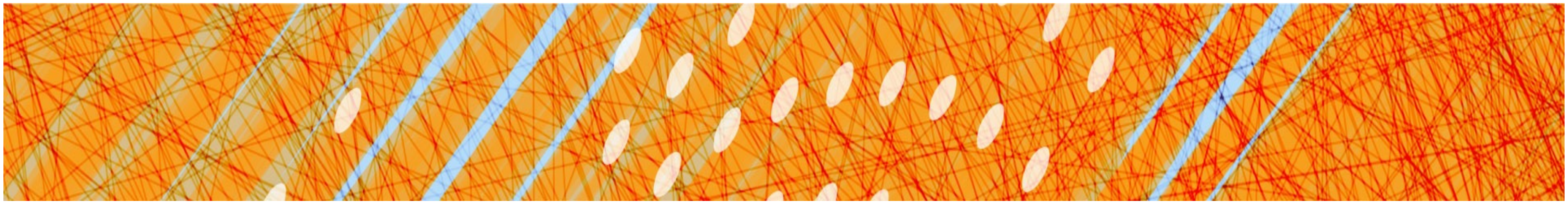


Lecture 18: **Subclasses & Inheritance**

(Chapter 18)

CS 1110

Introduction to Computing Using Python

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

Do you have a conflict with Prelim 2?

A. Yes, and I've filled out the Prelim 2 Conflict Survey.

B. Yes, but I haven't yet filled out the Prelim 2 Conflict Survey.

C. I have this sinking feeling I might have a conflict.

D. I know I do not have a conflict.

E. I have no idea.

Are you in one of these classes?? ASL 1102, BTRY 4090, CEE 4750, CEE 6075, CEE 6750, CHEM 1570, CHEM 2070, CHEM 3580, CHEM 3590, CHEM 3900, CHEME 2200, CHEME 3320

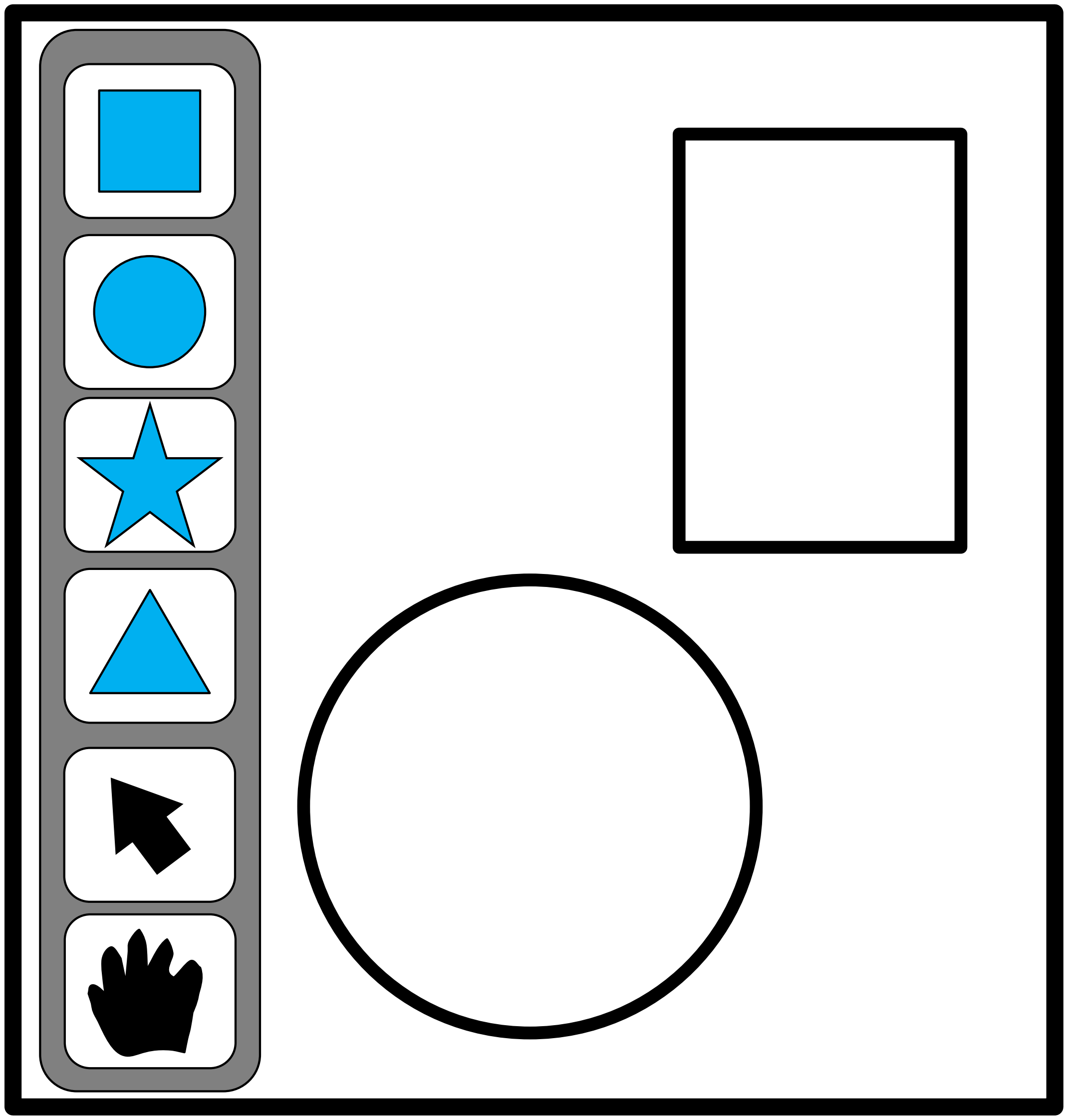
Announcements

- A5 is out! This assignment will help you study for Prelim 2 (and the Final Exam).
- Prelim 2 is next week!
  - See the Exams page for topic coverage, a draft reference sheet, and past Prelim 2 exams
  - Conflict survey is due Wednesday Night
- The Final Exam is May 18 @ 2pm. We do not offer early exams, so if you have a conflict, you would be taking it later, not earlier.

Topics

- Why define subclasses?
  - Understand the resulting hierarchy
  - Design considerations
- How to define a subclass
  - Initializer
  - New methods
  - Write modified versions of inherited methods
  - Access parent’s version using super()

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`

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

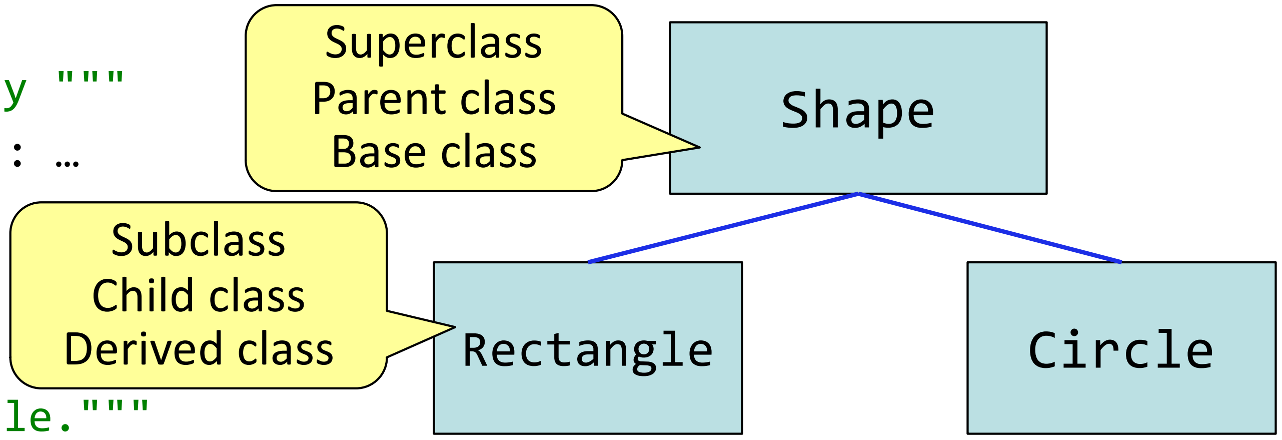
See `shapes_v1.py`

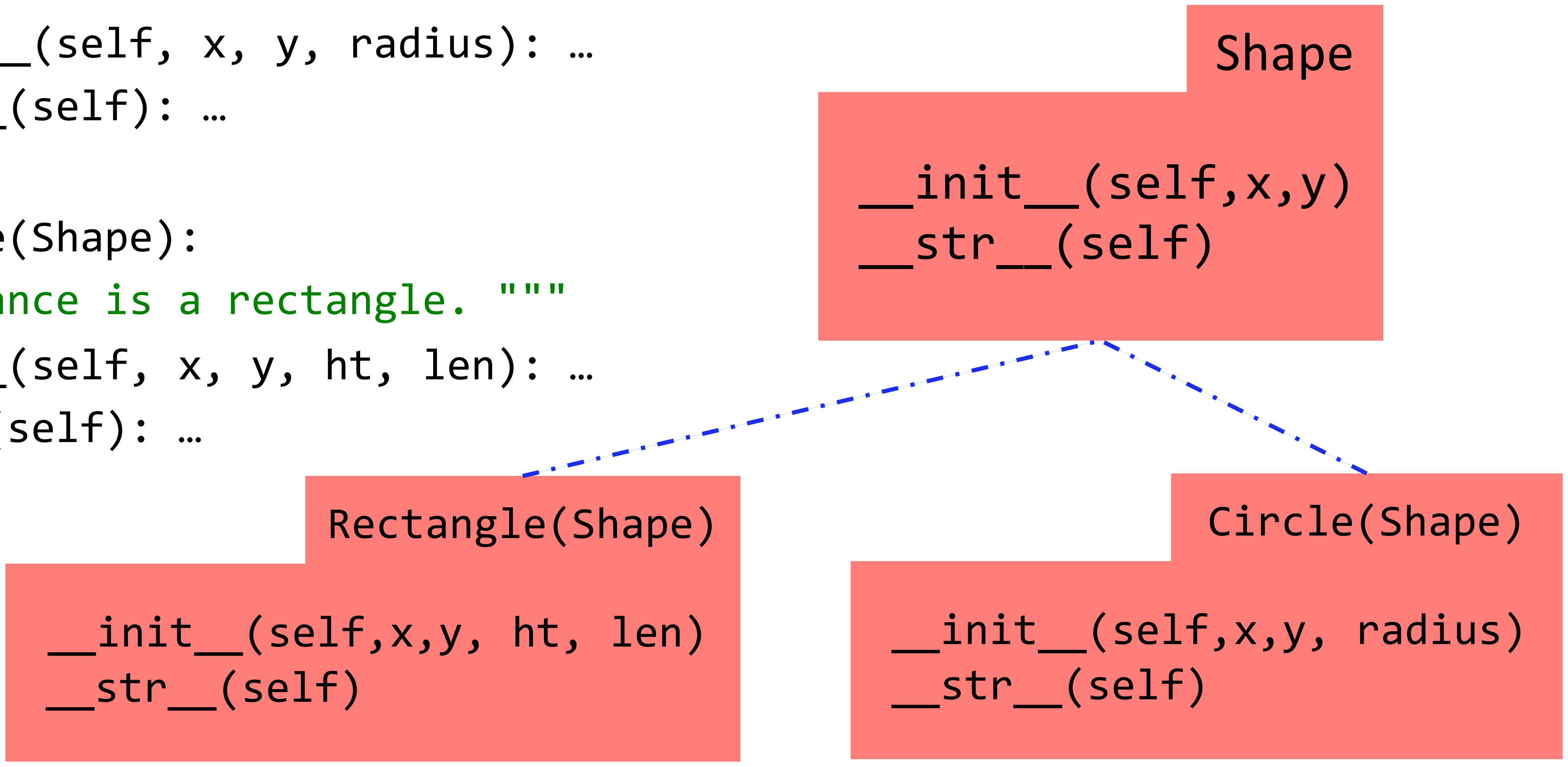
Defining a Subclass

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

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

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





Extending Classes

```
class <name>(<superclass>):
  """Class specification"""
  <class variables>
  <initializer>
  <methods>
```

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

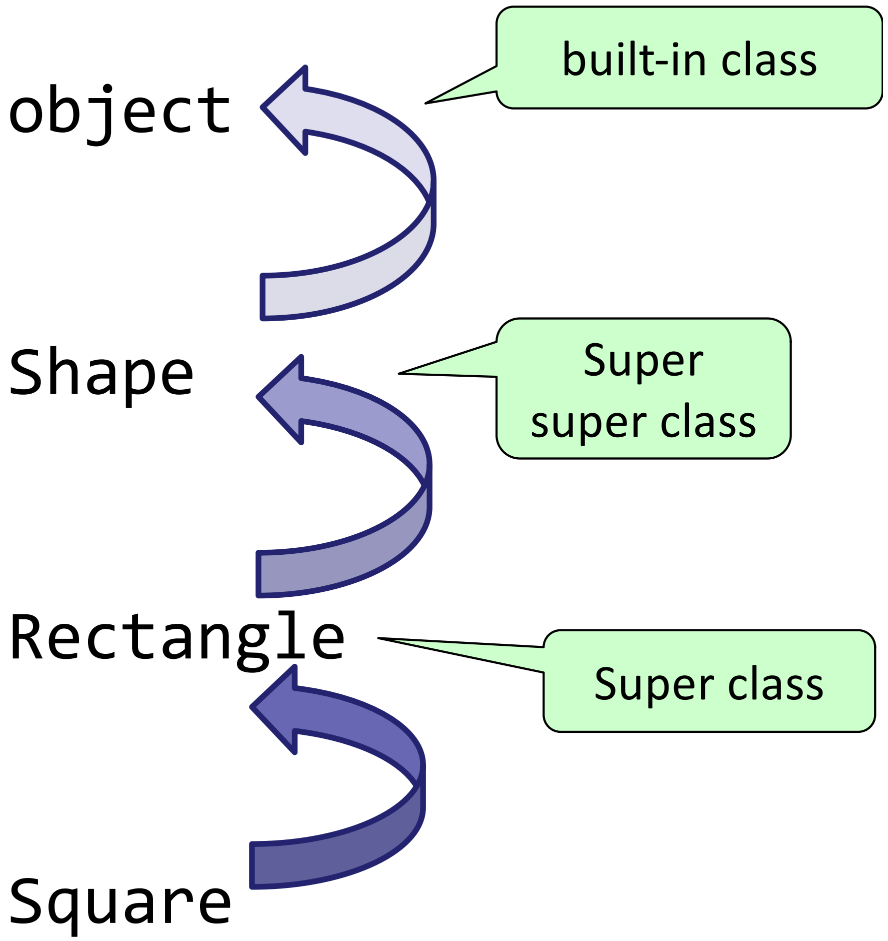
So far, classes have implicitly extended `object`

`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?

Example



`__init__`: write new one, access parent's

```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
```

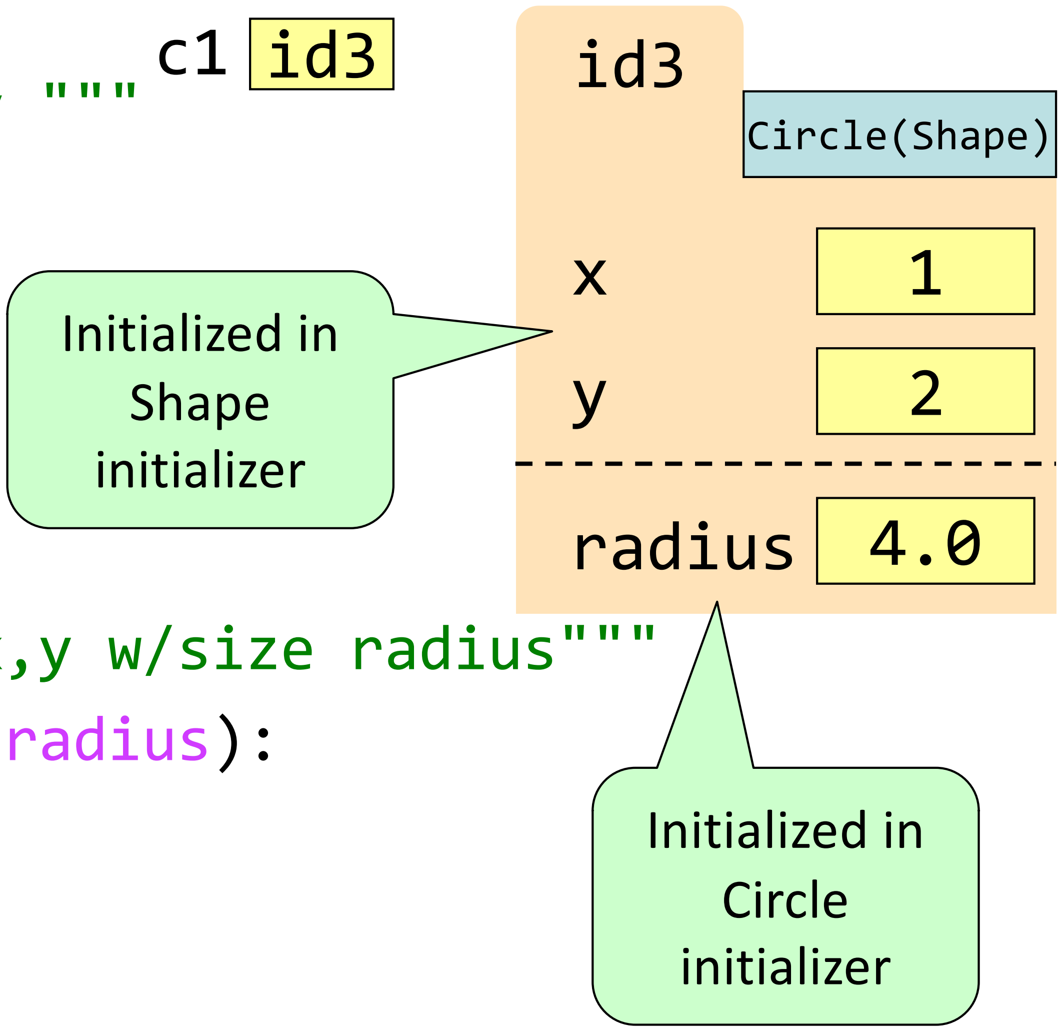
- Want to use the original version of the method?
  - New method = original+more
  - Don't repeat code from the original
- Call old method explicitly

Object Attributes can be Inherited

```
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


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

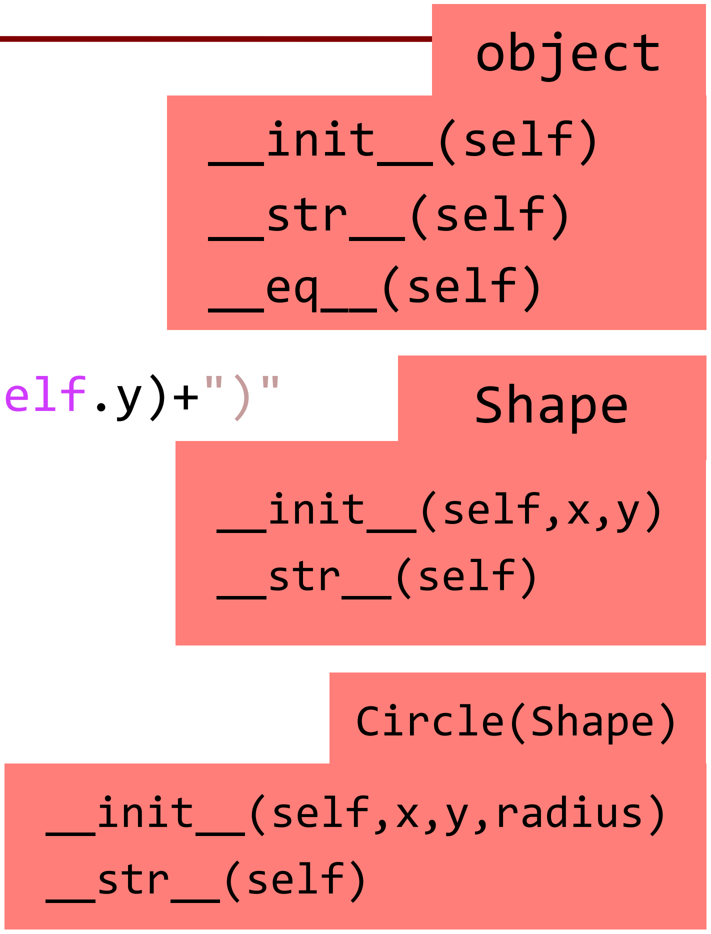


c1 id3
id3
Circle(Shape)
x 1
y 2
radius 4.0
Initialized in Shape initializer
Initialized in Circle initializer

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)+")"


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__()
```



See shapes_v2.py

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!

Inheritance-related terminology

- `eq` vs `is`
- `isinstance`

eq vs. is

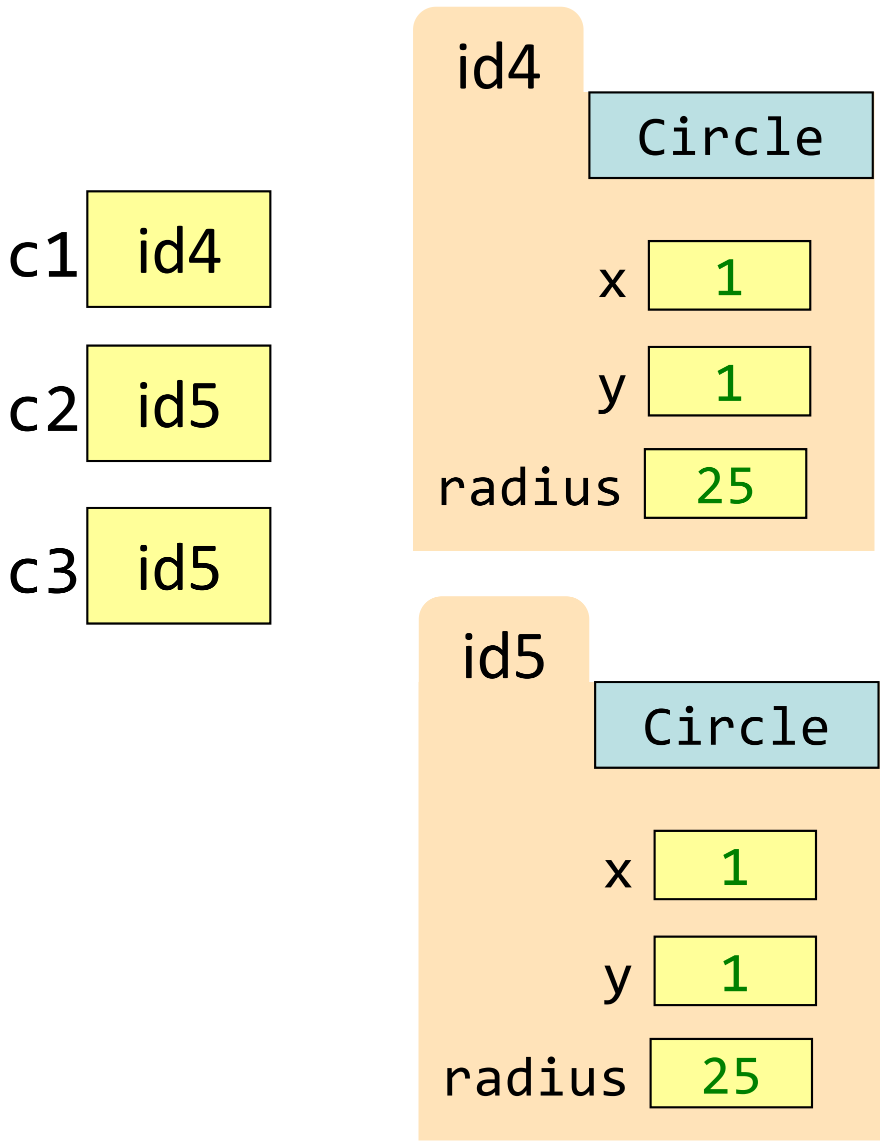
`==` compares equality

`is` compares identity

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

(A) True (B) False (C) I don't know.

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



The `isinstance` Function

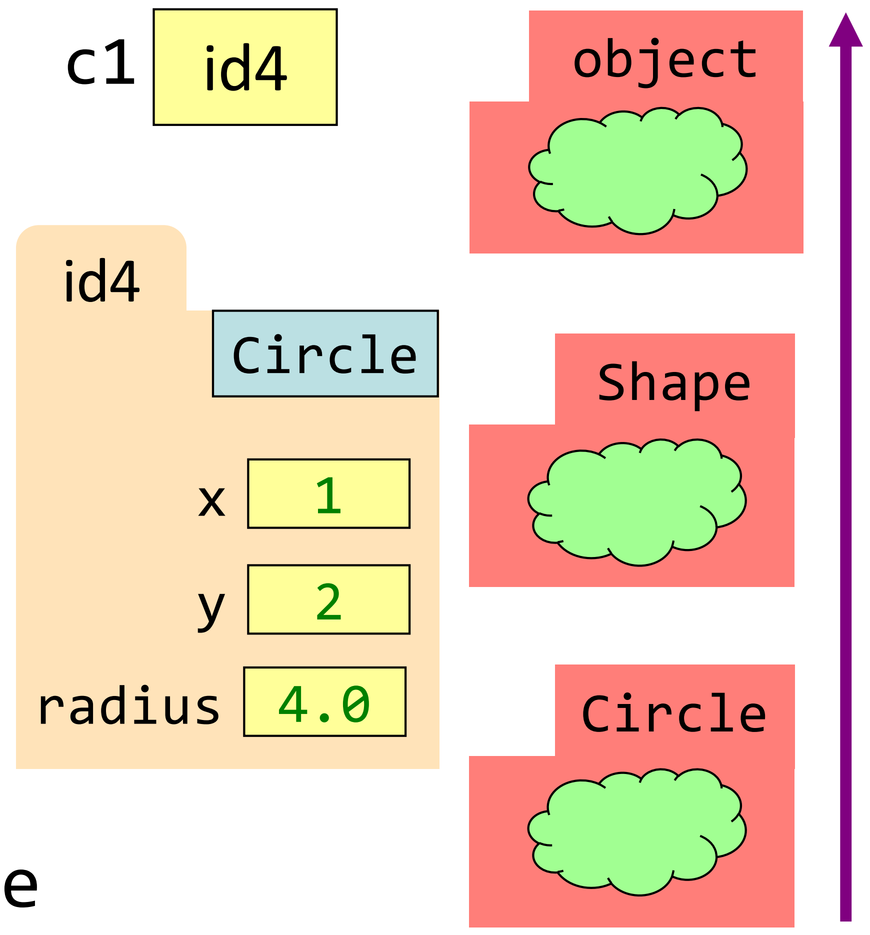
`isinstance(<obj>,<class>)`

- True if <obj>'s class is same as or a subclass of <class>
- False otherwise

Example:

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

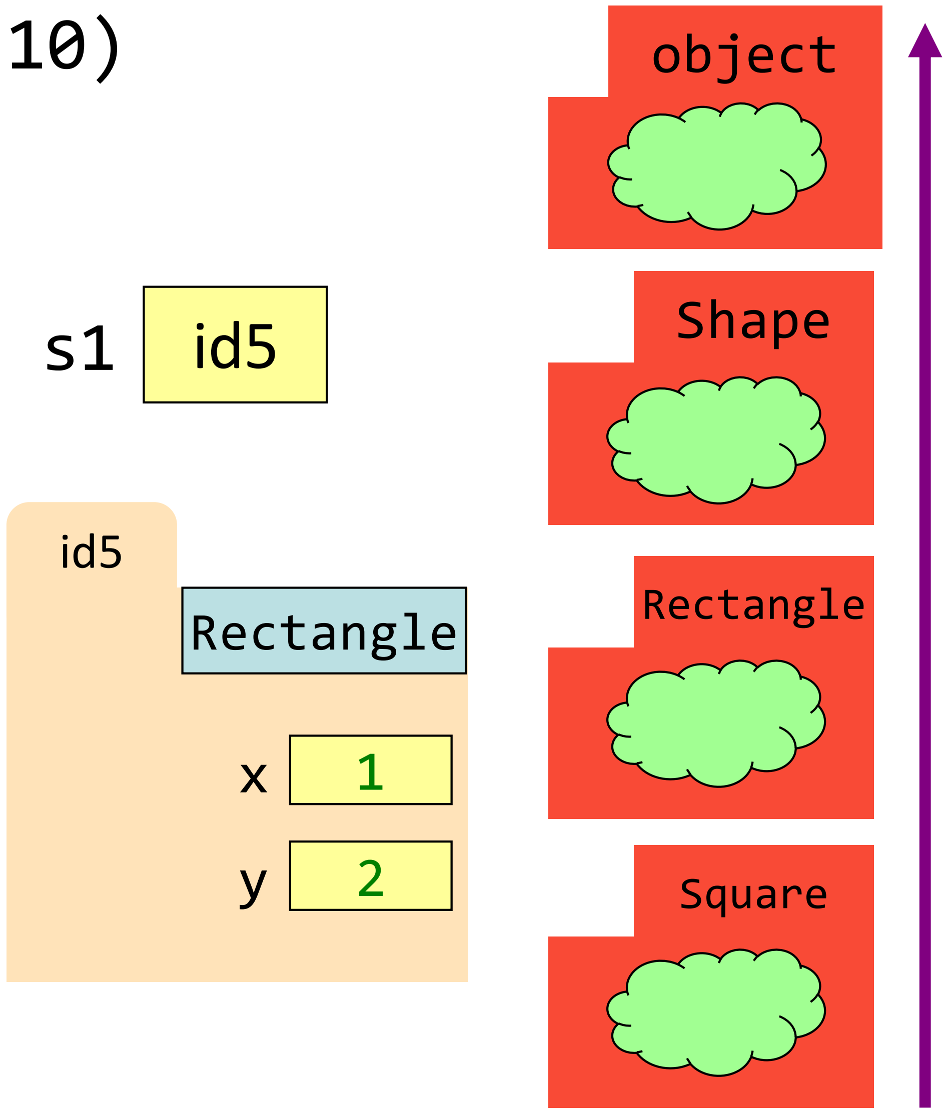
- `isinstance(c1,Circle)` is True
- `isinstance(c1,Shape)` is True
- `isinstance(c1,object)` is True
- `isinstance(c1,str)` is False
- Generally preferable to `type`
  - Works with base types too!



isinstance and Subclasses

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

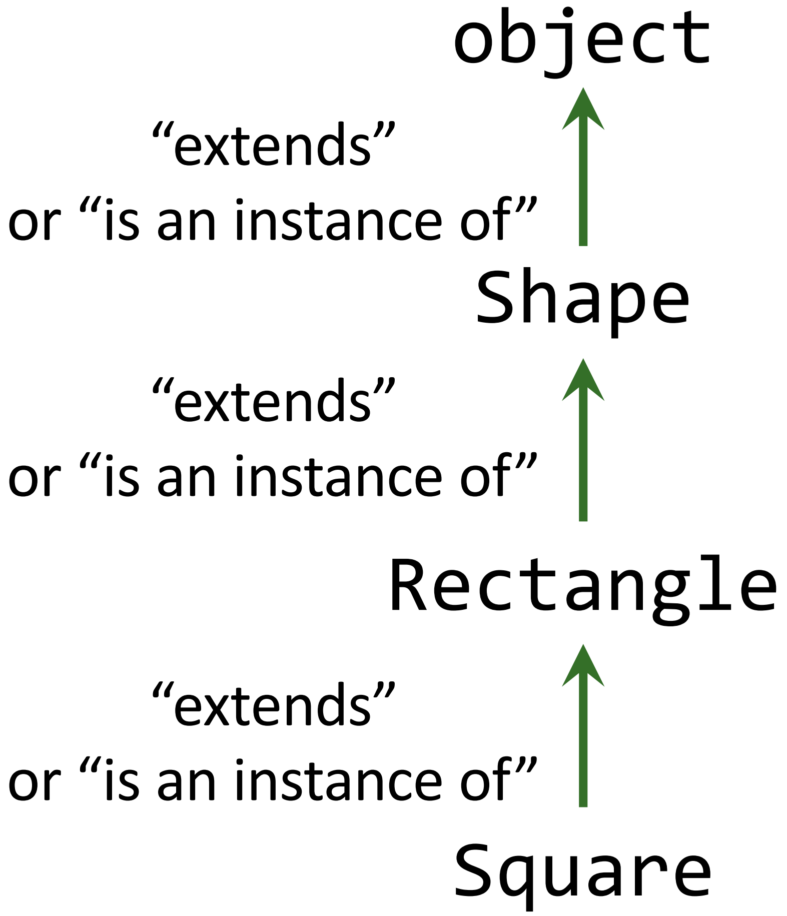
A: True
B: False
C: Error
D: I don't know



isinstance and Subclasses

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

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


object
"extends"
or "is an instance of"
Shape
"extends"
or "is an instance of"
Rectangle
"extends"
or "is an instance of"
Square