An Application

- **Goal**: Presentation program (e.g. PowerPoint)
- **Problem**: There are many types of content
  - **Examples**: text box, rectangle, image, etc.
  - Have to write code to display each one
- **Solution**: Use object oriented features
  - Define class for every type of content
  - Make sure each has a draw method:

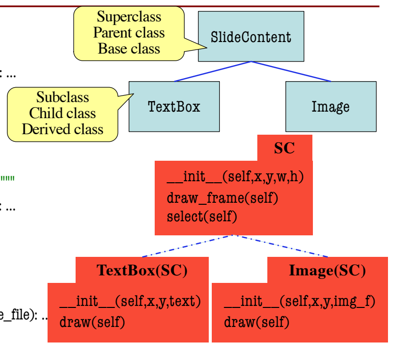
```
for x in slide[i].contents:
    x.draw(window)
```

1

Defining a Subclass

Abbreviate as SC to right

```
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): ...
```



2

Class Definition: Revisited

```
class <name>(<superclass>):
    """Class specification"""
    getters and setters
    initializer (__init__)
    definition of operators
    definition of methods
    anything else
```

Class type to extend (may need module name)

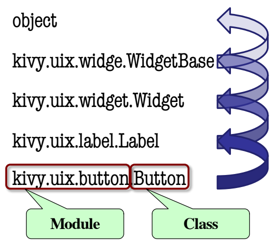
- Every class must extend *something*
- Previous classes all extended object

3

object and the Subclass Hierarcy

- Subclassing creates a **hierarchy** of classes
  - Each class has its own super class or parent
  - Until object at the "top"
- object has many features
  - Special built-in fields: __class__, __dict__
  - Default operators: __str__, __repr__

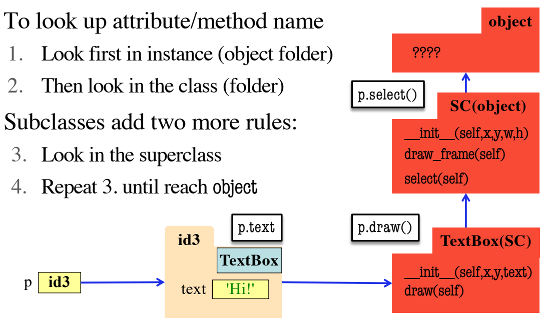
Kivy Example



4

Name Resolution Revisited

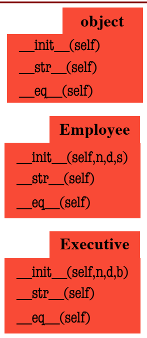
- To look up attribute/method name
  1. Look first in instance (object folder)
  2. Then look in the class (folder)
- Subclasses add two more rules:
  3. Look in the superclass
  4. Repeat 3. until reach object



5

A Simpler Example

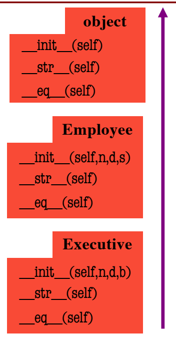
```
class Employee(object):
    """Instance is salaried worker"""
    # INSTANCE ATTRIBUTES:
    # _name:  full name, a string
    # _start:  first year hired,
    #   an int ≥ -1, -1 if unknown
    # _salary: yearly wage, a float

class Executive(Employee):
    """An Employee with a bonus"""
    # INSTANCE ATTRIBUTES:
    #_bonus: annual bonus, a float
```



6

Method Overriding

- Which __str__ do we use?
  - Start at bottom class folder
  - Find first method with name
  - Use that definition
- New method definitions **override** those of parent
  - Access to old version is **lost**
  - New version used instead
  - **Example**: __init__



7

Accessing the “Previous” Method

- What if you want to use the original version method?
  - New method = original+more
  - Do not want to repeat code from the original version
- Use the function super()
  - “Converts” type to parent class
  - Now methods go to the class
- **Example**:

  super().__str__()

  In Python 2 self goes here

```
class Employee(object):
    """An Employee with a salary"""
    ...
    def __str__(self):
        return (self._name +
                ', year ' + str(self._start) +
                ', salary ' + str(self._salary))

class Executive(Employee):
    """An Employee with a bonus."""
    ...
    def __str__(self):
        return (super().__str__()
                + ', bonus ' + str(self._bonus) )
```

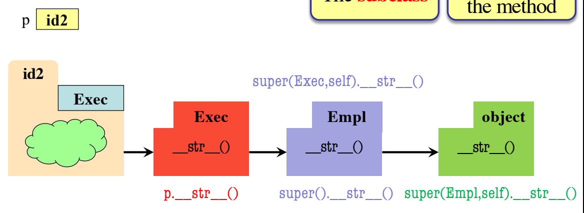
8

About super()

- super() is very limited
  - Can only go one level
  - **BAD**: super().super()
- Need arguments for more
  - super(class,self)

The subclass

Object in the method

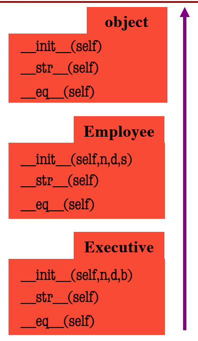


9

Primary Application: Initializers

```
class Employee(object):
    ...
    def __init__(self,n,d,s=50000.0):
        self._name = n
        self._start = d
        self._salary = s
```

```
class Executive(Employee):
    ...
    def __init__(self,n,d,b=0.0):
        super().__init__(n,d)
        self._bonus = b
```

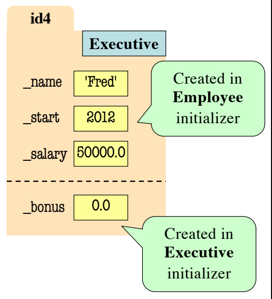


10

Instance Attributes are (Often) Inherited

```
class Employee(object):
    ...
    def __init__(self,n,d,s=50000.0):
        self._name = n
        self._start = d
        self._salary = s
```

```
class Executive(Employee):
    ...
    def __init__(self,n,d,b=0.0):
        super().__init__(n,d)
        self._bonus = b
```


id4
Executive
_name
'Fred'
_start
2012
_salary
50000.0
_bonus
0.0
Created in Employee initializer
Created in Executive initializer

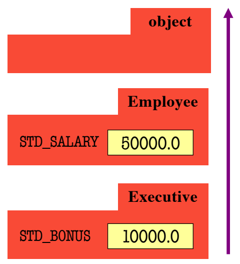
11

Also Works With Class Attributes

Class Attribute: Assigned outside of any method definition

```
class Employee(object):
    """Instance is salaried worker"""
    # Class Attribute
    STD_SALARY = 50000.0
```

```
class Executive(Employee):
    """An Employee with a bonus."""
    # Class Attribute
    STD_BONUS = 10000.0
```



12