http://www.cs.cornell.edu/courses/cs1110/2018sp

Lecture 19: Subclasses & Inheritance

(Chapter 18)

CS 1110

Introduction to Computing Using Python

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

Sharing Work

Problem: Redundant code.

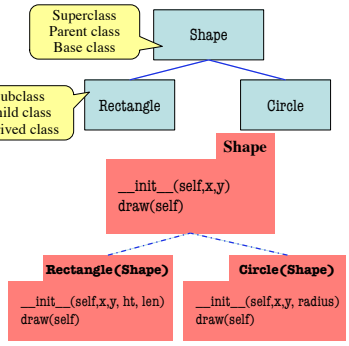
(Any time you copy-and-paste code, you are likely doing something wrong.)

Solution: Create a *parent* class with shared code

- Then, create *subclasses* of the *parent* class

Defining a Subclass

```
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 in stance is a rectangle. """
    def __init__(self, x, y, ht, len): ...
    def draw(self): ...
```



Extending Classes

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

"""Class specification"""

class variables

initializer (__init__)

methods

Class to extend (may need module name)

So far, classes have implicitly extended object

__init__

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

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

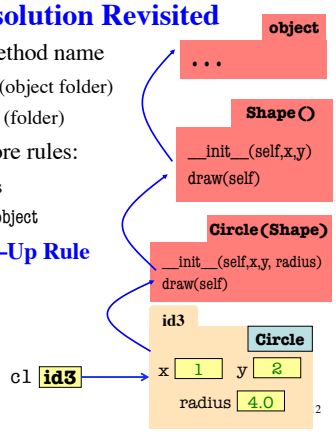
- Want to use the original version of the method?
  - New method = original+more
  - Do not want to repeat code from the original version
- Call old method **explicitly**

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**

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



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

13

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

15

Q3: Name Resolution and Inheritance

```
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?

A: 4
B: 3
C: 42
D: **ERROR**
E: I don't know

19

Q4: Name Resolution and Inheritance

```
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 a.z?

A: 4
B: 3
C: 42
D: **ERROR**
E: I don't know

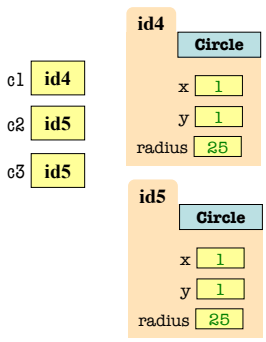
21

The is Function

== compares **equality**

is compares **identity**

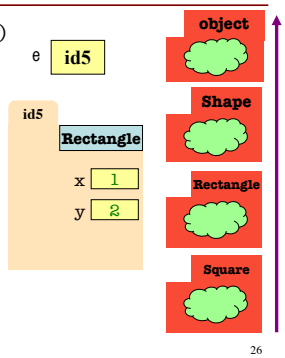
```
c1 = Circle(1, 1, 25)
c2 = Circle(1, 1, 25)
c3 = c2

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



Q5: isinstance and Subclasses

```
>>> shape1 = Rectangle(0,0,10,10)
>>> isinstance(shape1, Square)
???
```

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



26