Lecture 19

Subclasses & Inheritance
Announcements for Today

Reading

- **Prelim, Nov 8th 5:15 or 7:30**
  - Same break-up as last time
  - But will swap times assigned
- **Material up to November 1**
  - Review posted this weekend
  - Recursion + Loops + Classes
- **Conflict with Prelim time?**
  - Prelim 2 Conflict on CMS
  - Submit by next Thursday
  - SDS students must submit!

Assignments

- A4 graded by end of week
  - Survey is still open
- A5 was posted Wednesday
  - Shorter written assignment
  - Due Wednesday at Midnight
- A6 was posted Saturday
  - Due a **week after** prelim
  - Designed to take two weeks
  - Finish Task 2 before exam

10/30/18
Announcements for Today

Reading

- Prelim, Nov 8th 5:15 or 7:30
  - Same break-up as last time
  - But will swap times assigned

- Material
  - Review
  - Recursion

- Conflict with Prelim time?
  - Prelim 2 Conflict on CMS
  - Submit by next Thursday
  - SDS students must submit!

Assignments

- A4 graded by end of week
  - Survey is still open

- A5 was posted Wednesday
  - Assignment ends at Midnight

- A6 was posted Saturday
  - Due a week after prelim
  - Designed to take two weeks
  - Finish Task 2 before exam

Optional Reading: Chapter 18

10/30/18 Subclasses & Inheritance
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)
```
Sharing Work

- These classes will have a lot in common
  - Drawing handles for selection
  - Background and foreground color
  - Current size and position
  - And more (see the formatting bar in PowerPoint)

- **Result**: A lot of repetitive code

- **Solution**: Create one class with shared code
  - All content are subclasses of the parent class
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): ...

Abbreviate as SC to right

10/30/18 Subclasses & Inheritance
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*
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__`
  - Special built-in methods: `__str__`, `__repr__`

### Kivy Example

- `object`
- `kivy.uix.widget.WidgetBase`
- `kivy.uix.widget.Widget`
- `kivy.uix.label.Label`
- `kivy.uix.button.Button`
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__
  § Special built-in methods: __str__, __repr__
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
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

```
TextBox(id3)
  p(text)
  'Hi!'  
SC(object)
  __init__(self,x,y,w,h)
  draw_frame(self)
  select(self)

TextBox(SC)
  __init__(self,x,y,text)
  draw(self)
```

```
p.select()
```
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

Often Called the **Bottom-Up Rule**.
Subclass *inherits* methods of parent.

```
__init__(self, x, y, text)
draw(self)
select(self)

SC(object)
__init__ (self, x, y, w, h)
draw_frame (self)
select (self)
```

```
p.text
p.draw()

TextBox(SC)
__init__ (self, x, y, text)
draw (self)
```

```
TextBox
id3
Text Box
p
id3

p

'text

'Hi!'```

10/30/18

Submission & Inheritance
class Employee(object):

    """Instance is salaried worker

    INSTANCE ATTRIBUTES:
    _name: full name [string]
    _start: first year hired
        [int ≥ -1, -1 if unknown]
    _salary: yearly wage [float]"

class Executive(Employee):

    """An Employee with a bonus

    INSTANCE ATTRIBUTES:
    _bonus: annual bonus [float]"

object

Employee

Executive
A Simpler Example

```python
class Employee(object):
    """Instance is salaried worker

    INSTANCE ATTRIBUTES:
    _name:  full name [string]
    _start: first year hired
            [int ≥ -1, -1 if unknown]
    _salary: yearly wage [float]"
```

```python
class Executive(Employee):
    """An Employee with a bonus

    INSTANCE ATTRIBUTES:
    _bonus: annual bonus [float]"
```
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__`
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__()
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:**
  
  ```python
  super().__str__(self)
  ```

```python
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))
```

`self` is implied
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

```python
p.id2
```

```
id2
Exec
  __str__()
  p.__str__()
```

```
Exec
  __str__()
Empl
  __str__()
object
  __str__()
```
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

```
p.id2
```

```
p.__str__()
Exec.__str__()
super(Exec, self).__str__()
Empl.__str__()
object.__str__()
```
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

```
p.id2

id2
Exec

<table>
<thead>
<tr>
<th>Exec</th>
<th><strong>str</strong>( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>p.<strong>str</strong>( )</td>
<td>super(Exec,self).<strong>str</strong>( )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Empl</th>
<th><strong>str</strong>( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>super().<strong>str</strong>( )</td>
<td>super(Empl,self).<strong>str</strong>( )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object</th>
<th><strong>str</strong>( )</th>
</tr>
</thead>
</table>

```
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
Instance Attributes are (Often) Inherited

```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
```

10/30/18

Subclasses & Inheritance
Also Works With Class Attributes

**Class Attribute**: Assigned outside of any method definition

```python
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
```
class A(object):
    x = 3 # Class Attribute
    y = 5 # Class Attribute
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    y = 4 # Class Attribute
    z = 42 # Class Attribute
    def g(self):
        return 14
    def h(self):
        return 18

• Execute the following:
  >>> a = A()
  >>> b = B()

• What is value of a.f()?

A: 10
B: 14
C: 5
D: ERROR
E: I don’t know
Name Resolution and Inheritance

class A(object):
    x = 3  # Class Attribute
    y = 5  # Class Attribute

def f(self):
    return self.g()
def g(self):
    return 10

class B(A):
    y = 4  # Class Attribute
    z = 42  # Class Attribute

def g(self):
    return 14
def h(self):
    return 18

• Execute the following:
    >>> a = A()
    >>> b = B()

• What is value of a.f()?

A: 10  CORRECT
B: 14
C: 5
D: ERROR
E: I don’t know
**Name Resolution and Inheritance**

```python
class A(object):
    x = 3  # Class Attribute
    y = 5  # Class Attribute

def f(self):
    return self.g()

def g(self):
    return 10

class B(A):
    y = 4  # Class Attribute
    z = 42 # Class Attribute

def g(self):
    return 14

def h(self):
    return 18
```

- Execute the following:
  >>> a = A()
  >>> b = B()

- What is value of `b.f()`?

  **Choices:**
  A: 10  
  B: 14  
  C: 5   
  D: ERROR  
  E: I don’t know
Name Resolution and Inheritance

class A(object):
    x = 3 # Class Attribute
    y = 5 # Class Attribute
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    y = 4 # Class Attribute
    z = 42 # Class Attribute
    def g(self):
        return 14
    def h(self):
        return 18

• Execute the following:
  >>> a = A()
  >>> b = B()

• What is value of b.f()?

A: 10
B: 14 CORRECT
C: 5
D: ERROR
E: I don’t know
Name Resolution and Inheritance

class A(object):
    x = 3  # Class Attribute
    y = 5  # Class Attribute

def f(self):
    return self.g()

def g(self):
    return 10

class B(A):
    y = 4  # Class Attribute
    z = 42  # Class Attribute

def g(self):
    return 14

def h(self):
    return 18

• Execute the following:
  >>> a = A()
  >>> b = B()

• What is value of b.x?

A: 4  
B: 3  
C: 42 
D: ERROR
E: I don’t know
Name Resolution and Inheritance

class A(object):
    x = 3  # Class Attribute
    y = 5  # Class Attribute

def f(self):
    return self.g()

def g(self):
    return 10

class B(A):
    y = 4  # Class Attribute
    z = 42  # Class Attribute

def g(self):
    return 14

def h(self):
    return 18

- Execute the following:
  >>> a = A()
  >>> b = B()

- What is value of b.x?

  A: 4
  B: 3  CORRECT
  C: 42
  D: ERROR
  E: I don’t know
Name Resolution and Inheritance

```python
class A(object):
    x = 3  # Class Attribute
    y = 5  # Class Attribute

def f(self):
    return self.g()

def g(self):
    return 10

class B(A):
    y = 4  # Class Attribute
    z = 42  # Class Attribute

def g(self):
    return 14

def h(self):
    return 18
```

- Execute the following:
  ```python
  >>> a = A()
  >>> b = B()
  ```
- What is value of `a.z`?

  - A: 4
  - B: 3
  - C: 42
  - D: ERROR
  - E: I don’t know
Name Resolution and Inheritance

```python
class A(object):
    x = 3  # Class Attribute
    y = 5  # Class Attribute

def f(self):
    return self.g()

def g(self):
    return 10

class B(A):
    y = 4  # Class Attribute
    z = 42  # Class Attribute

def g(self):
    return 14

def h(self):
    return 18
```

- Execute the following:
  ```python
  >>> a = A()
  >>> b = B()
  ```

- What is value of `a.z`?

  A: 4
  B: 3
  C: 42
  D: ERROR  CORRECT
  E: I don’t know