Lecture 20

Subclasses & Inheritance
Announcements for Today

Assignments

• A4 graded by end of week
  ▪ Survey is still open
• A5 was posted Thursday
  ▪ Shorter written assignment
  ▪ Due Thursday at Midnight
• A6 was posted Monday
  ▪ Due 3 days before prelim
  ▪ Designed to take two weeks
  ▪ Follow micro-deadlines!

Video Lessons

• Lesson 22 for today
• Videos 23.1-23.7 next time
• Video 23.8 is very optional
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)
    ```
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

SC

Abbreviate as SC to right

__init__(self,x,y,w,h)
draw_frame(self)
select(self)

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

__init__(self,x,y,img_f)
draw(self)
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
```

---

Module

Class
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

```python
built-in class
```

```python
object
```

```python
kivy.uix.widget.Widget
```

```python
kivy.uix.label.Label
```

```python
kivy.uix.button.Button
```

- Module
- Class
- Super class
- Super super class
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(SC)
__init__(self,x,y,text)
draw(self)
```

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

```
object
```

```
TextBox

11/1/22
```

```
Subclasses & Inheritance
```

```
11/1/22
```
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

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

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

p
  text 'Hi!'
  id3

p.select()
p.text
p.draw()
```
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 the object

Subclasses & Inheritance

Often Called the Bottom-Up Rule. Subclass inherits methods of parent.
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
A Simpler Example

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

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

All double underscore methods are in class object.
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\_\_

```python
object
\_\_init\_(self)
\_\_str\_(self)
\_\_repr\_(self)

Employee
\_\_init\_(self,n,d,s)
\_\_str\_(self)
\_\_repr\_(self)

Executive
\_\_init\_(self,n,d,b)
\_\_str\_(self)
\_\_repr\_(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

self goes here
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
  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
What is `super()`?

- Like object with class “removed”
  - Still gives a reference to `self`
  - But bottom-up rule skips first class
- But cannot use it everywhere
  - Only allowed in method definitions

```
super() id3
  text 'Hi!'
```

```
object
  ????
```

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

```
SlideContent
  p.draw()
  p.select()
```

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

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

```
object.__str__()
```

```
Empl.__str__()
```

```
Exec.__str__()
```

```
p.__str__()
```
What is `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

p.__str__()
What is `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

```
11/1/22
Subclasses & Inheritance
```
What is `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

```
11/1/22
Subclasses & Inheritance
```

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

- **Employee**
  - _name: Fred
  - _start: 2012
  - _salary: 50000.0

- **Executive**
  - _name: Fred
  - _start: 2012
  - _salary: 50000.0
  - _bonus: 0.0
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

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:
  ```python
  >>> 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

```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.f()`?
  
  | A: 10 | B: 14  **CORRECT** |
  | C: 5  |
  | D: **ERROR** |
  | E: I don’t know |

11/1/22 Subclasses & Inheritance 28
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

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:
  ```
  >>> 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