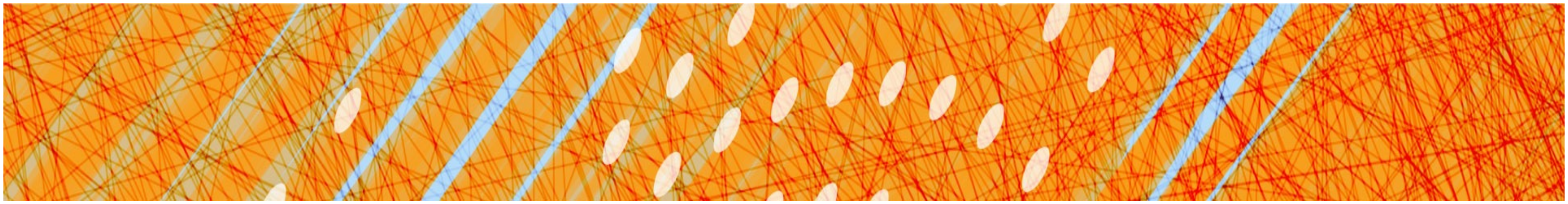


Lecture 19: **More on Subclassing** (Chapter 18)

CS 1110

Introduction to Computing Using Python

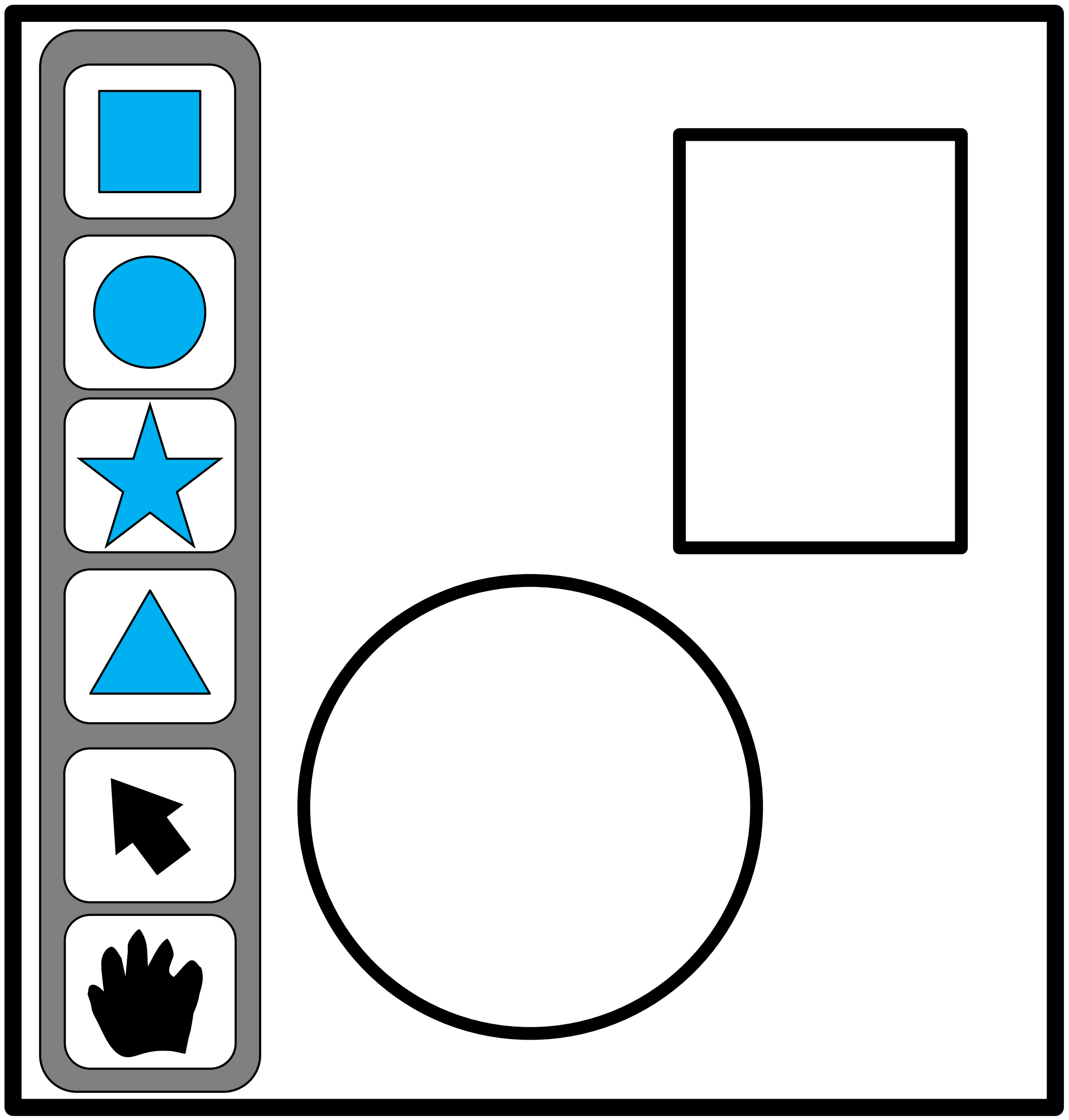
[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

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….

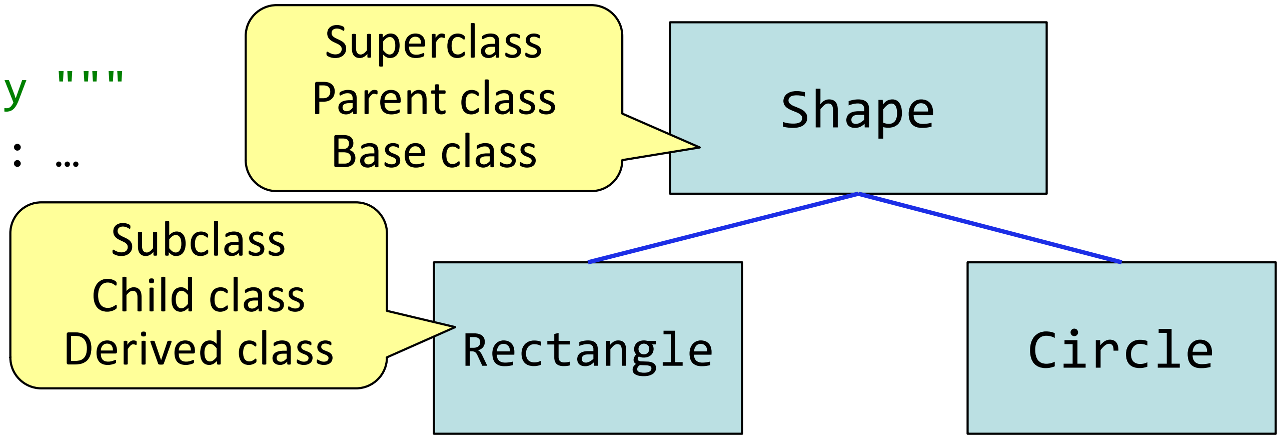
See `shapes_v0.py` → `shapes_v1.py` → `shapes_v2.py`

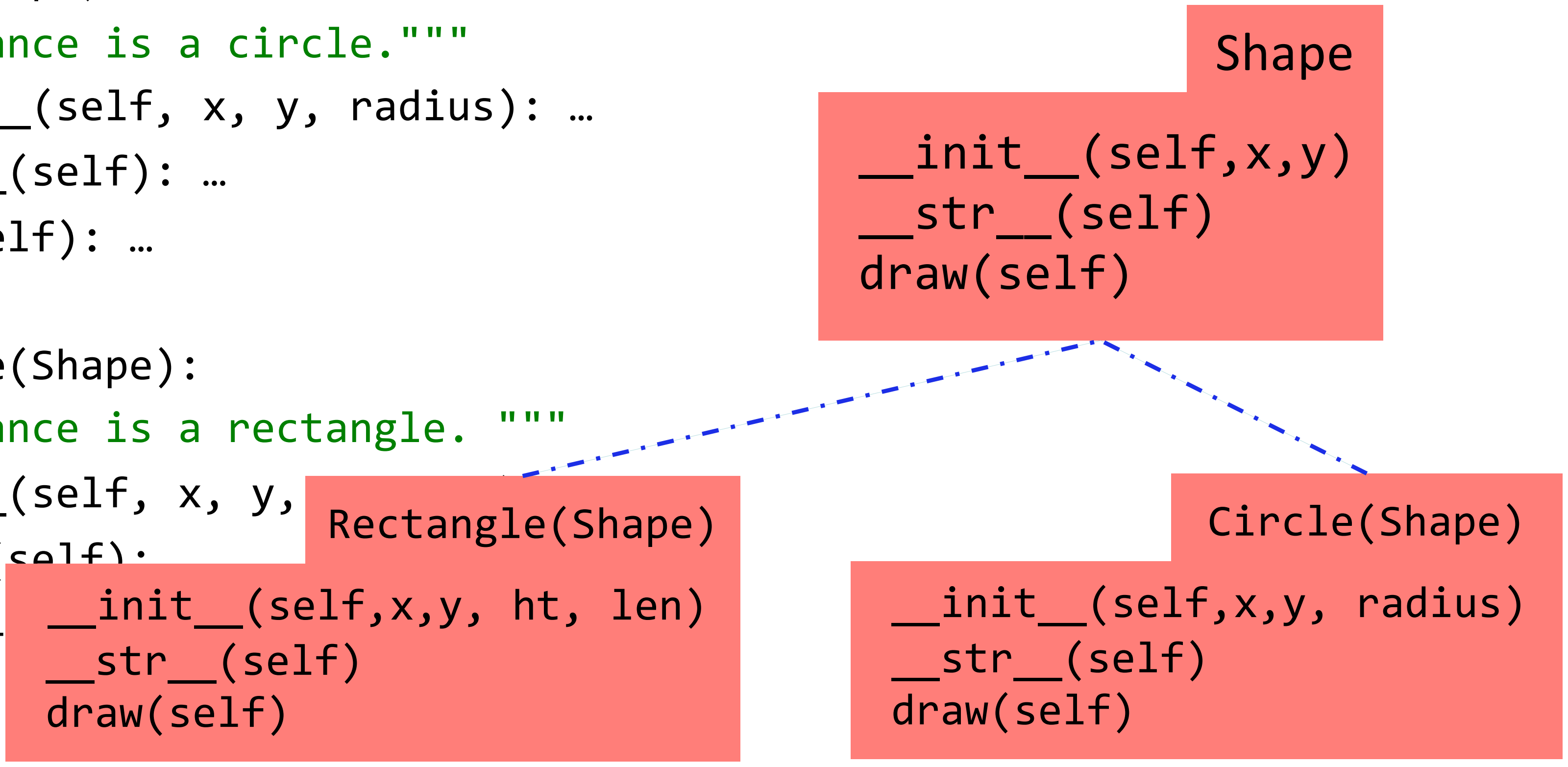
Recall: our Class Hierarchy

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

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

class Rectangle(Shape):
   """An in stance is a rectangle. """
   def __init__(self, x, y,
   def __str__(self):
   def draw(sel
```





Recall : **overriding** & **calling** `__init__`

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

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

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

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)

- 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?

```
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.

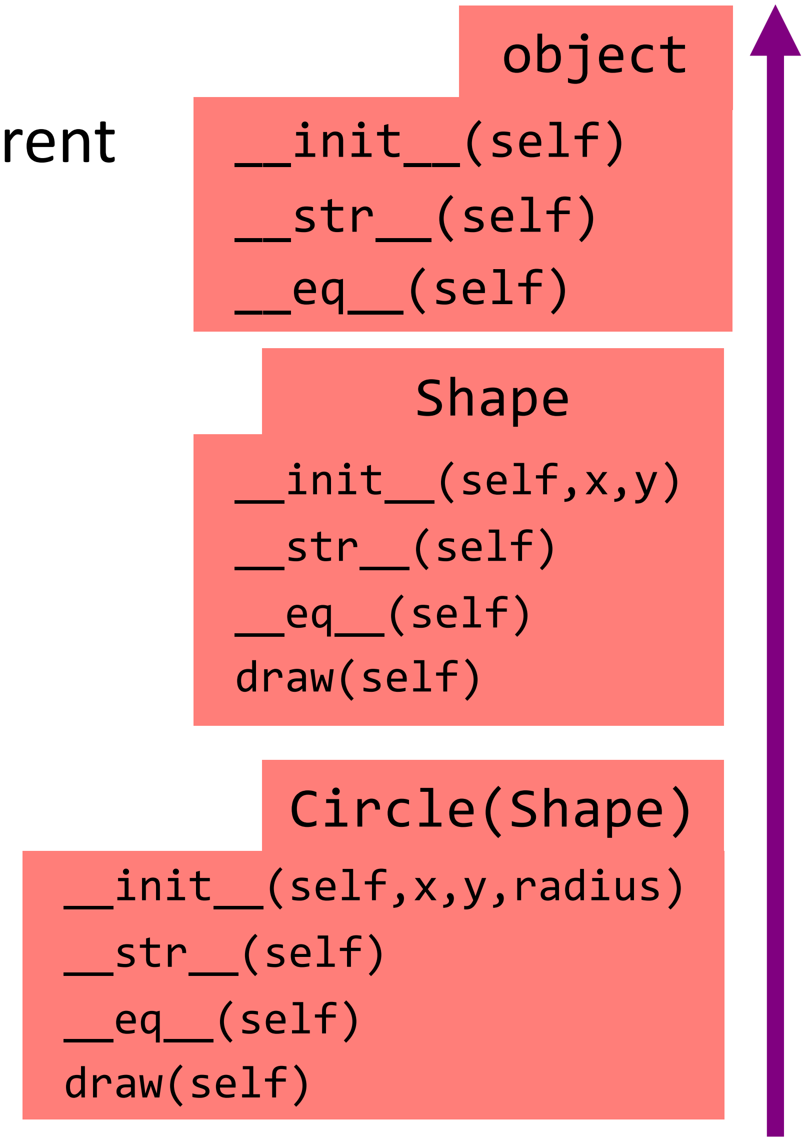
See shapes_v3.py, draw_shapes.py

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



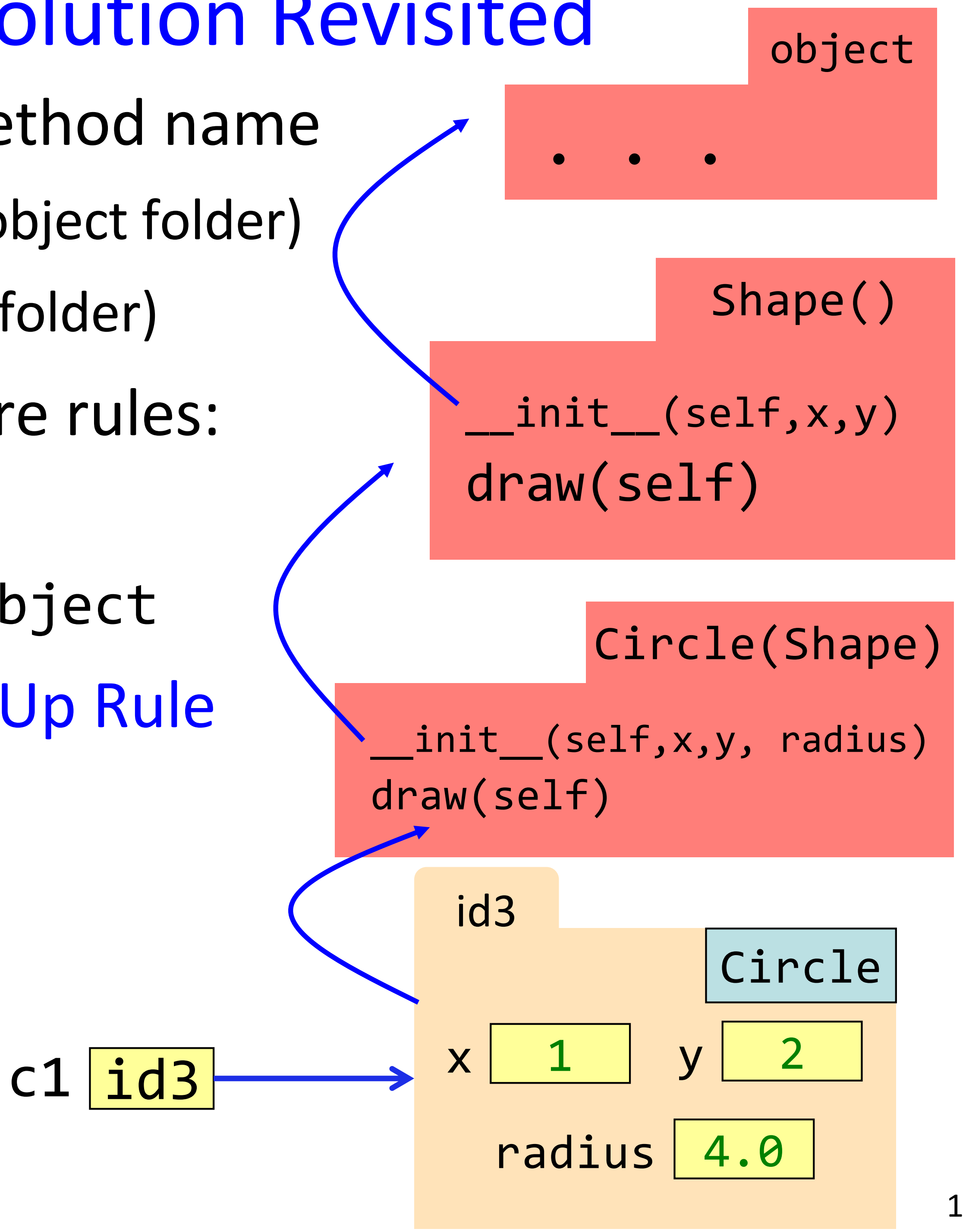
*[**Optional**] wondering what's in the object class? See*
https://docs.python.org/3/reference/datamodel.html#basic-customization

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

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:

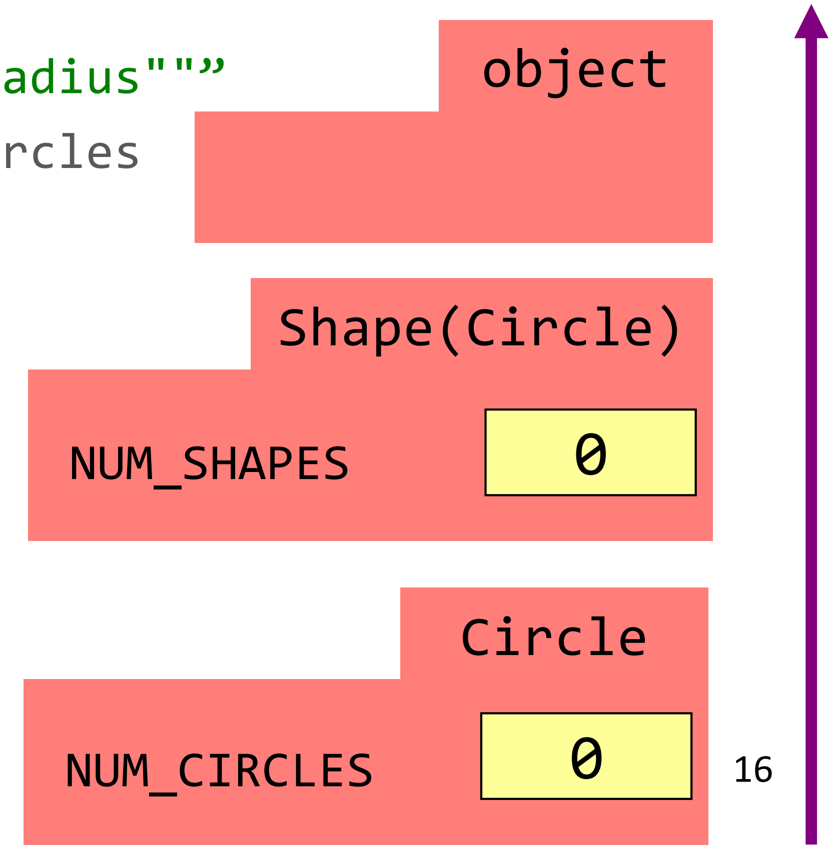
```
>>> a = A()
>>> b = B()
```

- 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"""
    # 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

```
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

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

Next Lecture

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

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)`