11-Python Classes

Unknown Author

August 22, 2013

1 11 - Python Classes

You don't need to write your own classes to write useful programs in Python. But you should be able to use and understand classes that others have written.

One way to think about classes is that they are a pattern for keeping a set of variables and related functions organized. We can do similar things without classes. Consider the following:

In [1]:
```
def init_data_processing(filename):
    my_data = {'filename': filename,
               'processed': False}
    return my_data
```

Now, we can set up a new file to process using init.

In [2]:
```
data_analysis = init_data_processing('example.txt')
data_analysis
```

Out [2]:
```
{'filename': 'example.txt', 'processed': False}
```

In [3]:
```
def do_analysis(my_data):
    if not my_data['processed']:
        print 'Analysing file!'
        my_data['count'] = 0
        infile = open(my_data['filename'])
        for line in infile:
            my_data['count'] += 1
        infile.close()
        my_data['processed'] = True
    else:
        print 'I already counted this file!'

    return my_data['count']

do_analysis(data_analysis)
```

```
Analysing file!
```

Out [3]:
```
5
```

A class is a way to bundle related code and data (like the code above) into a single block. Note that instead of putting each bit of data in a dictionary location, we use the "dot" notation, and the object will automatically be passed in as the first parameter (which we should always call self to keep things clear).

In [4]:
```
# Initially, always put '(object)' next to your class name.
# (But only in python 2.x.) You'll understand why later.
class DataAnalysis(object):
    # We can initialize class data here
    processed = False

    # Or initialize data dynamically in an __init__ function
    def __init__(self, filename):
        self.filename = filename
        # Note - an __init__ should NEVER return a value

    # Note that all methods include a 'self' parameter
    # This parameter will point to the specific object you
    # call the method on
    def do_analysis(self):
        if not self.processed:
            print 'Analyzing file! (you crazy Brits...)'
            self.count = 0
            infile = open(self.filename)
            for line in infile:
                self.count += 1
            infile.close()
            self.processed = True
        else:
            print 'I already counted this file!'

        return self.count
```

We can initialize as many objects as we want from the class (basically a description for objects). Their data will remain separate. There are some more advanced methods for sharing data between all objects of the same class, but we won't cover them here.

In [5]:
```
# Parameters here get passed along to __init__
data_analysis_obj = DataAnalysis('example.txt')
```

In [6]:
```
# Now what's in there?
# Remember - focus on stuff without __underscores__
dir(data_analysis_obj)
```

Out [6]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__doc__',
 '__format__',
 '__getattribute__',
 '__hash__',
 '__init__',
 '__module__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
```

```
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'do_analysis',
 'filename',
 'processed']
```

Note that we don't pass `data_analysis_obj` in to `do_analysis()`. This happens automatically - the object is passed in as the first argument.

In [7]:
```
data_analysis_obj.do_analysis()
```

```
Analyzing file! (you crazy Brits...)
```

Out [7]:
```
5
```