LECTURE 4: OBJECT-ORIENTED PROGRAMMING, MAGIC METHODS

Introduction to Scientific Python, CME 193

Jan. 30, 2014

Download exercises from:

`web.stanford.edu/~ermartin/Teaching/CME193-Winter15/lectures.html`

Eileen Martin

Feedback Reminder

- Course evaluations only help future classes
- If you want something changed, say so!
  - Talk to me
  - Email
  - Anonymous online survey:

    https://www.surveymonkey.com/s/NSVJDDJ

Overview Today

- Introduction to classes, objects
- Magic methods
- More realistic example
- Looking at the `unittest` module
- Inheritance
- Discussing assignment 3

Classes

- Python has a few useful data structures that each have some methods defined for them (e.g. appending to a list)
- *Classes* are the way we can define our own data structures
- Each instance of a class is an *object*
- Classes have *attributes* that are described as:
  - data (values associated with that class)
  - methods (functions that objects of that class can access)
- Example: student class for a course scheduling program
  - What data should each student have?
  - What methods should each student have?

Open myClass.py and test_myClass.py

```
class emptyClass:
      '''Defines a class that has no attributes'''
      pass
```

emptyClass definition

```
from myClass import emptyClass

# instantiate an object of type emptyClass
someObject = emptyClass()

# get information about emptyClass
print(emptyClass.__doc__) # get information directly from class
print(someObject.__doc__) # get information from object

# look at this instantiation of an emptyClass object
print(someObject)
```

instantiating an emptyClass object

Overview Today

- Introduction to classes, objects
- Magic methods
- More realistic example
- Looking at the `unittest` module
- Inheritance
- Discussing assignment 3

Magic methods

- Magic methods may have default behaviors, but can be overridden in your classes
- A few examples we’ve seen:

  `__init__`

  `__str__`

  `__add__` (example `‘mystring’+’another’`)

  `__float__` (example `float(True)`)

More info: http://www.rafekettler.com/magicmethods.html

More magic methods examples:

__add__	__sub__	__mul__	__div__
add a+b	subtract a-b	multiply a*b	divide a/b
__radd__	__rsub__	__ror__	__rand__
add b+a (reverse order)	subtract b-a (reverse order)	b | a (reverse order)	b & a (reverse order)
__iadd__	__xor__	__or__	__and__
add to self a += b	a ^ b (exclusive or)	a | b	a & b
__int__	__str__	__nonzero__	__getitem__
typecast to integer int(a)	represent as string str(a)	defines boolean type cast bool(a)	returns value at a[key]
__contains__	__iter__ and next	__cmp__	__del__
x in a	allows iteration over object	compares/orders objects	destructor (destroy the object)

More info: http://www.rafekettler.com/magicmethods.html

Open myClass2.py and test_myClass2.py

```
class emptyClass2:
        '''Defines a class that has no attributes'''
        pass
        def __str__(self):
                return “I am an empty object”
```

emptyClass2 definition

```
from myClass2 import emptyClass2

# instantiate an object of type emptyClass2
someObject = emptyClass2()

# look at this instantiation of an emptyClass2 object
print(someObject)
```

now you should get more than an address from print()

Overview Today

- Introduction to classes, objects
- Magic methods
- More realistic example
- Looking at the `unittest` module
- Inheritance
- Discussing assignment 3

Features in this example

- We can define the instantiation (initialization) of a class with `__init__` (there can only be one init for a class)
- The parameter `self` is used to refer to the object itself

```
      self.inventory = someDictionary
```

- This example has methods that:
  - modify the object but don't return anything
  - have a return value
- Users of a class can modify attributed in the class's methods or in any outside script

Open warehouseClass.py and test_warehouse.py

```
class warehouse:
          '''This class describes a warehouse with some inventory dictionary with
item:number pairs, location, and a set of employees’’’
          n_warehouses = 0              # shared class variable
          def __init__(self, inventory, location, employees):
                    self.inventory = inventory  # Personal variables for just
                    self.location = location       # one instance of class
                    self.employees = employees
                    warehouse.n_warehouses += 1 # Any warehouse can modify this

          def __str__(self):
                    …
          def hire(self, employee):
                    …
          def fire(self, employee):
                    …
          def newItem(self, item, number=1):
                    …
          def soldItem(self, item, number=1):
                    …
          def numItems(self):
                    …
```

Overview Today

- Introduction to classes, objects
- Magic methods
- More realistic example
- Looking at the `unittest` module
- Inheritance
- Discussing assignment 3

Open betTest2.py

What is happening when we use this module?

```
from betting import *
import unittest

class expectation(unittest.TestCase):
   def positiveCheck(self):
      '''check exp. value for positive prob. and winnings'''
      self.assertAlmostEqual(expValue(0.1,100),-10.0,11)
   def negativeCheck(self):
      '''check exp. value for prob > 0, winnings < 0'''
      self.assertAlmostEqual(expValue(0.1,-1.5),-0.15)
```

……more on next few slides…

Open betTest2.py

This first chunk:

- imports the `unittest` module (which has the `TestCase` class defined in it)
- defines the class `expectation` which inherits from `unittest.TestCase`
- defines two methods `positiveCheck` and `negativeCheck` that are specific to the expectation class
- and each of those methods calls the method `assertAlmostEqual`, which is inherited from `TestCase`

```
from betting import *
import unittest

class expectation(unittest.TestCase):
   def positiveCheck(self):
      '''check exp. value for positive prob. and winnings'''
      self.assertAlmostEqual(expValue(0.1,100),10.0,11)
   def negativeCheck(self):
      '''check exp. value for prob > 0, winnings < 0'''
      self.assertAlmostEqual(expValue(0.1,-1.5),-0.15)
```

Open betTest2.py

This next class:

- defines the class `decision` which inherits from `unittest.TestCase`
- defines 1 method `simpleCheck` that’s specific to the expectation class
- and that method calls the method `assertEqual`, which is inherited from `TestCase`

```
class decision(unittest.TestCase):
    def simpleCheck(self):
        '''check that you don't bet when cost is too big'''
        self.assertEqual(False, bet(0.1,100,11))
```

Open betTest2.py

This next chunk:

- defines the function `bettingSuiteFct`
- that function instantiates a `TestSuite` object called `bettingSuite`
- `TestSuite`’s method `addTest` is called on callable functions in classes
- `expectation` and `decision` inherited the `__call__` method for their newly-defined methods from `TestCase`
- a `TestSuite` object is returned which can run three tests

```
def bettingSuiteFct():
    bettingSuite = unittest.TestSuite()
    bettingSuite.addTest(expectation('positiveCheck'))
    bettingSuite.addTest(expectation('negativeCheck'))
    bettingSuite.addTest(decision('simpleCheck'))
    return bettingSuite
```

Open betTest2.py

This last chunk:

- Checks that we’re in the main script
- Creates an instance of a `TextTestRunner` object called `runner`
- Call’s the `TextTestRunner` class’ method `run` which takes an argument that is the `TestSuite` object returned by the function `bettingSuiteFct()`

```
if __name__ == '__main__':
    runner = unittest.TextTestRunner()
    runner.run(bettingSuiteFct())
```

Overview Today

- Introduction to classes, objects
- Magic methods
- More realistic example
- Looking at the `unittest` module
- Inheritance
- Discussing assignment 3

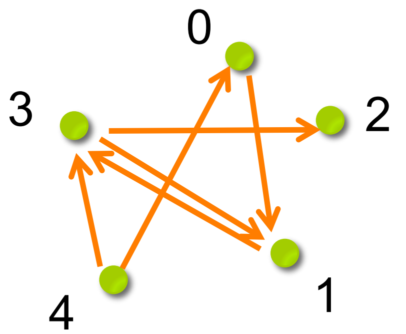
Inheritance

- What if you want to create a general class, and define more specific sub-classes?

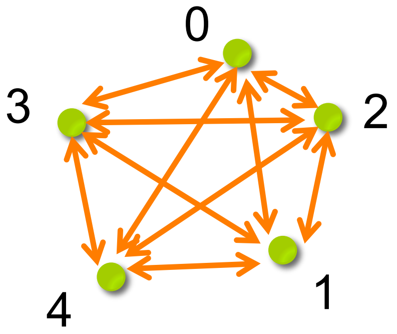
- Python allows classes to inherit from a parent class:
  - Get default attributes from parent class
  - Can override methods in parent class of the same name
  - Can define new methods that the parent class didn’t have

Inheritance example

- A graph is a set of vertices and set of edges (tuples)
- A complete graph is a graph which has an edge between every pair of vertices
- Open `graphClass.py` and `test_graph.py`
  - `graph` is parent class, `completeGraph` inherits from `graph`



an example of a graph



an example of a complete graph, which is also a graph

Overview Today

- Introduction to classes, objects
- Magic methods
- More realistic example
- Looking at the `unittest` module
- Inheritance
- Discussing assignment 3

Assignment 3

- Difficult questions, general questions?
- Assignment 4 posted on the course website: http://stanford.edu/~ermartin/Teaching/CME193-Winter15/assignments.html