Lecture 19:  
More on Subclassing  
(Chapter 18)  
CS 1110  
Introduction to Computing Using Python

Announcements

• Prelim 2 will be returned mid/late next week
• Don't Panic!
  ▪ Final is 30% so you can make up for any mis-steps

Topics

Continuation from last lecture
• Design considerations for overriding methods
• Name resolution for attributes and methods
• Different kinds of comparisons on objects

Goal: Make a drawing app

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

Recall: our Class Hierarchy

Recall: overriding & calling __init__

Subtle: super() calls the superclass' __init__ method
super().super() ← not a thing
Demo using Turtle Graphics

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

• setx, sety – set start coordinate
• pendown, penup – control whether to draw when moving
• forward
• turn

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

• You don’t need to know it
• Just a demo to explain design choices of draw() in our classes Shape, Circle, Rectangle, Square

Understanding Method Overriding

• Subclass inherits methods of parent
• Subclass definitions override those of parent

c1 = Circle(1,2,4.0)
c1.draw()

• Which draw() do we use?
  • Start at bottom class folder
  • Find first method with name
  • Use that definition

Who draws what?

class Shape:
    """Moves pen to correct location""
    def draw(self):
        turtle.penup()
        turtle.setx(self.x)
        turtle.sety(self.y)
        turtle.pendown()

class Circle(Shape):
    """Draws Circle""
    def draw(self):
        super().draw()
        turtle.circle(self.radius)

Part of the turtle module which allows us to move a pen on a 2D grid and draw shapes.

No matter the shape, we want to pick up the pen, move to the location of the shape, put the pen down. But only the shape subclasses know how to do the actual drawing.

Just a demo! You do not need to do anything with Turtle Graphics

Q1: Name Resolution and Inheritance

class A:
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18

Execute the following:
>>> a = A()
>>> b = B()

What is value of a.f()?
A: 10
B: 14
C: 5
D: ERROR
E: I don’t know

Class Variables can also be Inherited

class Shape: # inherits from object by default
    """Instance is shape @ x,y""
    NUM_SHAPES = 0

class Circle(Shape):
    """Instance is a Circle @ x,y with radius""
    NUM_CIRCLES = 0

Q1: Name Resolution and Inheritance

class A:
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18

Execute the following:
>>> a = A()
>>> b = B()

What is value of b.f()?
A: 10
B: 14
C: 5
D: ERROR
E: I don’t know

Class Variables can also be Inherited

class Shape: # inherits from object by default
    """Instance is shape @ x,y""
    NUM_SHAPES = 0

class Circle(Shape):
    """Instance is a Circle @ x,y with radius""
    NUM_CIRCLES = 0

Q2: Name Resolution and Inheritance

class A:
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18

Execute the following:
>>> a = A()
>>> b = B()

What is value of b.f()?
A: 10
B: 14
C: 5
D: ERROR
E: I don’t know

Class Variables can also be Inherited

class Shape: # inherits from object by default
    """Instance is shape @ x,y""
    NUM_SHAPES = 0

class Circle(Shape):
    """Instance is a Circle @ x,y with radius""
    NUM_CIRCLES = 0

Q2: Name Resolution and Inheritance

class A:
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18

Execute the following:
>>> a = A()
>>> b = B()

What is value of b.f()?
A: 10
B: 14
C: 5
D: ERROR
E: I don’t know

Class Variables can also be Inherited

class Shape: # inherits from object by default
    """Instance is shape @ x,y""
    NUM_SHAPES = 0

class Circle(Shape):
    """Instance is a Circle @ x,y with radius""
    NUM_CIRCLES = 0

Q2: Name Resolution and Inheritance

class A:
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18

Execute the following:
>>> a = A()
>>> b = B()

What is value of b.f()?
A: 10
B: 14
C: 5
D: ERROR
E: I don’t know

Class Variables can also be Inherited

class Shape: # inherits from object by default
    """Instance is shape @ x,y""
    NUM_SHAPES = 0

class Circle(Shape):
    """Instance is a Circle @ x,y with radius""
    NUM_CIRCLES = 0
Q3: Name Resolution and Inheritance

```python
class A:
    x = 3 # Class Variable
    y = 5 # Class Variable

def f(self):
    return self.g()

def g(self):
    return 10

class B(A):
    y = 4 # Class Variable
    z = 42 # Class Variable

def g(self):
    return 14
def h(self):
    return 18
```

- Execute the following:
  ```python
  >>> a = A()
  >>> b = B()
  ```
- What is value of `b.x`?
  
<table>
<thead>
<tr>
<th>Choice</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>42</td>
</tr>
<tr>
<td>D</td>
<td>ERROR</td>
</tr>
<tr>
<td>E</td>
<td>I don't know</td>
</tr>
</tbody>
</table>

Inheritance-related terminology

- **eq vs is**
- **isinstance**

`==` compares equality

`is` compares identity

```python

c1 = Circle(1, 1, 25)
c2 = Circle(1, 1, 25)
c3 = c2

c1 == c2  # True

c1 is c2  # False

c2 == c3  # True

c2 is c3  # False
```

The `isinstance` Function

- `isinstance(<obj>, <class>)`
  - True if `<obj>`'s class is same as or a subclass of `<class>`
  - False otherwise

**Example:**

```python

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

- `isinstance(c1, Circle)` is True
- `isinstance(c1, Shape)` is True
- `isinstance(c1, object)` is True
- `isinstance(c1, str)` is False
  - Generally preferable to `type`
  - Works with base types too!
Q5: `isinstance` and Subclasses

```python
>>> s1 = Rectangle(0,0,10,10)
>>> isinstance(s1, Square)
```

???

A: True  
B: False  
C: Error  
D: I don’t know

Next Lecture

- Programming Practice  
- Develop classes: Animal, Bird, Fish, Penguin, Parrot  
- Instances can swim, fly, and speak based on class membership

A5: `isinstance` and Subclasses

```python
>>> s1 = Rectangle(0,0,10,10)
>>> isinstance(s1, Square)
```

???

A: True  
B: False  
C: Error  
D: I don’t know

Questions to ask

- What does the class hierarchy look like?  
- What are class attributes? What are instance attributes? What are constants?  
- What does the `__init__` function look like?  
- How do we support default weights?  
- How do we implement the class methods?  
- What does a "stringified" Animal look like? `str(a)`