Beyond Blocks (Python)

Object Oriented Programming

Beyond Blocks: Python: Session #2 by Glenn Sugden is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.



Tonight's Plan

- Quick review
- Finish dictionaries
- What is OOP?
- OOP terms
- Python vs. other languages
 - (e.g., Java)
- OOP syntax
- Examples



What is Object Oriented Programming?

Object-oriented programming (OOP) is a programming paradigm using "objects" data structures consisting of data fields and methods together with their interactions... Programming techniques may include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance.

en.wikipedia.org - Object oriented programming



OOP Terms

- Class
- Instance
- Instance variables
- Instance methods

- Class variables
- Class methods
- Namespaces
- Inheritance



OOP Terms Class

- The template for an object
- Classes creates a type
- Analogy: recipe for a cake



OOP Terms Instance

- A variable created (instantiated) from the class definition
- Shares the same attributes as another instance from the same class
- Analogy: the cakes made from the cake recipe



OOP Terms Instance Variables

- Variables that are bound to an instance
- Just like global variables are bound to a global scope
- Analogy: The ingredients for a cake



OOP Terms Instance Methods

- Functions that are bound to an instance
- First parameter is always "self"
- Analogy: The actions required to bake that cake



OOP Terms Class Variables

- Variables that are bound to the class itself
- Instances of that class share the same variable
- Analogy: "Who wrote the recipe?"



OOP Terms Class Methods

- For completeness, but...
- In Python, these are a little messy
- Most methods are called with an instance (object).

code.activestate.com - 52304-static-methods-aka-class-methods-in-python



OOP Terms Namespaces

- Mapping from names to objects
- You've seen module namespaces
- e.g., Math.sin(x)
- Classes provide namespaces too
- Namespaces are a way to organize scope.
- Dot referencing (e.g., class.variable)



П

OOP Terms Inheritance

- The OOP way to reuse code
- Base new classes on the attributes and behaviors of previously defined classes."
- AKA child, derived, or sub-classes.
- Analogy: Specialized types of cakes based on the common cake recipe



OOP in Python Comparison

- Highly dynamic
 - Can add variables and methods at runtime
 - Don't have to declare instances first
 - Static vs. Weak/Dynamic typing
- Default global scope
 - Most OOP languages default to "private"
- Slower than compiled languages
 - e.g., Java, C++...



OOP in Python Class Syntax

```
class ClassName:
pass
```

>>> instance = ClassName()



OOP in Python Method Syntax

```
>>> class ClassName:
    def method( self ):
        pass

>>> instance = ClassName()
>>> instance.method()
```



OOP in Python Method Syntax

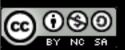
```
>>> class ClassName:
    def method( self ):
        pass

>>> instance = ClassName()
>>> instance.method()
```



OOP in Python Class Variables Syntax

```
class ClassName:
>>>
        classvariable = 0
>>>
        def func( self, n ):
>>>
          ClassName.classvariable = n
>>>
      instance = ClassName()
>>>
      instance.classvariable == 0
      instance.func(5)
       instance.classvariable == 5
```



OOP in Python Class Variables Syntax

```
class ClassName:
>>>
        classvariable = 0
>>>
        def func( self, n ):
>>>
          ClassName.classvariable = n
>>>
      instance = ClassName()
>>>
      instance.classvariable == 0
      instance.func(5)
      instance.classvariable == 5
```



OOP in Python Class Variables Syntax

```
instanceA = ClassName()
instanceA.classvariable == 0
instanceA.func(5)
instanceA.classvariable == 5
instanceB = ClassName()
instanceB.classvariable == 5
instanceB.func(10)
instanceB.classvariable == 10
```



OOP in Python "Docstring" Syntax

Help on class ClassName in module __main__:
class ClassName
 This is my class

@080

000

20

OOP in Python "Docstring" Syntax

```
>>> class ClassName:
    """ This is my class """
    pass
>>> help(ClassName)
```



OOP in Python init Method Syntax

```
>>> class ClassName:
    def __init__( self ):
        print "I'm init'ed! Weeee!"
>>> instance = ClassName()
        I'm init'ed! Weeee!
```



OOP in Python init Method Syntax

```
>>> class ClassName:
    def __init__( self ):
        print "I'm init'ed! Weeee!"
>>> instance = ClassName()
        I'm init'ed! Weeee!
```



OOP in Python init Method Syntax

```
>>> class ClassName:
    def __init__( self ):
        print "I'm init'ed! Weeee!"
>>> instance = ClassName()
        I'm init'ed! Weeee!
```



OOP Terms init Method

- Called as soon as you instantiate an instance / object
- First parameter is self
 - No different from other methods
- Often called a "constructor"
 - But different from other languages in Python the object is already constructed by the time __init__ is called!



```
class ClassName:
 def __init__( self ):
   print "I'm init'ed! Weeee!"
   self.localVar = 13
instance = ClassName()
I'm initialized! Weeeee!
print instance.localVar
13
```



```
class ClassName:
 def __init__( self ):
   print "I'm init'ed! Weeee!"
   self.localVar = 13
instance = ClassName()
I'm initialized! Weeeee!
print instance.localVar
13
```



```
instanceA = ClassName()
>>>
       instanceB = ClassName()
>>>
       print instanceA.localVar
       13
       instanceB.localVar = 42
       print instanceA.localVar
       13
       print instanceB.localVar
       42
```

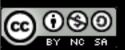


```
instanceA = ClassName()
>>>
       instanceB = ClassName()
>>>
       print instanceA.localVar
       13
       instanceB.localVar = 42
       print instanceA.localVar
       13
       print instanceB.localVar
       42
```



OOP in Python

Add Instance Variables "on the fly!"



OOP in Python

Add Instance Variables "on the fly!"



OOP in Python Even add methods!

```
class ClassName:
>>>
        pass
      instance = ClassName()
>>>
      def sayBlah():
>>>
        print "blah!"
      instance.sayIt = sayBlah
>>>
      instance.sayIt()
       blah!
```



OOP in Python Even add methods!

```
class ClassName:
>>>
        pass
      instance = ClassName()
>>>
      def sayBlah():
>>>
        print "blah!"
      instance.sayIt = sayBlah
>>>
      instance.sayIt()
       blah!
```



OOP in Python Even add methods!

```
class ClassName:
>>>
        pass
      instance = ClassName()
>>>
      def sayBlah():
>>>
        print "blah!"
       instance.sayIt = sayBlah
>>>
      instance.sayIt()
>>>
       blah!
```



OOP in Python "getters" and "setters" Syntax

```
class ClassName:
>>>
        def __init__( self ):
          self.localVar = 13
        def getLocalVar(self):
          return self.localVar
        def setLocalVar(self, n):
          self.localVar = n
```



OOP in Python "getters" and "setters" Syntax

```
class ClassName:
>>>
        def __init__( self ):
          self.localVar = 13
        def getLocalVar(self):
          return self.localVar
        def setLocalVar(self, n):
          self.localVar = n
```



OOP in Python "getters" and "setters" Syntax

```
>>> instanceA = ClassName()
>>> instanceA.setLocalVar(5)
>>> print instanceA.getLocalVar()
5
```



OOP in Python "getters" and "setters," Why?

```
class ClassName:
 def setLocalVar(self, n):
   if (n>0):
     self.localVar = n
   else:
     print "n is too low!"
```



OOP in Python "getters" and "setters" Syntax

```
>>> instanceA = ClassName()
>>> instanceA.setLocalVar(5)
>>> print instanceA.getLocalVar()
5
>>> instanceA.setLocalVar(-5)
"n is too low!"
```



```
>>> class ClassName:
... def __init__( self, n=13 ):
... self.localVar = n
```



```
>>> class ClassName:
... def __init__( self, n=13 ):
... self.localVar = n
```



```
>>> instanceA = ClassName()
>>> print instanceA.getLocalVar()
13
>>> instanceB = ClassName(42)
>>> print instanceA.getLocalVar()
42
```



```
>>> instanceA = ClassName()
>>> print instanceA.getLocalVar()
13
>>> instanceB = ClassName(42)
>>> print instanceA.getLocalVar()
42
```



Default parameters must be last!

```
class ClassName:
>>>
        def __init__( self, n=13 ):
          self.localVar = n
      class ClassName:
>>>
        def \_init\_( self, n=13, m):
          self.localVar = n
          self.otherLocalVar = m
```



Default parameters must be last!

```
class ClassName:
>>>
        def __init__( self, n=13 ):
          self.localVar = n
      class ClassName:
>>>
        def __init__( self, n=13, m ):
          self.localVar = n
          self.otherLocalVar = m
```



Default parameters must be last!

```
class ClassName:
>>>
        def __init__( self, n=13 ):
          self.localVar = n
      class ClassName:
        der __init__( self, n=13, m ):
          self la Larvar
          self.otherLocalVar = m
```



OOP in Python Counter Class Example



Thursday, December 8, 11

OOP in Python More OOP Examples

>>> Found in BeyondBlocks3.py



OOP in Python Inheritance: Recall...

```
class Parent:
        def __init__(self,name):
          self.localName = name
        def who(self):
          print self.LocalName
       aParent = Parent("Me!")
>>>
       aParent.who()
      Me!
```



OOP in Python Inheritance: Subclass



OOP in Python Inheritance: Subclass



OOP in Python Inheritance : Overriding

- You've already seen it in action!
 - __init__(self)
- Variable and Methods
- Methods must match signature to override.
 - Function name and
 - Number of Function Parameters



OOP in Python Inheritance: Recall...

```
class Parent:
        def __init__(self,name):
          self.localName = name
        def who(self):
          print self.localName
       aParent = Parent("Me!")
>>>
       aParent.who()
      Me!
```



OOP in Python Inheritance: Now Override!

```
>>> class Child( Parent ):
    def who(self):
        print "Child."+self.localName
>>> aChild = Child("Me too!")
>>> aChild.who()
    Child.Me too!
```



OOP in Python Inheritance: Now Override!

```
>>> class Child( Parent ):
    def who(self):
        print "Child."+self.localName
>>> aChild = Child("Me too!")
>>> aChild.who()
    Child.Me too!
```



OOP in Python Inheritance: Override

```
class Parent:
        def setName(self,newName):
          self.localName=newName
      aParent = Parent("Me!")
>>>
      aParent.setName("No, me!")
      aParent.who()
      No, me!
```



OOP in Python Inheritance : Call Your Parents!

```
class Child( Parent ):
>>>
        def setName(self,name):
          Parent.setName(self,name)
          self.localName+="Renamed!"
      aChild = Child("Me too!")
>>>
      aChild.setName("No, me!")
>>>
      aChild.who()
      No, me! Renamed!
```



OOP in Python Inheritance : Call Your Parents!

```
class Child( Parent ):
>>>
        def setName(self,name):
          Parent.setName(self,name)
          self.localName+=" Renamed!"
      aChild = Child("Me too!")
>>>
      aChild.setName("No, me!")
>>>
      aChild.who()
      No, me! Renamed!
```

@ 080 BY NC SA

OOP in Python Inheritance : Call Your Parents!

```
class Child( Parent ):
>>>
        def setName(self,name):
          Parent.setName(self,name)
          self.localName+=" Renamed!"
      aChild = Child("Me too!")
>>>
      aChild.setName("No, me!")
>>>
      aChild.who()
      No, me! Renamed!
```



Inheritance: Override Variables

```
class Child( Parent ):
>>>
        def rename(self,name):
          self.localName="Child:"+name
      aChild = Child("Me too!")
>>>
      aChild.rename("No, me!")
>>>
      aChild.who()
      Child:No, me!
```



Inheritance: Override Variables

```
class Child( Parent ):
>>>
        def rename(self,name):
          self.localName="Child:"+name
      aChild = Child("Me too!")
>>>
      aChild.rename("No, me!")
>>>
      aChild.who()
      Child:No, me!
```



OOP in Python Inheritance: Init Your Parents!

```
class Child( Parent ):
>>>
        def __init__(self,name):
          Parent.__init__(self,name)
          self.localName+="is a child."
      aChild = Child("Glenn")
>>>
      aChild.who()
      Glenn is a child.
```



OOP in Python Inheritance: Init Your Parents!

```
class Child( Parent ):
>>>
        def __init__(self,name):
          Parent.__init__(self,name)
          self.localName+="is a child."
      aChild = Child("Glenn")
>>>
      aChild.who()
      Glenn is a child.
```



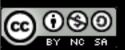
OOP in Python Inheritance : Account Example

>>> <demo>



OOP in Python Inheritance: More Examples

>>> Found in BeyondBlocks3.py



Thursday, December 8, 11 65

OOP in Python Inheritance: isinstance()

```
# isinstance(instance,ClassName)
isinstance(aChild,Child)
True
isinstance(aChild, Parent)
True
isinstance(aParent, Child)
False
```



OOP in Python Inheritance: issubclass()

```
# issubclass(SubClassName,ClassName
>>>
       issubclass(Child, Parent)
      True
       issubclass(Parent, Child)
       False
       issubclass(Parent, Parent)
       True
```

⊚ ③ ⑤ ⑤

Classes (and instances) are stored internally as dict()ionaries!



```
>>> print Parent.__dict__
{'__module__': '__main__',
'setName': <function setName at 0x46b330>,
'__str__': <function __str__ at 0x46b2f0>,
'__init__': <function __init__ at 0x46b2b0>,
'__doc__': None}
```



```
>>> print Parent.__dict__
{'__module__': '__main__',
'setName': <function setName at 0x46b330>,
'__str__': <function __str__ at 0x46b2f0>,
'__init__': <function __init__ at 0x46b2b0>,
'__doc__': None}
```



```
>>> print Parent.__dict__
{'__module__': '__main__',
'setName': <function setName at 0x46b330>,
'__str__': <function __str__ at 0x46b2f0>,
'__init__': <function __init__ at 0x46b2b0>,
'__doc__': None}
```





```
>>> print Child.__dict__
{'rename': <function rename at 0x46b370>,
   '__module__': '__main__',
   '__doc__': ' A Child class, derived from a
Parent class ',
   'classVariable': 'A kid.'}
```



```
>>> print Child.__dict__
{'rename': <function rename at 0x46b370>,
   '__module__': '__main__',
   '__doc__': ' A Child class, derived from a
Parent class ',
   'classVariable': 'A kid.'}
```



```
>>> print Child.__dict__
{'rename': <function rename at 0x46b370>,
   '__module__': '__main__',
   '__doc__': ' A Child class, derived from a
Parent class ',
   'classVariable': 'A kid.'}
```



Resources

- Introduction to OOP with Python
 - www.voidspace.org.uk/python/articles/OOP.shtml
- Python Tutorial : Classes
 - docs.python.org/tutorial/classes.html
- Object Oriented Programming With Python
 - www.devshed.com/c/a/Python/Object-Oriented-Programming-With-Python-part-1/
 - www.devshed.com/c/a/Python/ObjectOriented-Programming-With-Python-part-2/
- Building Skills in OOD
 - homepage.mac.com/s_lott/books/oodesign/build-python/ html/

