Lecture 20

Subclasses & Inheritance
# Announcements for Today

## Reading
- Today: Chapter 18
- Online reading for Thursday

## Assignments
- **Prelim, Nov 9th 7:30-9:00**
  - Material up to Thursday
  - Review posted on Thursday
  - Recursion + Loops + Classes
- **S/U Students are exempt**
- **Conflict with Prelim time?**
  - Prelim 2 Conflict on CMS
  - Submit by Thursday
- **A4 graded by end of week**
  - Survey is still open
- **A5 was posted Thursday**
  - Shorter written assignment
  - Due Thursday at Midnight
- **A6 was posted Saturday**
  - Due a **week after** prelim
  - Designed to take two weeks
  - Finish Task 3 before exam
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
Defining 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): ...
```

Abbreviate as SC to right

Superclass Parent class Base class
SlideContent

Subclass Child class Derived class
TextBox
Image

Subclasses & Inheritance
Class Definition: Revisited

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

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

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

Class type to extend (may need module name)
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__`
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__

Kivy Example

```
object
kivy.uix.wide.Widget
kivy.uix.widget.Widget
kivy.uix.label.Label
kivy.uix.button.Button
```

10/31/17 Subclasses & Inheritance 8
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
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
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.
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]"""
class Employee(object):

    """Instance is salaried worker

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

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

- Also applies to
  - Initializers
  - Operators
  - Properties
  
  
  ```
  object
  __init__ (self)
  __str__ (self)
  __eq__ (self)
  
  Employee
  __init__ (self, n, d, s)
  __str__ (self)
  __eq__ (self)
  
  Executive
  __init__ (self, n, d, b)
  __str__ (self)
  __eq__ (self)
  ```
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__()
  ```

  **In Python 2**

  ```python
  self goes here
  ```

  **Employee**

  ```
  __init__(self,n,d,s)
  __str__(self)
  __eq__(self)
  ```

  **Executive**

  ```
  __init__(self,n,d,b)
  __str__(self)
  __eq__(self)
  ```

  **object**

  ```
  __init__(self)
  __str__(self)
  __eq__(self)
  ```
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() (self is implied)
  ▪ “Converts” type to parent class
  ▪ Now methods go to the class

• Example:
  super().__str__()
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
```

10/31/17
Subclasses & Inheritance 18
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
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.f()`?
  - A: 10
  - 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:
  ```python
  >>> a = A()
  >>> b = B()
  ```

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

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

```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 b.x?

  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 \( b.x \)?

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

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