>> 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
>>> 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

>>> another_mystery()  # No arguments!
```

```python
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
```

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

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

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

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

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

```python
>>> 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
[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
```

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()
>>> turtle
['Anything could be inside!']
```
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)
```
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

```
>>> 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

Name change:
Tuples are Immutable Sequences

Immutable values are protected from mutation

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

```python
>>> ooze()
```

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

```python
>>> ooze()
```

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

---

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

```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
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

>>> 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:

```python
>>> x = 2
>>> 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
>>> 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
Next change:
>>> x = 3
>>> x + x
```

Name change:
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

Name change:

```
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:**

**Object mutation:**
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
Name change:
>>> x = 3
>>> x + x
6
```

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

Object mutation:
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:  Object mutation:

Next lecture: ooze can change turtle's binding

```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()
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]
Object mutation:
```
Tuples are Immutable Sequences

Immutable values are protected from mutation

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

Object mutation:

```python
>>> x = [1, 2]
>>> x + x [1, 2, 1, 2]
>>> x.append(3)
>>> x + x 14
```
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:

Object mutation:

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

Immutable values are protected from mutation

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

annes comment: 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 = 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
>>> x = (1, 2, 3)
>>> x.append(3)
>>> x
[1, 2, 3, 1, 2, 3]
```

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

>>> 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:

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

Object mutation:

```python
>>> 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

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

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
```
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]
```

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

```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:**

**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
```

Next lecture: ooze can change turtle's binding

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

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

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

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

Next lecture: ooze can change turtle's binding

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

```python
>>> ooze()
```

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

```python
>>> turtle['Anything could be inside!']
['Anything could be inside!']
```

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

```python
>>> x = 2
```

```python
>>> x + x
4
```

Name change:

```python
>>> x = 3
```

```python
>>> x + x
6
```

Object mutation:

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

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

```python
>>> x.append(3)
```

```python
>>> 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)
```

```python
>>> s[0] = 4
ERROR
```

```python
>>> s[0][0] = 4
```

```python
>>> s
([1, 2], 3)
```

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

```python
>>> s[0][0] = 4
```

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

Immutable values are protected from mutation

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

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

Name change:

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

Object mutation:

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

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

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

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

>>> s = ([4, 2], 3)

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

```python
>>> 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
```
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
```
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
>>> a.append(20)
>>> a
[10, 20, 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
• 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]
```
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.append(20)
>>> a
[10, 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

\(<\text{exp0}\> \text{ is } \text{exp1}\>

evaluates to True if both \(<\text{exp0}\> \text{ and } \text{exp1}\> evaluate to the same object

Equality

\(<\text{exp0}\> == \text{exp1}\>

evaluates to True if both \(<\text{exp0}\> \text{ and } \text{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)
...```

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

>>> 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!
Lists
Lists in Environment Diagrams
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>s + t</td>
<td>[2, 3] + [5, 6]</td>
<td>[7, 9]</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>append</code></td>
<td>adds one element to a list</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **append** adds one element to a list | s.append(t)  
  t = 0       |         |
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>( s = [2, 3] )</td>
<td>( t = 0 )</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one element to a list</td>
<td>s.append(t)</td>
<td>t = 0</td>
</tr>
</tbody>
</table>

```
Global

```

```
list

```

```
list

```
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
s = [2, 3] \\
t = [5, 6]
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| append adds one element to a list | s.append(t)  
\n|                         | t = 0                       |        |
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td><code>s.append(t)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>t = 0</code></td>
<td></td>
</tr>
</tbody>
</table>
### Lists in Environment Diagrams

Assume that before each example below we execute:

- $s = [2, 3]$
- $t = [5, 6]$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append add one element to a list</td>
<td><code>s.append(t)</code></td>
<td>$s \rightarrow [2, 3, [5, 6]]$</td>
</tr>
<tr>
<td></td>
<td>$t = \emptyset$</td>
<td>$t \rightarrow \emptyset$</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one element to a list</td>
<td>s.append(t)</td>
<td>[2, 3, [5, 6]]</td>
</tr>
<tr>
<td></td>
<td>t = 0</td>
<td>t → 0</td>
</tr>
</tbody>
</table>

Global

\[ s \]
\[ t \]

list

\[ 0 \]
\[ 2 \]
\[ 3 \]

list

\[ 0 \]
\[ 5 \]
\[ 6 \]
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td><code>s.append(t)</code></td>
<td><code>s \rightarrow [2, 3, [5, 6]]</code></td>
</tr>
<tr>
<td></td>
<td><code>t = 0</code></td>
<td><code>t \rightarrow 0</code></td>
</tr>
<tr>
<td>extend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adds all</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elements in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>one list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>to another</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| append adds one element to a list | `s.append(t)`  
  `t = 0` | `s \rightarrow [2, 3, [5, 6]]`  
  `t \rightarrow 0` |
| extend adds all elements in one list to another list | `s.extend(t)`  
  `t[1] = 0` | |

![Environment Diagram](image)
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one element</td>
<td>s.append(t) t = 0</td>
<td>s → [2, 3, [5, 6]] t → 0</td>
</tr>
<tr>
<td>extend adds all elements</td>
<td>s.extend(t) t[1] = 0</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
s = [2, 3]\\
t = [5, 6]
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one element to a list</td>
<td>s.append(t) \n t = 0</td>
<td>s → [2, 3, [5, 6]] \n t → 0</td>
</tr>
<tr>
<td>extend adds all elements in one list to another list</td>
<td>s.extend(t) \n t[1] = 0</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| append adds one element to a list | \[s.append(t) \]
\[t = 0\] | \[s \rightarrow [2, 3, [5, 6]]\]
\[t \rightarrow 0\] | | |
| extend adds all elements in one list to another list | \[s.extend(t) \]
\[t[1] = 0\] | \[s \rightarrow [2, 3, 5, 6]\]
\[t \rightarrow [5, 0]\] | | |
Lists in Environment Diagrams

Assume that before each example below we execute:
\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one</td>
<td>s.append(t)</td>
<td>s → [2, 3, [5, 6]]</td>
</tr>
<tr>
<td>element to a list</td>
<td>t = 0</td>
<td>t → 0</td>
</tr>
<tr>
<td>extend adds all</td>
<td>s.extend(t)</td>
<td>s → [2, 3, 5, 6]</td>
</tr>
<tr>
<td>elements in one</td>
<td>t[1] = 0</td>
<td>t → [5, 0]</td>
</tr>
<tr>
<td>list to another list</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

**Assume that before each example below we execute:**

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td><code>s.append(t)</code> ( t = 0 )</td>
<td>( s \rightarrow [2, 3, [5, 6]] ) ( t \rightarrow 0 )</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td><code>s.extend(t)</code> ( t[1] = 0 )</td>
<td>( s \rightarrow [2, 3, 5, 6] ) ( t \rightarrow [5, 0] )</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Environment Diagram](image)
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **append** adds one element to a list | \( s.append(t) \)
  \( t = 0 \) | \( s \rightarrow [2, 3, [5, 6]] \)
  \( t \rightarrow 0 \) |
| **extend** adds all elements in one list to another list | \( s.extend(t) \)
  \( t[1] = 0 \) | \( s \rightarrow [2, 3, 5, 6] \)
  \( t \rightarrow [5, 0] \) |
| **addition & slicing** create new lists containing existing elements | \( a = s + [t] \)
  \( b = a[1:] \)
  \( a[1] = 9 \)
  \( b[1][1] = 0 \) |
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>s.append(t)</td>
<td>s → [2, 3, [5, 6]]</td>
</tr>
<tr>
<td></td>
<td>t = 0</td>
<td>t → 0</td>
</tr>
<tr>
<td>extend</td>
<td>s.extend(t)</td>
<td>s → [2, 3, 5, 6]</td>
</tr>
<tr>
<td></td>
<td>t[1] = 0</td>
<td>t → [5, 0]</td>
</tr>
<tr>
<td>addition &amp; slicing</td>
<td>a = s + [t]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = a[1:]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a[1] = 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b[1][1] = 0</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
    s &= [2, 3] \\
    t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td><code>s.append(t)</code> \hspace{1em} <code>t = 0</code></td>
<td><code>s \rightarrow [2, 3, [5, 6]]</code> \hspace{1em} <code>t \rightarrow 0</code></td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td><code>s.extend(t)</code> \hspace{1em} <code>t[1] = 0</code></td>
<td><code>s \rightarrow [2, 3, 5, 6]</code> \hspace{1em} <code>t \rightarrow [5, 0]</code></td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td><code>a = s + [t]</code> \hspace{1em} <code>b = a[1:]</code> \hspace{1em} <code>a[1] = 9</code> \hspace{1em} <code>b[1][1] = 0</code></td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
- s = [2, 3]
- t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one</td>
<td>s.append(t)</td>
<td>s → [2, 3, [5, 6]]</td>
</tr>
<tr>
<td>element to a list</td>
<td>t = 0</td>
<td>t → 0</td>
</tr>
<tr>
<td><strong>extend</strong> adds all</td>
<td>s.extend(t)</td>
<td>s → [2, 3, 5, 6]</td>
</tr>
<tr>
<td>elements in one list</td>
<td>t[1] = 0</td>
<td>t → [5, 0]</td>
</tr>
<tr>
<td>to another list</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong></td>
<td>a = s + [t]</td>
<td></td>
</tr>
<tr>
<td>create new lists</td>
<td>b = a[1:]</td>
<td></td>
</tr>
<tr>
<td>containing existing</td>
<td>a[1] = 9</td>
<td></td>
</tr>
<tr>
<td>elements</td>
<td>b[1][1] = 0</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
  s = [2, 3]
  t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one element to a list</td>
<td>s.append(t)\n t = 0</td>
<td>s → [2, 3, [5, 6]]\n t → 0</td>
</tr>
<tr>
<td>extend adds all elements in one list to another list</td>
<td>s.extend(t)\n t[1] = 0</td>
<td>s → [2, 3, 5, 6]\n t → [5, 0]</td>
</tr>
<tr>
<td>addition &amp; slicing create new lists containing existing elements</td>
<td>a = s + [t]\n b = a[1:]\n a[1] = 9\n b[1][1] = 0</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one element to a list</td>
<td>s.append(t) t = 0</td>
<td>s → [2, 3, [5, 6]] t → 0</td>
</tr>
<tr>
<td>extend adds all elements in one list to another list</td>
<td>s.extend(t) t[1] = 0</td>
<td>s → [2, 3, 5, 6] t → [5, 0]</td>
</tr>
<tr>
<td>addition &amp; slicing create new lists containing existing elements</td>
<td>a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0</td>
<td></td>
</tr>
</tbody>
</table>

Global

```
list
0 2 3
```

```
list
0 5 6
```

```
list
0 2 3 2
```

```
list
a
b
```

```
s t
```

```
0
```
Lists in Environment Diagrams

Assume that before each example below we execute:

\( s = [2, 3] \)
\( t = [5, 6] \)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **append** adds one element to a list | \( s.\text{append}(t) \)
  \( t = 0 \) | \( s \rightarrow [2, 3, [5, 6]] \)
  \( t \rightarrow 0 \) |
| **extend** adds all elements in one list to another list | \( s.\text{extend}(t) \)
  \( t[1] = 0 \) | \( s \rightarrow [2, 3, 5, 6] \)
  \( t \rightarrow [5, 0] \) |
| **addition & slicing** create new lists containing existing elements | \( a = s + [t] \)
  \( b = a[1:] \)
  \( a[1] = 9 \)
  \( b[1][1] = 0 \) | |
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **append** adds one element to a list | s.append(t) \[ \]
\[ t = 0 \] | \[ s \to [2, 3, 5, 6] \]
\[ t \to 0 \] |
| **extend** adds all elements in one list to another list | s.extend(t) \[ \]
\[ t[1] = 0 \] | \[ s \to [2, 3, 5, 6] \]
\[ t \to [5, 0] \] |
| **addition** & **slicing** create new lists containing existing elements | a = s + [t] \[ \]
\[ b = a[1:] \]
\[ a[1] = 9 \]
\[ b[1][1] = 0 \] | 3 | 1 | 2 |

\[ 0 \]
\[ 0 \]
\[ 2 \]
\[ 1 \]

Global list: 1

List 1: 1

List 2: 1

List 3: 1

List 4: 1

List 5: 1

List 6: 1

List 7: 1

List 8: 1

List 9: 1
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>( s.\text{append}(t) ) ( t = 0 )</td>
<td>( s \rightarrow [2, 3, [5, 6]] ) ( t \rightarrow 0 )</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>( s.\text{extend}(t) ) ( t[1] = 0 )</td>
<td>( s \rightarrow [2, 3, 5, 6] ) ( t \rightarrow [5, 0] )</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>( a = s + [t] ) ( b = a[1:] ) ( a[1] = 9 ) ( b[1][1] = 0 )</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td><code>s.append(t)</code> \hspace{1em} <code>t = 0</code></td>
<td><code>s \rightarrow [2, 3, [5, 6]]</code> \hspace{1em} <code>t \rightarrow 0</code></td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td><code>s.extend(t)</code> \hspace{1em} <code>t[1] = 0</code></td>
<td><code>s \rightarrow [2, 3, 5, 6]</code> \hspace{1em} <code>t \rightarrow [5, 0]</code></td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td><code>a = s + [t]</code> <code>b = a[1:]</code> \hspace{1em} <code>a[1] = 9</code> \hspace{1em} <code>b[1][1] = 0</code></td>
<td><code>s \rightarrow [2, 3]</code> <code>t \rightarrow [5, 0]</code> <code>a \rightarrow [2, 9, [5, 0]]</code> <code>b \rightarrow [3, [5, 0]]</code></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>s.append(t) \text{ t = 0}</td>
<td>s → [2, 3, [5, 6]] \text{ t → 0}</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>s.extend(t) \text{ t[1] = 0}</td>
<td>s → [2, 3, 5, 6] \text{ t → [5, 0]}</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>a = s + [t] \text{ b = a[1:] \text{ a[1] = 9 \text{ b[1][1] = 0}}}</td>
<td>s → [2, 3] \text{ t → [5, 0]} \text{ a → [2, 9, [5, 0]] \text{ b → [3, [5, 0]]}}</td>
</tr>
</tbody>
</table>

The **list** function also creates a new list containing existing elements
\[ t = \text{list(s)} \text{ s[1] = 0} \]
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong></td>
<td>s.append(t)</td>
<td>s → [2, 3, [5, 6]]</td>
</tr>
<tr>
<td></td>
<td>t = 0</td>
<td>t → 0</td>
</tr>
<tr>
<td><strong>extend</strong></td>
<td>s.extend(t)</td>
<td>s → [2, 3, 5, 6]</td>
</tr>
<tr>
<td></td>
<td>t[1] = 0</td>
<td>t → [5, 0]</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong></td>
<td>a = s + [t]</td>
<td>a → [2, 3]</td>
</tr>
<tr>
<td></td>
<td>b = a[1:]</td>
<td>b → [5, 0]</td>
</tr>
<tr>
<td></td>
<td>a[1] = 9</td>
<td>a → [2, 9, [5, 0]]</td>
</tr>
<tr>
<td></td>
<td>b[1][1] = 0</td>
<td>b → [3, [5, 0]]</td>
</tr>
<tr>
<td><strong>The list function</strong></td>
<td>t = list(s)</td>
<td>t = list(s)</td>
</tr>
<tr>
<td></td>
<td>s[1] = 0</td>
<td>s[1] = 0</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ \text{s} = [2, 3] \]
\[ \text{t} = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one</td>
<td>( \text{s}.append(\text{t}) )</td>
<td>( \text{s} \to [2, 3, [5, 6]] ) ( \text{t} \to 0 )</td>
</tr>
<tr>
<td>element to a list</td>
<td>( \text{t} = 0 )</td>
<td></td>
</tr>
<tr>
<td>extend adds all</td>
<td>( \text{s}.extend(\text{t}) )</td>
<td>( \text{s} \to [2, 3, 5, 6] ) ( \text{t} \to [5, 0] )</td>
</tr>
<tr>
<td>elements in one</td>
<td>( \text{t}[1] = 0 )</td>
<td></td>
</tr>
<tr>
<td>list to another list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addition &amp; slicing</td>
<td>( \text{a} = \text{s} + [\text{t}] )</td>
<td>( \text{s} \to [2, 3] ) ( \text{t} \to [5, 0] ) ( \text{a} \to [2, 9, [5, 0]] ) ( \text{b} \to [3, [5, 0]] )</td>
</tr>
<tr>
<td>create new lists</td>
<td>( \text{b} = \text{a}[1:] )</td>
<td></td>
</tr>
<tr>
<td>containing existing elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The list function</td>
<td>( \text{t} = \text{list}(\text{s}) )</td>
<td></td>
</tr>
<tr>
<td>also creates a new</td>
<td>( \text{s}[1] = 0 )</td>
<td></td>
</tr>
<tr>
<td>list containing existing elements</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ \text{s} = [2, 3] \]
\[ \text{t} = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>\text{s}.append(\text{t}) \text{t} = 0</td>
<td>\text{s} \rightarrow [2, 3, [5, 6]] \text{t} \rightarrow 0</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>\text{s}.extend(\text{t}) \text{t}[1] = 0</td>
<td>\text{s} \rightarrow [2, 3, 5, 6] \text{t} \rightarrow [5, 0]</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>\text{a} = \text{s} + [\text{t}] \text{b} = \text{a}[1:] \text{a}[1] = 9 \text{b}[1][1] = 0</td>
<td>\text{s} \rightarrow [2, 3] \text{t} \rightarrow [5, 0] \text{a} \rightarrow [2, 9, [5, 0]] \text{b} \rightarrow [3, [5, 0]]</td>
</tr>
<tr>
<td>The <strong>list</strong> function also creates a new list containing existing elements</td>
<td>\text{t} = \text{list}(\text{s}) \text{s}[1] = 0</td>
<td></td>
</tr>
</tbody>
</table>

![Environment Diagram](image-url)
Lists in Environment Diagrams

Assume that before each example below we execute:
\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>s.append(t) [ t = 0 ]</td>
<td>[ s \rightarrow [2, 3, [5, 6]], t \rightarrow 0 ]</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>s.extend(t) [ t[1] = 0 ]</td>
<td>[ s \rightarrow [2, 3, 5, 6], t \rightarrow [5, 0] ]</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>a = s + [t] [ b = a[1:] ] [ a[1] = 9 ] [ b[1][1] = 0 ]</td>
<td>[ s \rightarrow [2, 3], t \rightarrow [5, 0], a \rightarrow [2, 9, [5, 0]], b \rightarrow [3, [5, 0]] ]</td>
</tr>
<tr>
<td>The <strong>list</strong> function also creates a new list containing existing elements</td>
<td>t = list(s) [ s[1] = 0 ]</td>
<td>[ s \rightarrow [2, 0], t \rightarrow [2, 3] ]</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td><code>s.append(t)</code>&lt;br&gt;<code>t = 0</code></td>
<td><code>s → [2, 3, [5, 6]]</code>&lt;br&gt;<code>t → 0</code></td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td><code>s.extend(t)</code>&lt;br&gt;<code>t[1] = 0</code></td>
<td><code>s → [2, 3, 5, 6]</code>&lt;br&gt;<code>t → [5, 0]</code></td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td><code>a = s + [t]</code>&lt;br&gt;<code>b = a[1:]</code>&lt;br&gt;<code>a[1] = 9</code>&lt;br&gt;<code>b[1][1] = 0</code></td>
<td><code>s → [2, 3]</code>&lt;br&gt;<code>t → [5, 0]</code>&lt;br&gt;<code>a → [2, 9, [5, 0]]</code>&lt;br&gt;<code>b → [3, [5, 0]]</code></td>
</tr>
<tr>
<td><strong>The list</strong> function also creates a new list containing existing elements</td>
<td><code>t = list(s)</code>&lt;br&gt;<code>s[1] = 0</code></td>
<td><code>s → [2, 0]</code>&lt;br&gt;<code>t → [2, 3]</code></td>
</tr>
<tr>
<td><strong>slice assignment</strong> replaces a slice with new values</td>
<td><code>s[0:0] = t</code>&lt;br&gt;<code>s[3:] = t</code>&lt;br&gt;<code>t[1] = 0</code></td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>s.append(t) t = 0</td>
<td>s → [2, 3, [5, 6]] t → 0</td>
</tr>
<tr>
<td>extend</td>
<td>s.extend(t) t[1] = 0</td>
<td>s → [2, 3, 5, 6] t → [5, 0]</td>
</tr>
<tr>
<td>addition &amp; slicing</td>
<td>a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0</td>
<td>s → [2, 3] t → [5, 0] a → [2, 9, [5, 0]] b → [3, [5, 0]]</td>
</tr>
<tr>
<td>The list function</td>
<td>t = list(s) s[1] = 0</td>
<td>s → [2, 0] t → [2, 3]</td>
</tr>
<tr>
<td>slice assignment</td>
<td>s[0:0] = t s[3:] = t t[1] = 0</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **append** adds one element to a list | \( s.append(t) \) 
\( t = 0 \) | \( s \rightarrow [2, 3, [5, 6]] \) 
\( t \rightarrow 0 \) |
| **extend** adds all elements in one list to another list | \( s.extend(t) \) 
\( t[1] = 0 \) | \( s \rightarrow [2, 3, 5, 6] \) 
\( t \rightarrow [5, 0] \) |
| **addition & slicing** create new lists containing existing elements | \( a = s + [t] \) 
\( b = a[1:] \) 
\( a[1] = 9 \) 
\( b[1][1] = 0 \) | \( s \rightarrow [2, 3] \) 
\( t \rightarrow [5, 0] \) 
\( a \rightarrow [2, 9, [5, 0]] \) 
\( b \rightarrow [3, [5, 0]] \) |
| The **list** function also creates a new list containing existing elements | \( t = \text{list}(s) \) 
\( s[1] = 0 \) | \( s \rightarrow [2, 0] \) 
\( t \rightarrow [2, 3] \) |
| **slice assignment** replaces a slice with new values | \( s[0:0] = t \) 
\( s[3:] = t \) 
\( t[1] = 0 \) | |
### Lists in Environment Diagrams

**Assume that before each example below we execute:**

\[
s = [2, 3] \\
t = [5, 6]
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>( s).append(t) t = 0</td>
<td>( s \rightarrow [2, 3, [5, 6]] ) t ( \rightarrow 0 )</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>( s).extend(t) t[1] = 0</td>
<td>( s \rightarrow [2, 3, 5, 6] ) t ( \rightarrow [5, 0] )</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>( a = s + [t] ) ( b = a[1:] ) a[1] = 9 b[1][1] = 0</td>
<td>( s \rightarrow [2, 3] ) t ( \rightarrow [5, 0] ) ( a \rightarrow [2, 9, [5, 0]] ) b ( \rightarrow [3, [5, 0]] )</td>
</tr>
<tr>
<td>The list function also creates a new list containing existing elements</td>
<td>( t = ) list(s) s[1] = 0</td>
<td>( s \rightarrow [2, 0] ) t ( \rightarrow [2, 3] )</td>
</tr>
<tr>
<td><strong>slice assignment</strong> replaces a slice with new values</td>
<td>( s[0:] = t ) ( s[3:] = t ) t[1] = 0</td>
<td></td>
</tr>
</tbody>
</table>

---

![Environment Diagram](image-url)
Lists in Environment Diagrams

Assume that before each example below we execute:

- \( s = [2, 3] \)
- \( t = [5, 6] \)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>( s.append(t) ) ( t = 0 )</td>
<td>( s \rightarrow [2, 3, [5, 6]] ) ( t \rightarrow 0 )</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>( s.extend(t) ) ( t[1] = 0 )</td>
<td>( s \rightarrow [2, 3, 5, 6] ) ( t \rightarrow [5, 0] )</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>( a = s + [t] ) ( b = a[1:] ) ( a[1] = 9 ) ( b[1][1] = 0 )</td>
<td>( s \rightarrow [2, 3] ) ( t \rightarrow [5, 0] ) ( a \rightarrow [2, 9, [5, 0]] ) ( b \rightarrow [3, [5, 0]] )</td>
</tr>
<tr>
<td>The list function also creates a new list containing existing elements</td>
<td>( t = \text{list}(s) ) ( s[1] = 0 )</td>
<td>( s \rightarrow [2, 0] ) ( t \rightarrow [2, 3] )</td>
</tr>
<tr>
<td><strong>slice assignment</strong> replaces a slice with new values</td>
<td>( s[0:0] = t ) ( s[3:] = t ) ( t[1] = 0 )</td>
<td>( s \rightarrow [5, 6, 2, 5, 6] ) ( t \rightarrow [5, 0] )</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

```python
s = [2, 3]
t = [5, 6]
```
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
    s &= [2, 3] \\
    t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>pop</td>
<td>removes &amp; returns the last element</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>pop removes &amp; returns the last element</td>
<td>t = s.pop()</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| pop removes & returns the last element | t = s.pop() | s → [2]
|                    |          | t → 3     |
Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **pop** removes & returns the last element | \[ t = s.pop() \] | \[ s \rightarrow [2] \]
|                         |               | \[ t \rightarrow 3 \] |
| **remove** removes the first element equal to the argument | | |
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **pop** removes & returns the last element | t = s.pop()    | s → [2]  
t → 3   |
| **remove** removes the first element equal to the argument | t.extend(t)   | t.remove(5) |
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| pop removes & returns the last element | \( t = s.pop() \) | \( s \rightarrow [2] \)
|                   |            | \( t \rightarrow 3 \) |
| remove removes the first element equal to the argument | \( t.extend(t) \)
|                   | \( t.remove(5) \) | \( s \rightarrow [2, 3] \)
|                   |            | \( t \rightarrow [6, 5, 6] \) |
**Lists in Environment Diagrams**

Assume that before each example below we execute:

\[
\begin{align*}
    s &= [2, 3] \\
    t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pop</strong> removes &amp; returns the last element</td>
<td>( t = s.pop() )</td>
<td>( s \rightarrow [2] ) ( t \rightarrow 3 )</td>
</tr>
</tbody>
</table>
| **remove** removes the first element equal to the argument | \( t.extend(t) \)
\( t.remove(5) \) | \( s \rightarrow [2, 3] \) \( t \rightarrow [6, 5, 6] \) |

**slice assignment** can remove elements from a list by assigning `[]` to a slice.
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **pop** removes & returns the last element | t = s.pop() | s → [2] 
t → 3 |
| **remove** removes the first element equal to the argument | t.extend(t)
  t.remove(5) | s → [2, 3]
t → [6, 5, 6] |
| **slice assignment** can remove elements from a list by assigning [] to a slice. | s[:1] = []
  t[0:2] = [] |        |
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pop</strong> removes &amp; returns the last element</td>
<td>( t = s.pop() )</td>
<td>( s \rightarrow [2] ) ( t \rightarrow 3 )</td>
</tr>
<tr>
<td><strong>remove</strong> removes the first element equal to the argument</td>
<td>( t.extend(t) ) ( t.remove(5) )</td>
<td>( s \rightarrow [2, 3] ) ( t \rightarrow [6, 5, 6] )</td>
</tr>
<tr>
<td><strong>slice assignment</strong> can remove elements from a list by assigning ([] ) to a slice.</td>
<td>( s[1:] = [] ) ( t[0:2] = [] )</td>
<td>( s \rightarrow [3] ) ( t \rightarrow [] )</td>
</tr>
</tbody>
</table>
Lists in Lists in Lists in Environment Diagrams

```python
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```

```python
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```
Lists in Lists in Lists in Environment Diagrams

\[
t = [1, 2, 3] \\
t[1:3] = [t] \\
t.extend(t)
\]

\[
t = [[1, 2], [3, 4]] \\
t[0].append(t[1:2])
\]
Lists in Lists in Lists in Environment Diagrams

\[
t = [1, 2, 3]
\]
\[
t[1:3] = [t]
\]
\[
t.extend(t)
\]

[t] evaluates to:

\[
t = [[1, 2], [3, 4]]
\]
\[
t[0].append(t[1:2])
\]
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

[t] evaluates to:

```
[1, [2, [3, 4]]]
```

```
[1, [2, [3, 4]]]
```
```
Lists in Lists in Lists in Environment Diagrams

\[ t = [1, 2, 3] \]
\[ t[1:3] = [t] \]
\[ t.extend(t) \]

\[ t = [[1, 2], [3, 4]] \]
\[ t[0].append(t[1:2]) \]
Lists in Lists in Lists in Environment Diagrams

t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

```python
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```

![Environment Diagram](image-url)
Lists in Lists in Lists in Environment Diagrams

```python
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```

---

```python
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```
Lists in Lists in Lists in Environment Diagrams

\[ t = [1, 2, 3] \]
\[ t[1:3] = [t] \]
\[ t.extend(t) \]

\[ t = \begin{bmatrix} 1, 2 \end{bmatrix}, \begin{bmatrix} 3, 4 \end{bmatrix} \]
\[ t[0].append(t[1:2]) \]
Lists in Lists in Lists in Environment Diagrams

\[ t = [1, 2, 3] \]
\[ t[1:3] = [t] \]
\[ t.extend(t) \]

\[
\begin{array}{c}
\text{Global} \\
t\leftarrow 1_1 \ 2_2 \ 3_3 \\
\end{array}
\]

\[ 1_0 \ 2_1 \ 3_2 \ 1_3 \ 2_1 \ 3_3 \]

\[ [1, [...], 1, [...]] \]

\[ t = [[1, 2], [3, 4]] \]
\[ t[0].append(t[1:2]) \]

\[
\begin{array}{c}
\text{Global} \\
t\leftarrow 0_0 \ 1_1 \\
\end{array}
\]

\[
\begin{array}{c}
\text{list} \\
0_0 \ 1_1 \\
\end{array}
\]

\[
\begin{array}{c}
\text{list} \\
3_0 \ 4_1 \\
\end{array}
\]

\[
\begin{array}{c}
\text{list} \\
1_0 \ 1_1 \ 2_2 \\
\end{array}
\]
Lists in Lists in Lists in Environment Diagrams

\[ t = [1, 2, 3] \]
\[ t[1:3] = [t] \]
\[ t.extend(t) \]

\[ [1, [...], 1, [...]] \]

\[ t = [[1, 2], [3, 4]] \]
\[ t[0].append(t[1:2]) \]
Lists in Lists in Lists in Environment Diagrams

t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

[1, [...], 1, [...]]
Lists in Lists in Lists in Environment Diagrams

```python
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```

```
[1, [...], 1, [...]]
```

```python
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```

```
[[1, 2, [[3, 4]]], [3, 4]]
```