Announcements for Today

Reading
- Today: Chapter 18
- Online reading for Thursday

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 Saturday
- Due a week after prelim
- Designed to take two weeks
- Finish Task 3 before exam

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

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
```
class <name>(<superclass>):
    """Class specification""
    getters and setters
    initializer (__init__)
    definition of operators
definition of methods
anything else
```

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

Kivy Example
```
object
  object
    kivy.uix.widget.Widget
    kivy.uix.button.Button
    kivy.uix.label.Label
    kivy.uix.button.Button
kivy.uix.widget.Module
kivy.uix.button.Button
kivy.uix.textinput
```
**A Simpler Example**

```python
class Employee(object):
    """An Employee with a salary"""
    __slots__ = '__init__(), __str__(), __eq__(),
    __name__: name, __start__: start,
    __salary__: salary, __bonus__: bonus,

    def __init__(self, name, start, salary, bonus):
        self.name = name
        self.start = start
        self.salary = salary
        self.bonus = bonus

    def __str__(self):
        return '{}: {} salary: {} bonus: {}'.format(self.name, self.start, self.salary, self.bonus)

    def __eq__(self, other):
        return self.name == other.name

class Executive(Employee):
    """An Employee with a bonus"""
    STD_BONUS = 10000.0
    def __init__(self, name, start, salary, bonus=STD_BONUS):
        super().__init__(name, start, salary, bonus)

    def __str__(self):
        return super().__str__() + ' Annual Bonus: ' + str(self.bonus)

    def __eq__(self, other):
        return super().__eq__(other)

# Creating an Employee
employee = Employee('Fred', 2012, 50000.0)
print(employee)

# Creating an Executive
executive = Executive('Fred', 2012, 50000.0)
print(executive)
```

**Method Overriding**

- Which `__str__` do we use?
  - Start at bottom class folder
  - Find first method with name
  - Use that definition
- New method definitions
  - `overide` 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 `super()`
    - “Converts” type to parent class
    - New methods go to the class
- Example:
  ```python
  super().__str__()
  ``
  ```Python
  In Python 2
  self goes here.
  ```

**Primary Application: Initializers**

```python
class Employee(object):
    """An Employee with a salary"""
    __slots__ = '__init__(), __str__(), __eq__(),
    __name__: name, __start__: start,
    __salary__: salary, __bonus__: bonus,

    def __init__(self, name, start, salary, bonus=STD_BONUS):
        super().__init__(name, start, salary, bonus)

    def __str__(self):
        return super().__str__() + ' Annual Bonus: ' + str(self.bonus)

    def __eq__(self, other):
        return super().__eq__(other)

# Creating an Employee
employee = Employee('Fred', 2012, 50000.0)
print(employee)

# Creating an Executive
executive = Executive('Fred', 2012, 50000.0)
print(executive)
```

**Instance Attributes are (Often) Inherited**

```python
class Employee(object):
    """An Employee with a salary"""
    __slots__ = '__init__(), __str__(), __eq__(),
    __name__: name, __start__: start,
    __salary__: salary, __bonus__: bonus,

    def __init__(self, name, start, salary, bonus=STD_BONUS):
        super().__init__(name, start, salary, bonus)

    def __str__(self):
        return super().__str__() + ' Annual Bonus: ' + str(self.bonus)

    def __eq__(self, other):
        return super().__eq__(other)

# Creating an Employee
employee = Employee('Fred', 2012, 50000.0)
print(employee)

# Creating an Executive
executive = Executive('Fred', 2012, 50000.0)
print(executive)
```

**Also Works With Class Attributes**

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

    def __init__(self, name, start, salary, bonus=STD_BONUS):
        super().__init__(name, start, salary, bonus)

    def __str__(self):
        return super().__str__() + ' Annual Bonus: ' + str(self.bonus)

    def __eq__(self, other):
        return super().__eq__(other)

# Creating an Employee
employee = Employee('Fred', 2012, 50000.0)
print(employee)

# Creating an Executive
executive = Executive('Fred', 2012, 50000.0)
print(executive)
```