CS 1110:
Introduction to Computing Using Python

Lecture 19

Subclasses & Inheritance

[Andersen, Gries, Lee, Marschner, Van Loan, White]
Announcements

• Regrade Requests
  • Please put on the table
Announcements

• A3 solutions will be released soon
• A4 will be released by Wednesday morning
  ▪ due Thursday, April 20th, 11:59pm
• Prelim 2
  ▪ Tuesday, April 25th, 7:30-9:00pm
  ▪ Please go to the same room you went for Prelim 1
  ▪ Conflict arrangements being worked out; stay tuned
• Lab 10 is out
Review of Attributes and Variables
Goal: Make something like Powerpoint

Image

Text box with text
Goal: Make something like Powerpoint

Image:
- x, y
- width, height
- isSelected
- pixels[][]
- __init__()
- draw()
- select()

TextBox:
- x, y
- width, height
- isSelected
- text
- __init__()
- draw()
- select()
Ideally…

Image:
- pixels[][]
- __init__()
- draw()

TextBox:
- text
- __init__()
- draw()

SlideContent:
- x, y
- width, height
- isSelected
- __init__() (inherited)
- select() (inherited)

“inherits”
Sharing Work

- **Solution**: Create a *parent* class with shared code
  - Then, create *subclasses* of the *parent* class
class SlideContent(object):
    """Any object on a slide."""
    def __init__(self, x, y, w, h): …
    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): …
Extending Classes

class <name>(<superclass>):

"""Class specification"""

initializer (\_\_init\_\_)

methods

class variables

anything else

Class to extend (may need module name)

So far, classes have extended \texttt{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
  ▪ Default operators: __str__, __repr__

Example

object
SlideContent
TextBox
InputTextBox
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
  - Default operators: `__str__`, `__repr__`

Example

```
object
SlideContent
TextBox
InputTextBox
```

built-in class
Super super class
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

```python
TextBox.__init__(self, x, y, text)
TextBox.draw(self)
TextBox.select(self)
```
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:
  3. Look in the superclass
  4. Repeat 3. until reach
     object

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

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 ≥ -1, -1 if unknown]
    _salary: yearly wage [float]"

class Executive(Employee):
    
    """An Employee with a bonus
    INSTANCE ATTRIBUTES:
    _bonus: annual bonus [float]"

double underscore methods are in class object
>>> e = Executive(“Megan”, 2009, 10000.0)
>>> print e

• Which \_
\_str\_
\_ do we use?
  ▪ Start at bottom class folder
  ▪ Find first method with name
  ▪ Use that definition

• New method definitions \texttt{override} those of parent
class A(object):
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    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

• 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
```

```python
class A(object):
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18
```
Name Resolution and Inheritance

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

• What is value of \(b.f()\)?

```python
class A(object):
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18
```

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

```
class A(object):
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    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 |
Accessing the “Original” 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
- Call old method explicitly
  - Use method as a function
  - Pass object as first argument
- **Example:**
  Employee.__str__(self)
Accessing the “Original” 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

• Call old method explicitly
  ▪ Use method as a function
  ▪ Pass object as first argument

• Example:
  Employee.__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 (Employee.__str__(self)
                + ', bonus ' + str(self._bonus))
```
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):
        Employee.__init__(self, n, d)
        self._bonus = b
Object Attributes can be 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):
        Employee.__init__(self, n, d)
        self._bonus = b

4/11/17 Subclasses & Inheritance 25
Also Works With Class Variables

**Class Variable:** 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
```
Name Resolution and Inheritance

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

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

- What is value of `b.x`?

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

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

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 Variable
    y = 5  # Class Variable

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

def g(self):
    return 10

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

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

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

**class** B(A):
    y = 4  # Class Variable
    z = 42  # Class Variable
    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
Mixed Number Example