Lecture 22:  
Subclasses & Inheritance  
(Chapter 18)  

CS 1110  
Introduction to Computing Using Python

Topics

• Why define subclasses?
  * Understand the resulting hierarchy  
  * Design considerations  

• How to define a subclass
  * Initializer  
  * New methods  
  * Write modified versions of inherited methods  
  * Access parent’s version using super()

Goal: Make a drawing app

Rectangles, Stars, Circles, and Triangles have a lot in common, but they are also different in very fundamental ways....

Sharing Work

Problem: Redundant code.
(Any time you copy-and-paste code, you are likely doing something wrong.)

Solution: Create a parent class with shared code
  * Then, create subclasses of the parent class  
  * A subclass deals with specific details different from the parent class

Defining a Subclass

Announcements

• No new lab exercises this week. Lab sections cancelled but there’ll be extra office hours. **Good opportunity to go over A4 if you have any questions.** (Hours are listed in the office hr calendar):
  * Tues 1:15-2:30pm (Jonathan C.)  
  * Wedn 10:10-11am (Priya M.)  

• Prelim 2: we expect feedback to be available on Monday  
• Assignment 5: expected release tonight (Tues)
Extending Classes

```python
class <name>(<superclass>):
    """Class specification""
    <class variables>
    <initializer>
    <methods>
```

Class to extend (may need module name: <modulename>.<superclass>)

So far, classes have implicitly extended `object`

Object Attributes can be Inherited

```python
class Shape:
    """A shape @ location x,y""
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Circle(Shape):
    """Instance is a Circle @ x,y with size radius""
    def __init__(self, x, y, radius):
        super().__init__(x, y)
        self.radius = radius
```

Can override methods; can access parent’s version

```python
class Shape:
    """Instance is shape @ x,y""
    def __init__(self, x, y, radius):
        super().__init__(x, y)
        self.radius = radius

class Circle(Shape):
    """Instance is a Circle @ x,y with radius""
    def __init__(self, x, y, radius):
        super().__init__(x, y)
        self.radius = radius
```

Understanding Method Overriding

```python
c1 = Circle(1, 2, 4.0)
print(str(c1))
```

Object and the Subclass Hierarchy

Subclassing creates a hierarchy of classes
- Each class has its own super class or parent
- Until `object` at the "top"
- `object` has many features
  - Default operators: `__init__, __str__, __eq__`
  - Which of these need to be replaced?

```python
Example
object

Super class
Super super class

Built-in class
Circle
Shape
Rectangle
Super class

```

Example

```python
c1 = Circle(1, 2, 4.0)
```

Object Attributes can be Inherited

```python
class Shape:
    """A shape @ location x,y""
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Circle(Shape):
    """Instance is a Circle @ x,y with size radius""
    def __init__(self, x, y, radius):
        super().__init__(x, y)
        self.radius = radius
```

Understanding Method Overriding

```python
c1 = Circle(1, 2, 4.0)
print(str(c1))
```
Name Resolution Revisited

- To look up attribute/method name
  1. Look first in instance (object folder)
  2. Then look in the class (folder)
- Subclasses add two more rules:
  3. Look in the superclass
  4. Repeat 3. until reach \textit{object}

Often called the \textit{Bottom-Up Rule}

Q1: Name Resolution and Inheritance

<table>
<thead>
<tr>
<th>class A:</th>
<th>class B(A):</th>
</tr>
</thead>
<tbody>
<tr>
<td>def f(self):</td>
<td>def g(self):</td>
</tr>
<tr>
<td>\hline</td>
<td>\hline</td>
</tr>
<tr>
<td>\quad \textbf{return self.g()}</td>
<td>\quad \textbf{return 10}</td>
</tr>
<tr>
<td>def g(self):</td>
<td>def h(self):</td>
</tr>
<tr>
<td>\hline</td>
<td>\hline</td>
</tr>
<tr>
<td>\quad \textbf{return 10}</td>
<td>\quad \textbf{return 14}</td>
</tr>
<tr>
<td>\quad \textbf{return 18}</td>
<td>\quad \textbf{return 18}</td>
</tr>
</tbody>
</table>

What is value of \texttt{a.f()}?

- Execute the following:  
  \begin{itemize}
  \item \texttt{a = A()}
  \item \texttt{b = B()}
  \end{itemize}

- What is value of \texttt{b.f()}?

Q2: Name Resolution and Inheritance

<table>
<thead>
<tr>
<th>class A:</th>
<th>class B(A):</th>
</tr>
</thead>
<tbody>
<tr>
<td>def f(self):</td>
<td>def g(self):</td>
</tr>
<tr>
<td>\hline</td>
<td>\hline</td>
</tr>
<tr>
<td>\quad \textbf{return self.g()}</td>
<td>\quad \textbf{return 14}</td>
</tr>
<tr>
<td>def g(self):</td>
<td>def h(self):</td>
</tr>
<tr>
<td>\hline</td>
<td>\hline</td>
</tr>
<tr>
<td>\quad \textbf{return 14}</td>
<td>\quad \textbf{return 18}</td>
</tr>
</tbody>
</table>

A turtle holds a pen and can draw as it walks! Follows simples commands:

- \texttt{setx, sety} – set start coordinate
- \texttt{pendown, penup} – control whether to draw when moving
- \texttt{forward}
- \texttt{turn}

Part of the turtle module in Python:  
(docs.python.org/3.7/library/turtle.html)

- \texttt{setx, sety} – set start coordinate
- \texttt{pendown, penup} – control whether to draw when moving
- \texttt{forward}
- \texttt{turn}

\textbf{Just a demo! You do not need to do anything with Turtle Graphics}

Who draws what?

<table>
<thead>
<tr>
<th>class \texttt{Shape}:</th>
<th>class \texttt{Circle}(Shape):</th>
</tr>
</thead>
<tbody>
<tr>
<td>exttt{def draw(self):} \begin{itemize}</td>
<td>exttt{def draw(self):} \begin{itemize}</td>
</tr>
<tr>
<td>\hline</td>
<td>\hline</td>
</tr>
<tr>
<td>\quad \texttt{turtle.penup()}</td>
<td>\quad \texttt{super().draw()}</td>
</tr>
<tr>
<td>\quad \texttt{turtle.setx(x)}</td>
<td>\quad \texttt{turtle.circle(self.radius)}</td>
</tr>
<tr>
<td>\quad \texttt{turtle.sety(y)}</td>
<td>\quad \texttt{turtle.circle(self.radius)}</td>
</tr>
<tr>
<td>\quad \texttt{turtle.pendown()}</td>
<td>\quad \texttt{turtle.circle(self.radius)}</td>
</tr>
</tbody>
</table>

Note: need to import the \texttt{turtle} module which allows us to move a pen on a 2D grid and draw shapes.

Job for Shape:
No matter the shape, we want to pick up the pen, move to the location of the shape, put the pen down.

Job for subclasses:
But only the shape subclasses know how to do the actual drawing.

See \texttt{shapes.py, drawShapes.py}