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</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>NUL</td>
</tr>
<tr>
<td>DLE</td>
</tr>
<tr>
<td>!</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>@</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>`</td>
</tr>
<tr>
<td>p</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</td>
</tr>
<tr>
<td>NUL</td>
</tr>
<tr>
<td>DLE</td>
</tr>
</tbody>
</table>

8 rows: 3 bits

| 2 | ! | " | # | $ | % | & | ( | ) | * | + | , | - | . | / |
| 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 6 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7 | p | q | r | s | t | u | v | w | x | y | z | { | | | } | ~ | DEL |
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</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>NUL</td>
</tr>
<tr>
<td>DLE</td>
</tr>
<tr>
<td>!</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>@</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>`</td>
</tr>
<tr>
<td>p</td>
</tr>
<tr>
<td>16 columns: 4 bits</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</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>NUL</td>
</tr>
<tr>
<td>DLE</td>
</tr>
</tbody>
</table>

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

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

American Standard Code for Information Interchange

| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | A  | B  | C  | D  | E  | F  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NUL| SOH| STX| ETX| EOT| ENQ| ACK| BEL| BS | HT | LF | VT | FF | CR | SO | SI |
| DLE| DC1| DC2| DC3| DC4| NAK| SYN| ETB| CAN| EM | SUB| ESC| FS | GS | RS | US |
| !  | "  | #  | $  | %  | &  | '  | (  | )  | *  | +  | ,  | -  | .  | /  |    |
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | :) | ;  | <= | >  | ?  |    |
| @  | 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|

- 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</td>
</tr>
<tr>
<td>NUL</td>
</tr>
<tr>
<td>DLE</td>
</tr>
<tr>
<td>!</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>@</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>p</td>
</tr>
</tbody>
</table>

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
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</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>NUL</td>
</tr>
<tr>
<td>DLE</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

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

"Line feed" (\n)
### 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>

#### Layout
- **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**

#### Characters
- "Bell" (\a)
- "Line feed" (\n)

**16 columns: 4 bits**

- **8 rows: 3 bits**

The ASCII code chart shows the layout of the standard, with characters arranged in rows and columns, supporting various functionalities and purposes.
### Representing Strings: the ASCII Standard

#### American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>ASCII Code Chart</th>
</tr>
</thead>
<tbody>
<tr>
<td></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

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

• **109,000 characters**

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

- 109,000 characters
- 93 scripts (organized)

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

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 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

- 109,000 characters
- 93 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

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

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

• 109,000 characters
• 93 scripts (organized)
• Enumeration of character properties, such as case
• Supports bidirectional display order
• A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE
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

[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

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

[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

[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

```
jessica  ⟷
same_person  ⟷
```

Unicode character name

Woman emoji
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

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
Limitations on Dictionaries
Limitations on Dictionaries

Dictionaries are unordered collections of key-value pairs
Limitations on Dictionaries

Dictionaries are unordered collections of key-value pairs

Dictionary keys do have two restrictions:
Limitations on Dictionaries

Dictionaries are unordered collections of key-value pairs

Dictionary keys do have two restrictions:

• A key of a dictionary cannot be a list or a dictionary (or any mutable type)
Limitations on Dictionaries

Dictionaries are *unordered* collections of key–value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary *cannot be* a list or a dictionary (or any *mutable type*)

- Two *keys cannot be equal*; There can be at most one value for a given key
Limitations on Dictionaries

Dictionaries are unordered collections of key-value pairs.

Dictionary keys do have two restrictions:

- A key of a dictionary cannot be a list or a dictionary (or any mutable type)
- Two keys cannot be equal; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries.
Limitations on Dictionaries

Dictionaries are unordered collections of key-value pairs.

Dictionary keys do have two restrictions:

• A key of a dictionary cannot be a list or a dictionary (or any mutable type).

• Two keys cannot be equal; There can be at most one value for a given key.

This first restriction is tied to Python's underlying implementation of dictionaries.

The second restriction is part of the dictionary abstraction.
Limitations on Dictionaries

Dictionaries are unordered collections of key-value pairs

Dictionary keys do have two restrictions:

• A key of a dictionary cannot be a list or a dictionary (or any mutable type)

• Two keys cannot be equal; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

If you want to associate multiple values with a key, store them all in a sequence value: {'odds': [1, 3, 5], 'evens': [2, 4, 6]}
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()
```

Interactive Diagram
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
>>> len(four)
2
def mystery(s):
    s.pop()
    s.pop()
```

```python
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]
```

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

Interactive Diagram
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()
    or
def mystery(s):
    s[2:] = []
```

```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)
>>> len(four)
2
```

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

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

**Interactive Diagram**
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
>>> another_mystery() # No arguments!
>>> len(four)
2
```

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

# or

def mystery(s):
    s[2:] = []
```

Interactive Diagram
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]
>>> def mystery(s):
     s.pop()
     s.pop()

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

Interactive Diagram
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]
```

```python
>>> 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()
```
**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)

>>> 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)

Next lecture: ooze can change turtle's binding

>>> 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)
>>> 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()
>>> 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

Name change:
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!']
```

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

```python
>>> x + x
Name change:
>>> x + x
```
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

Name change:

>>> x + x

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

>>> 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
4

Name change:
```
```
>>> x = 3
>>> x + x
```
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!']
```

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

```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:  
Object mutation:
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

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

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

Object mutation:

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

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

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

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

Object mutation:
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)
```

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
```

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

**Object mutation:**

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

```python
>>> x + x
```
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)
```

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:  
```
```
>>> 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

```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]
```

An immutable sequence may still change if it contains a mutable value as an element
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
(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)
```
**Tuples are Immutable Sequences**

Immutable values are protected from mutation

```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 = 2
```

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

4

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

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

6

Name change:

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
```
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]
```

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
>>> 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

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

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

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

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

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

Next lecture: ooze can change turtle's binding
**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)  # 'Anything could be inside!'
```

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:  

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

>>> 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

>>> 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 == 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)
>>> a == b
True
>>> 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 == b
True
>>> 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
• 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 == b
True
>>> 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
• Conversely, we could have two lists that happen to have the same contents, but are different

```python
>>> a = [10]  >>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> 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.
- 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 == b
True
>>> 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.
- Conversely, we could have two lists that happen to have the same contents, but are different.

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

<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)
...```
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
```

Interactive Diagram
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!