Program design

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

Definitions of a subclass

```python
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."""
    def __init__(self, x, y, text):
        ...
    def draw(self):
        ...

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

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.

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.
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)

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?

Instance variables in a subclass

```
class A(object):
    def __init__(self):
        self.x = 3
        self.y = 5
        def __init__(self):
            print "A.\_init\_()", self.x
            print "A.\_init\_()", self.y

class B(A):
    def __init__(self):
        super(B, self).__init__()
        self.x = 42
        def __init__(self):
            super(B, self).__init__()
            print "B.\_init\_()", self.x
            print "B.\_init\_()", self.y
```

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 overridden 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!

Name resolution examples

```
class A(object):
    x = 3
    y = 5
    def __init__(self):
        super(A, self).__init__()
        self.g = 42
        print "this is A.g"

class B(A):
    y = 4
    x = 42
    def __init__(self):
        super(B, self).__init__()
        print "this is B.g"
        def __init__(self):
            super(B, self).__init__()
            print "B.\_init\_()", self.x
            print "B.\_init\_()", self.y
```

Subclass initialization example

```
class SlideContent(object):
    """Any object on a slide."
    def __init__(self, x, y, w, h):
        """Obj. with given postion and size"
        self.x = x; self.y = y
        self.w = w; self.h = h
        ...
        def h(self):
            """SlideContent\_h() method"""
            return self.h

    def g(self):
        """SlideContent\_g() method"""
        return self.g

class TextBox(SlideContent):
    """An object containing text."
    def __init__(self, x, y, text):
        """w = width(text)
        h = height(text)
        SlideContent\_init\_(self, x, y, w, h)
        self.text = text"
        ...
        def f(self):
            """SlideContent\_f() method"""
            return self.f

    def e(self):
        """SlideContent\_e() method"""
        return self.e

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

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 overridden 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!