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:

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

Class Definition: Revisited

```python
class <name>(<superclass>):

    """Class specification"
    """
    # geters and setters
    # initializer (__init__)
    # definition of operators
    # definition of methods
    # anything else
```

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
    - Special built-in fields: `__class__`, `__dict__`
    - Default operators: `__str__`, `__repr__`

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

A Simpler Example

```python
class Employee(object):

    """Instance is salaried worker
    INSTANCE ATTRIBUTES:
    _name:  full name [string]
    _start:  first year hired
    _salary: yearly wage [float][""

    class Executive(Employee):

        """An Employee with a bonus
        INSTANCE ATTRIBUTES:
        _bonus: annual bonus [float][""
```

Kivy Example

```python
import kivy

from kivy.app import App
from kivy.uix.widget import Widget
from kivy.uix.label import Label
from kivy.uix.button import Button

class Module(Widget):
    pass

class Class(Widget):
    pass

class Object(Widget):
    pass

class KivyExample(App):
    def build(self):
        return Module()

if __name__ == '__main__':
    KivyExample().run()
```
Method Overriding

• Which \texttt{str} do we use?
  * Start at bottom class folder
  * Find first method with name
  * Use that definition
• New method definitions \textbf{override} those of parent
• Also applies to
  * Initializers
  * Operators
  * Properties

### 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 \texttt{super()}
  * “Converts” type to parent class
  * Now methods go to the class

#### Example:

```python
super().__str__()
```

In Python 2

```python
self goes here
```

### About \texttt{super()}

• \texttt{super()} is very limited
  * Can only go one level
  * BAD: \texttt{super(),super()}

#### Need arguments for more

* \texttt{super(class,self)}

### Primary Application: Initializers

```python
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
```

#### Also Works With Class Attributes

Class Attribute: Assigned outside of any method definition

```python
class Employee(object):
    """An Employee with a salary""
    STD_SALARY = 50000.0

    def \_\_init\_\_\_(self,n,d,s=STD_SALARY):
        self._name = n
        self._start = d
        self._salary = s

class Executive(Employee):
    """An Employee with a bonus""
    STD_BONUS = 10000.0

    def \_\_init\_\_\_(self,n,d,b=STD_BONUS):
        super().\_\_init\_\_\_(n,d)
        self._bonus = b
```

### Instance Attributes are (Often) Inherited

```python
class Employee(object):
    ...""
    def \_\_init\_\_\_(self,n,d,s=STD_SALARY):
        self._name = n
        self._start = d
        self._salary = s

class Executive(Employee):
    ...""
    def \_\_init\_\_\_(self,n,d,b=STD_BONUS):
        super().\_\_init\_\_\_(n,d)
        self._bonus = b
```