Program design

- Example: drawing program (e.g. PowerPoint)
- Many types of content can appear on slides
- Want do do things like

```
for x in slide[i].contents:
    x.draw(window)
```

- No problem: define class for every type of content (text box, rectangle, image, …), make sure each has a `draw` method

Sharing work

- Defining separate classes for text box, image, etc. is fine, but could get repetitive
  - all have code for drawing selection handles, frames, backgrounds, …
- Solution: make these shapes *subclasses* of a single class, where the shared code lives

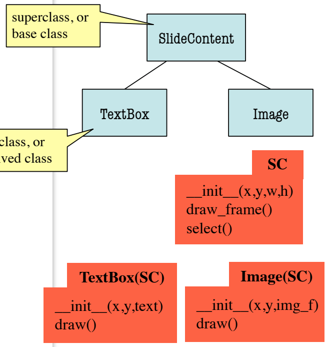
Customizing a class

- Example: telephony program (e.g. Skype)
- Call and Hang Up buttons should be green and red (to follow convention from cell phones)
- Already have a class for normal buttons
- Implement from scratch? No, what a waste…
- Instead create a *subclass* of the button class that is just like a normal button, except it draws itself with a different color.

Defining a subclass

(abbreviated SC on this slide)

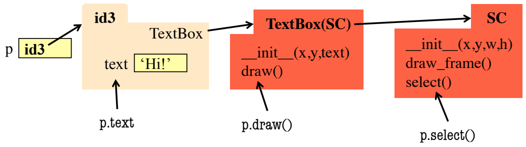
```
class SlideContent(object):
    """Any object on a slide."""
    def __init__(self, x, y, w, h): ...
    def draw_frame(self): ...
    def select(self): ...

class TextBox(SlideContent):
    """An object containing text."""
    __init__(self, x, y, text): ...
    def draw(self): ...

class Image(SlideContent):
    """An image."""
    __init__(self, x, y, image_file): ...
    def draw(self): ...
```


superclass, or base class

subclass, or derived class

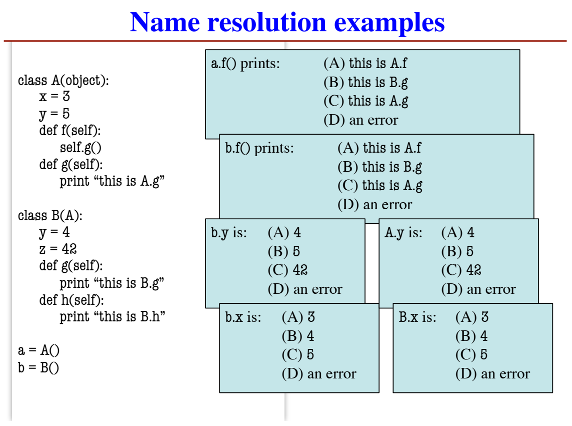
Names in subclasses and superclasses

- Recall rule for looking up attribute names in classes: look first in the instance, then in the class.
- With inheritance, there's one simple addition: look in the instance, then in the class, then in the superclass.



Inheritance

- Superclass also called "parent"
- If subclass does nothing special, it has all the same attributes as the parent class—it *inherits* all the methods and variables
- Subclass can *add* new methods and variables (with different names)
- Subclass can *override* methods and class variables (by giving them the same names)



Name resolution examples

```
class A(object):
    x = 3
    y = 5
    def f(self):
        self.g()
    def g(self):
        print “this is A.g”

class B(A):
    y = 4
    z = 42
    def g(self):
        print “this is B.g”
    def h(self):
        print “this is B.h”

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

a.f() prints: (A) this is A.f (B) this is B.g (C) this is A.g (D) an error

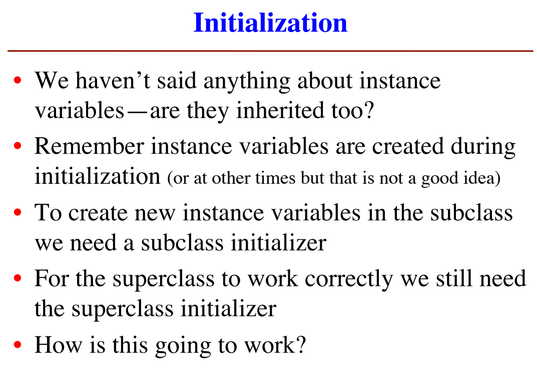
b.f() prints: (A) this is A.f (B) this is B.g (C) this is A.g (D) an error

b.y is: (A) 4 (B) 5 (C) 42 (D) an error

A.y is: (A) 4 (B) 5 (C) 42 (D) an error

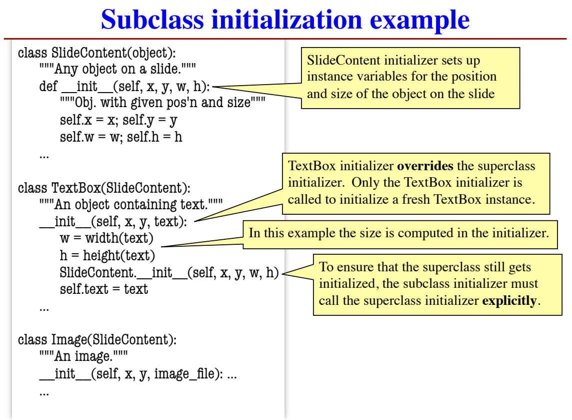
b.x is: (A) 3 (B) 4 (C) 5 (D) an error

B.x is: (A) 3 (B) 4 (C) 5 (D) an error



Initialization

- We haven’t said anything about instance variables—are they inherited too?
- Remember instance variables are created during initialization (or at other times but that is not a good idea)
- To create new instance variables in the subclass we need a subclass initializer
- For the superclass to work correctly we still need the superclass initializer
- How is this going to work?



Subclass initialization example

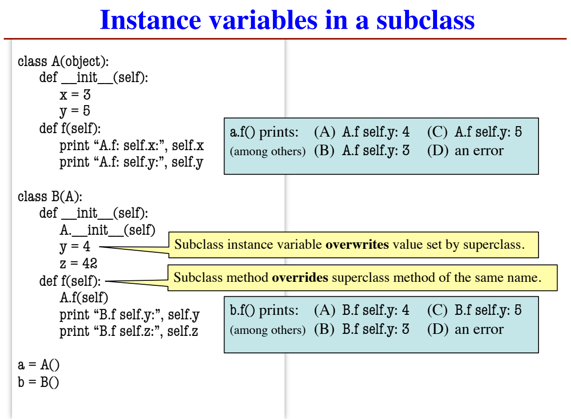
```
class SlideContent(object):
    """Any object on a slide."""
    def __init__(self, x, y, w, h):
        """Obj. with given pos'n and size"""
        self.x = x; self.y = y
        self.w = w; self.h = h
    ...

class TextBox(SlideContent):
    """An object containing text."""
    __init__(self, x, y, text):
        w = width(text)
        h = height(text)
        SlideContent.__init__(self, x, y, w, h)
        self.text = text
    ...

class Image(SlideContent):
    """An image."""
    __init__(self, x, y, image_file): ...
    ...
```

SlideContent initializer sets up instance variables for the position and size of the object on the slide

TextBox initializer **overrides** the superclass initializer. Only the TextBox initializer is called to initialize a fresh TextBox instance.

In this example the size is computed in the initializer.

To ensure that the superclass still gets initialized, the subclass initializer must call the superclass initializer **explicitly**.



Instance variables in a subclass

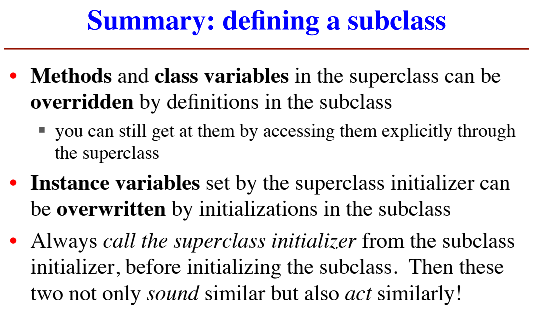
```
class A(object):
    def __init__(self):
        x = 3
        y = 5
    def f(self):
        print “A.f: self.x:”, self.x
        print “A.f: self.y:”, self.y

class B(A):
    def __init__(self):
        A.__init__(self)
        y = 4
        z = 42
    def f(self):
        A.f(self)
        print “B.f self.y:”, self.y
        print “B.f self.z:”, self.z

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

a.f() prints: (among others) (A) A.f self.y: 4 (B) A.f self.y: 3 (C) A.f self.y: 5 (D) an error

Subclass instance variable **overwrites** value set by superclass.

Subclass method **overrides** superclass method of the same name.

b.f() prints: (among others) (A) B.f self.y: 4 (B) B.f self.y: 3 (C) B.f self.y: 5 (D) an error



Summary: defining a subclass

- **Methods** and **class variables** in the superclass can be **overridden** by definitions in the subclass
    - you can still get at them by accessing them explicitly through the superclass
- **Instance variables** set by the superclass initializer can be **overwritten** by initializations in the subclass
- Always *call the superclass initializer* from the subclass initializer, before initializing the subclass. Then these two not only *sound* similar but also *act* similarly!