Lecture 22:
Subclasses & Inheritance
(Chapter 18)
CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Fan, D. Gries, L. Lee,
S. Marschner, C. Van Loan, W. White]
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

```python
class Shape():
    """A shape located at x,y"""
    def __init__(self, x, y):
    def draw(self):

class Circle(Shape):
    """An instance is a circle."""
    def __init__(self, x, y, radius):
    def draw(self):

class Rectangle(Shape):
    """An instance is a rectangle."""
    def __init__(self, x, y, ht, len):
    def draw(self):
```

[Diagram showing inheritance relationships between Shape, Circle, and Rectangle classes]
Extending Classes

class `<name>`(<superclass>):

    """Class specification"""
    class variables
    initializer (__init__)
    methods

Class to extend
(may need module name: `<modulename>..<superclass>`)
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?*
__init__: write new one, access parent’s

```
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
```

- Want to use the original version of the method?
  - New method = original+more
  - Don't repeat code from the original
- Call old method explicitly
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

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

```
Initialized in Shape initializer
```
```
Initialized in Circle initializer
```
Can override methods; can access parent’s version

class Shape():
    """Instance is shape @ x,y"""
    def __init__(self, x, y):
    def __str__(self):
        return "Shape @ ("+str(self.x)+", "+str(self.y)+")"
    def draw(self):...

class Circle(Shape):
    """Instance is a Circle @ x,y with radius"""
    def __init__(self, x, y, radius):
    def __str__(self):
        return "Circle: Radius="+str(self.radius)+" "+super().__str__()
    def draw(self):...
Understanding Method Overriding

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

• Which `__str__` do we use?
  ▪ Start at bottom class folder
  ▪ Find first method with name
  ▪ Use that definition

• Each subclass automatically \textit{inherits} methods of parent.

• New method definitions \textit{override} those of parent.
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 object

Often called the **Bottom–Up Rule**

```python
c1 = Circle(1,2,4.0)
r = c1.radius
c1.draw()
```

```python
circle
  __init__(self,x,y)
  draw(self)
```

```python
shape
  __init__(self,x,y)
  draw(self)
```

```python
circle
  __init__(self,x,y, radius)
  draw(self)
```

```
circle
  id3
  x 1
  y 2
  radius 4.0
```
Q1: Name Resolution and Inheritance

```python
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:
  ```python
  >>> a = A()
  >>> b = B()
  ```

- What is value of `a.f()`?

  A: 10  
  B: 14  
  C: 5   
  D: `ERROR`  
  E: `I don’t know`
Q2: Name Resolution and Inheritance

```python
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:
  ```python
  >>> a = A()
  >>> b = B()
  ```

- What is value of `b.f()`?

  A: 10
  B: 14
  C: 5
  D: ERROR
  E: I don’t know
Start next video:
Design choices for method draw
Demo using Turtle Graphics

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

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

Part of the turtle module in Python (docs.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`
Who draws what?

```python
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)
```

Note: need to import 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.

See `shapes.py`, `draw_shapes.py`
Start next video:
Class attributes
Class Variables can also be Inherited

class Shape():  # inherits from object by default
    """""""Instance is shape @ x,y"""""
    # Class Attribute tracks total num shapes
    NUM_SHAPES = 0
    ...

class Circle(Shape):
    """""""Instance is a Circle @ x,y with radius"""""
    # Class Attribute tracks total num circles
    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:
  ```
  >>> a = A()
  >>> b = B()
  ```
- What is value of `b.x`?
  
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A:</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>B:</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C:</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>D:</td>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>E:</td>
<td>I don’t know</td>
<td></td>
</tr>
</tbody>
</table>
  ```
Q4: 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 `a.z`?
  
<table>
<thead>
<tr>
<th>A: 4</th>
<th>B: 3</th>
<th>C: 42</th>
<th>D: ERROR</th>
<th>E: I don’t know</th>
</tr>
</thead>
</table>

23
Why override \_\_eq\_\_? Compare equality

class Shape:

"""Instance is shape @ x,y"""
    def \_\_init\_(self,x,y):
    def \_\_eq\_(self, other):
        """If position is the same, then equal as far as Shape knows"""
        return self.x == other.x and self.y == other.y

class Circle(Shape):

"""Instance is a Circle @ x,y with radius"""
    def \_\_init\_(self,x,y,radius):
    def \_\_eq\_(self, other):
        """If radii are equal, let super do the rest"""
        return self.radius == other.radius and super().\_\_eq\_(other)
eq vs. is

== compares equality
is compares identity

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

c1 == c2 ?
c1 is c2 ?
c2 == c3 ?
c2 is c3 ?
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
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