Python Classes

This tutorial seeks to remind us about basics, and some fine points, related to object oriented mechanisms in Python:

- instance and class attributes
- inheritance
- static and class methods
- properties

Instance and Class Attributes

In [1]:

```python
class A:
    x = "class attribute of class A"

a = A()
print("a.x": a.x)
```

```
a.x: class attribute of class A
```

In [2]:

```python
x = "global value x"

class B(A):
    x = "class attribute of class B"
    def __init__(self):
        self.x = x
        #self.x = "instance attribute of a B instance"

b = B()
print("b.x": b.x)
```

```
b.x: global value x
```

# What will be printed?
# 1 - b.x: class attribute of class A
# 2 - b.x: class attribute of class B
# 3 - b.x: global value x
# 4 - b.x: instance attribute of a B instance
# 5 - Other or error

In [3]:

```python
class A:
    x = "class attribute of class A"

class B(A):
    x = "class attribute of class B"
    def __init__(self):
        self.x = "instance attribute of a B instance"

class C(B):
    x = "class attribute of class C"
    def __init__(self):
        pass
        #super().__init__()
        #B.__init__(self)

c = C()
print("c.x": c.x)
```

```
c.x: class attribute of class C
```

# What will be printed?
# 1 - c.x: class attribute of class A
# 2 - c.x: class attribute of class B
# 3 - c.x: class attribute of class C
# 4 - c.x: instance attribute of a B instance
# 5 - Other or error
Some attribute accessors (fine points)

```python
In [5]:
   : get attrib(c,'x','default val')
   : get attrib(c,'y','default val')
   : c.x
   : C.x
   : type(c).x
```

```
Out[5]: 'class attribute of class C'
```

Method inheritance

Methods can be thought of as class (or instance) attributes that happen to be functions: they are resolved similarly, then called on their arguments. There are various special syntactic forms and protocols that govern instance creation, initialization, destruction, as well as method invocation syntax to make it convenient to pass along the instance object itself.

```python
In [6]:
   : class Bar():
   :     def __init__(self, val):
   :         self.x = val

   : class Foo(Bar):
   :     x = 100
   :     def increment(this):
   :         # conventionally 'self' rather than 'this' or other variable names
   :         this.x += 1

   : f = Foo(33)
   : print("f.x: ", f.x)
   : f.increment()
   : print("f.x: ", f.x)
```

```
f.x: 33
```

```
f.x: 34
```
Invoking a subclass method from a superclass:

```
In [8]: class Bar():
    ...:     def __init__(self, val):
    ...:         self.x = val
    ...:
    ...:     def double_increment(self):
    ...:         self.increment()
    ...:         self.increment()
    ...:         #type(self).increment(self)

class Foo(Bar):
    ...:     def increment(self):
    ...:         self.x += 1

class Gorp(Bar):
    ...:     delta = 100
    ...:     def increment(self):
    ...:         self.x += self.delta

f = Foo(0)
print("f.x:", f.x)
print("f.x:", f.x)
f = Foo(33)
print("f.x:", f.x)
f.increment()
print("f.x:", f.x)

f = Foo(33)
print("f.x:", f.x)
f.increment()  
print("f.x:", f.x)

f = Foo(0)
print("f.x:", f.x)
print("f.x:", f.x)
f.double_increment()  
print("f.x:", f.x)

f = Foo(0)
print("g.x:", g.x)
g.double_increment()  
print("g.x:", g.x)

f.x: 0
f.x: 2
f.x: 3
f.x: 5
f.x: 7
f.x: 9

Static Methods

Sometimes we don't want to pass (implicitly or explicitly) the object instance. For this we can use decorators to indicate this modified behavior.

@staticmethod -- omits the self argument
In [9]:

class Gorp(Bar):
    delta = 100

    def increment(self):
        self.x += self.delta

    def set_delta(self, d): #note -- conventional method, takes self argument
        Gorp.delta = d
        return Gorp.delta

    @staticmethod
    def set_del(d): #note -- staticmethod, does not take self argument
        Gorp.delta = d
        return Gorp.delta

g = Gorp(0)
print("g.set_delta(200):", g.set_delta(200))  # but feels wrong to change a class attribute through g. something...
print("g.set_del(300):", g.set_del(300))    # still feels wrong
print("Gorp.set_del(400):", Gorp.set_del(400))  # cleaner/clearer

g.set_delta(200): 200
Gorp.set_del(400): 400

@classmethod -- passes class of target rather than target

Sometimes we want to have the class of the object as the lead argument, not the object itself:

In [10]:
class Polygon:
    color = "white"
    @classmethod
    def shade(cls, color):
        cls.color = color

class Rectangle(Polygon):
    color = "green"

class Square(Rectangle):
    color = "blue"

print("Square.color: ", Square.color)
Square.shade("red")
print("Square.color: ", Square.color)
r1 = Rectangle()
r2 = Rectangle()
print("Rectangle.color: ", Rectangle.color)
print("r2.color: ", r1.color)
r1.shade("burnt orange")  # UGLY -- changes color of ALL Rectangles, not just r1
print("after r1.shade('burnt orange'), r2.color: ", r1.color)
Square.color: blue
Rectangle.color: green
r2.color: green
after r1.shade('burnt orange'), r2.color: burnt orange

Properties

Suppose we want a simple "object.x" syntax for getting or setting an attribute, but we want/need computation beyond just looking up or setting an instance variable? We have @property and @<var>.setter for this:
```python
In [11]:
class Bounded():
    """ Keep track of variable x, but clipped to xmin and xmax """
    xmin, xmax = 50, 100
    def __init__(self, x):
        self.__x = x  # __ means a private attribute not to be accessed outside class

    # maintain invariant when on variable get
    @property
def x(self):
        return max(min(self.xmax, self.__x), self.xmin)

a = Bounded(231)
print("a.x:", a.x)
# a.x = -33
```

```
a.x: 100
```

```
In [12]:
class Bounded():
    """ Keep track of variable x, but clipped to xmin and xmax """
    xmin, xmax = 50, 100
    def __init__(self, x):
        self.x = x  # calls x.setter

    @property
def x(self):
        return self.__x

    # maintain invariant on variable set
    @x.setter
def x(self, val):
        self.__x = max(min(self.xmax, val), self.xmin)

a = Bounded(231)
print("a.x:", a.x)
a.x = -33
print("a.x:", a.x)
```

```
a.x: 100
a.x: 50
```