



# Lecture 23: More on Subclassing (Chapter 18)

CS 1110

Introduction to Computing Using Python

*Revised after lecture:* on slide 12, the class Shape folder's tab should read Shape(object); the class Circle folder's tab should read Circle(Shape)

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

3

## Announcements

- **Assignment 5** due Wedn May 5<sup>th</sup>
  - Minor update—read cover page of A5 pdf posted on course website
- **Prelim 2:** we expect feedback to be available on Monday
- **WICC** (student org **Women in Computing At Cornell**) invites responses from CIS students on how the semester has gone: <https://forms.gle/L72qPkYvYJDJ8cdx9>

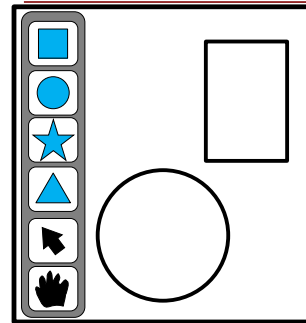
## Topics

Continuation from last lecture

- Design considerations for overriding methods
- Class attributes
- Different kinds of comparisons on objects

4

## Goal: Make a drawing app



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

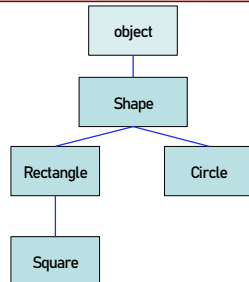
5

## Example

```
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): ...
```



[Optional] wondering what's in the object class? See <https://docs.python.org/3/reference/datamodel.html#basic-customization>

...

## Extending Classes

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

```
    """Class specification"""
    <class variables>
    <initializer>
    <methods>
```

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

- From last lecture:
- Parent class (superclass)
  - Child class (subclass)
  - Attributes, methods are inherited
  - Can override parent's method
  - Function call `super()` to access method of parent

7

# Design choices for method draw

## 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`

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

Part of the turtle module in Python ([docs.python.org/3.7/library/turtle.html](https://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`

8

9

## 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)
```

Note: need to import the `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 `shapes_v3.py`, `draw_shapes.py`

# Class attributes

11

## 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
    ...
```

Diagram showing inheritance of class variables: `object` (NUM\_SHAPES = 0) is inherited by `Shape` (NUM\_SHAPES = 0), which is inherited by `Circle` (NUM\_CIRCLES = 0).

12

## Q1: 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

13

## Q2: 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

Different kinds of comparisons

15

17

## 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)
```

Want to compare equality of the values (data) of two instances, not the id of the two instances!

## Q3: 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 → ?

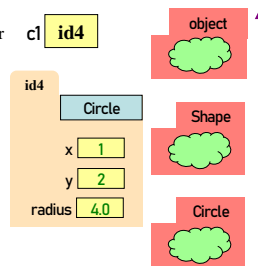
## The `isinstance` Function

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

**Example:**

```
c1 = Circle(1,2,4.0)
instance(c1,Circle) is True
instance(c1,Shape) is True
instance(c1,object) is True
instance(c1,str) is False
```

- Generally preferable to `type`
  - Works with base types too!

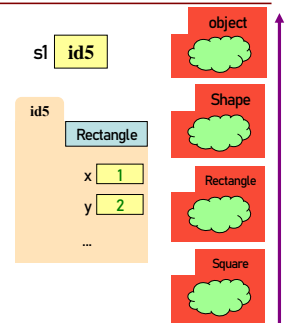


22

## Q4: `isinstance` and Subclasses

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

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



23