Mini-Lecture 18

Classes
Recall: Objects as Data in Folders

• An object is like a **manila folder**
• It contains other variables
  - Variables are called **attributes**
  - Can change values of an attribute (with assignment statements)
• It has a “tab” that identifies it
  - Unique number assigned by Python
  - Fixed for lifetime of the object
Recall: Classes are Types for Objects

- Values must have a type
  - An object is a **value**
  - Object type is a **class**

- Classes are any type not already built-into Python

<table>
<thead>
<tr>
<th>id2</th>
<th>x</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>5.0</td>
</tr>
</tbody>
</table>

**Types**
- int
- float
- bool
- str
- list
- dict

**Classes**
- Point3
- RGB
- CMYK
- HSV
Classes Have Folders Too

Object Folders

- Separate for each *instance*

Class Folders

- Data common to all instances

<table>
<thead>
<tr>
<th>id2</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2.0</td>
</tr>
<tr>
<td>y</td>
<td>3.0</td>
</tr>
<tr>
<td>z</td>
<td>5.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id3</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5.0</td>
</tr>
<tr>
<td>y</td>
<td>7.2</td>
</tr>
<tr>
<td>z</td>
<td>-0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>????</td>
</tr>
</tbody>
</table>
Name Resolution for Objects

• \(\text{\{object\}.\{name\}\}}\) means
  - Go the folder for \textit{object}
  - Find attribute/method \textit{name}
  - If missing, check \textbf{class folder}
  - If not in either, raise error

• What is in the class folder?
  - Data common to \textbf{all} objects
  - First must understand the \textit{class definition}
The Class Definition

```
class <class-name>(object):
    """Class specification"""
    <function definitions>
    <assignment statements>
    <any other statements also allowed>
```

Example

```
class Example(object):
    """The simplest possible class."""
    pass
```
The Class Definition

keyword **class**
Beginning of a class definition

**class** `<class-name>`(object):

"""Class specification""

<function definitions>
<assignment statements>
<any other statements also allowed>

Goes inside a module, just like a function definition.

Do not forget the colon!

more on this later

…but not often used

Example

```python
class Example(object):
    """The simplest possible class.""
    pass
```

Python creates after reading the class definition
Recall: Constructors

• Function to create new instances
  ▪ Function name == class name
  ▪ Created for you automatically

• Calling the constructor:
  ▪ Makes a new object folder
  ▪ Initializes attributes
  ▪ Returns the id of the folder

• By default, takes no arguments
  ▪ e = Example()
Instances and Attributes

• Assignments add object attributes
  - `<object>..<att> = <expression>`
  - **Example**: `e.b = 42`

• Assignments can add class attributes
  - `<class>..<att> = <expression>`
  - **Example**: `Example.a = 29`

• Objects can access class attributes
  - **Example**: `print(e.a)`
  - But assigning it creates object attribute
  - **Example**: `e.a = 10`

• **Rule**: check object first, then class
Instances and Attributes

• Assignments add object attributes
  ▪ \(<object> \cdot \langle \text{att}\rangle = \langle \text{expression}\rangle\>
  ▪ **Example**: \(e.b = 42\)

• Assignments can add class attributes
  ▪ \(<\text{class}> \cdot \langle \text{att}\rangle = \langle \text{expression}\rangle\>
  ▪ **Example**: \(\text{Example}.a = 29\)

• Objects can access class attributes
  ▪ **Example**: \(\text{print}(e.a)\)
  ▪ But assigning it creates object attribute
  ▪ **Example**: \(e.a = 10\)

• **Rule**: check object first, then class
Instances and Attributes

- Assignments add object attributes
  - `<object>.<att> = <expression>`
  - **Example**: `e.b = 42`

- Assignments can add class attributes
  - `<class>.<att> = <expression>`
  - **Example**: `Example.a = 29`

- Objects can access class attributes
  - **Example**: `print(e.a)`
  - But assigning it creates object attribute
  - **Example**: `e.a = 10`

- **Rule**: check object first, then class
**Recall: Objects can have Methods**

- **Method**: function tied to object
  - Function call: `<function-name>(<arguments>)`
  - Method call: `<object-variable>.<function-call>`
- **Example**: `p.distance(q)`
  - Both `p` and `q` act as arguments
  - Very much like `distance(p, q)`
- For most Python objects
  - **Attributes** are in `object` folder
  - **Methods** are in `class` folder

<table>
<thead>
<tr>
<th></th>
<th>id3</th>
<th>id4</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>id3</td>
<td>id4</td>
</tr>
<tr>
<td>x</td>
<td>5.0</td>
<td>7.4</td>
</tr>
<tr>
<td>y</td>
<td>2.0</td>
<td>0.0</td>
</tr>
<tr>
<td>z</td>
<td>3.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Method Definitions

- Looks like a function `def`
  - But indented `inside` class
  - The first parameter is always called `self`
- In a method call:
  - Parentheses have one less argument than parameters
  - The object in front is passed to parameter `self`
- Example: `a.distance(b)`

```python
class Point3(object):
    
    ""
    Instances are points in 3d space
    ""

def distance(self, q):
    """Returns: dist from self to q
    Precondition: q a Point3"
    sqrdst = ((self.x-q.x)**2 +
              (self.y-q.y)**2 +
              (self.z-q.z)**2)
    return math.sqrt(sqrdst)
```
Methods Calls

- **Example:** `a.distance(b)`

```
class Point3(object):
    """
    Instances are points in 3d space
    """

def distance(self, q):
    """Returns: dist from self to q
    Precondition: q a Point3"
    sqrdst = ((self.x-q.x)**2 +
              (self.y-q.y)**2 +
              (self.z-q.z)**2)
    return math.sqrt(sqrdst)
```

10/2/17
• **Example:** `a.distance(b)`

```python
class Point3(object):
    ""
    Instances are points in 3d space
    ""
    def distance(self,q):
        """Returns: dist from self to q
        Precondition: q a Point3"
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

10/2/17