EML4930/EML6934: Lecture 03

Objects, Namespace, Python libraries, and pip

Charles Jekel

September 14, 2017

Objects - everything in Python is an object

In this example x isn't really a string. It is an object containing a collection of functions

```
in : x = 'hello world'
```

```
in : dir(x)
```

dir(object) returns an alphabetized list of the attributes of the object

Object Oriented Programming - OOP

Object Oriented Programming
A programming paradigm in which code related to a particular *object* is grouped together.

- Class - the **definition** of an Object
- Object - an **instance** of a Class
- Everything in Python is an object
- Advantage - less duplicate code
- Disadvantage - performance issues with OOP

Objects and dogs...

Imagine you had to write a veterinarian program about dogs. What functions and attributes would all dogs have in common?

Use class to create a new form of object

Some reading:

`https://docs.python.org/3/tutorial/classes.html`

`http://anandology.com/python-practice-book/object_oriented_programming.html`

I like this idea of creating a bank account to explain how to use class.

Creating a bank account object

```python
class bank_account:
    def __init__(self, initial_balance=0.0):
        self.balance = initial_balance
john = bank_account()
```

- use class to create an object named bank_account
- def __init__() is an initialization function that is run automatically on a new instance
- you can add required and optional variables to the __init__() function
- self is the naming convention in python of objects own instance (it's self)
- self is the first variable in your functions of your object
- pass self to your object's functions if you need access to your object's attributes
- balance is an attribute of the object
- a new instance of the object is created by calling bank_account()

adding a withdrawn and deposit function

```
class bank_account:
    def __init__(self, initial_balance=0.0):
        self.balance = initial_balance

    def withdraw(self, ammount):
        self.balance -= ammount
        print('Your new account balance is', self.balance)

    def deposit(self, ammount):
        self.balance += ammount
        print('Your new account balance is', self.balance)
```

```
in : john = bank_account(100.0)
in : john.withdraw(2.77)
out: Your new account balance is 97.23
in : john.deposit(10.0)
out: Your new account balance is 107.23
```

Adding extra attribute - liability and loan

```
class bank_account:
    def __init__(self, initial_balance=0.0, initial_debt=0.0):
        self.balance = initial_balance; self.debt = initial_debt
    def withdraw(self, ammount):
        self.balance -= ammount; self.print_balance()
    def deposit(self, ammount):
        self.balance += ammount; self.print_balance()
    def get_loan(self, ammount):
        self.balance += ammount; self.debt += ammount
        self.print_balance()
    def pay_debt(self, ammount):
        self.balance -= ammount; self.debt -= ammount
        self.print_balance()
    def print_balance(self):
        print('Your account balance is', self.balance,
         '\n You own the bank', self.debt)
```

Playing arround with the newly created object

```
in : john = bank_account(100.0,10)

in : john.withdraw(2.77)
out: Your account balance is 97.23
 You own the bank 10

in : john.deposit(10.00)
out: Your account balance is 107.23
 You own the bank 10

in : john.get_loan(1000.0)
out: Your account balance is 1107.23
 You own the bank 1010.0

in : john.pay_debt(723.0)
out: Your account balance is 384.23
 You own the bank 287.0
```

Objects are incredibly useful

- I have no idea how people created large programs before object oriented programming
- You can use objects to organize a collection of functions
- Use dir() to see all of the attributes of an object
- Python naming convention object_instance.attribute
- attributes can be new data types, data structures, and even new objects
- . is used to access the object's attributes

Custom rich comparisons for your object

Here I define custom rich comparisons for my object, which only compares the attribute x of the object.

```
def __lt__(self, other) # <
    return self.x < other.x
def __le__(self, other) # <=
    return self.x < other.x
def __eq__(self, other) # ==
    return self.x < other.x
def __ne__(self, other) # !=
    return self.x < other.x
def __gt__(self, other) # >
    return self.x < other.x
def __ge__(self, other) # >=
    return self.x < other.x
```

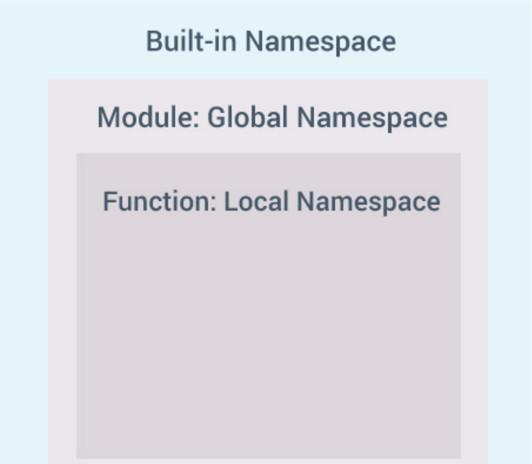
Objects summary

- class defines an object
- def __init__ is a function that runs upon the initialization of an object
- dir(object) displays the attributes of an object
- __eq__ defines equality for your object
- You can define custom comparisons for your objects

Python Namespace

The namespace in Python is a system to make sure that all the names in a program are unique and can be used without conflict. Namespace is a way to implement scope.

`https://www.programiz.com/python-programming/namespace`



An example of name space - dir() shows us the active names or attributes

```python
a = 2.0

def outter_fun():
    b = 3.0
    def inner_fun():
        c = 3.0
        print('inner function', dir())
    inner_fun()
    print('outter function', dir())

outter_fun()
print('script space', dir())
```

Functions and namespace

```
x = 2.0
def print_x_new():
    x = 1.0
    print(x)
print_x_new()
```

Does this code change x?

Functions and namespace - global

```
x = 2.0
def print_x_new():
    global x
    x = 1.0
    print(x)
print_x_new()
```

Does this code change x?

You need to declare *global* before variable assignment

Namespaces in Python is a good idea

http://pclib.github.io/safari/program/learning-python/Text/ch29s04.html

Some good reading with Namespaces and Python.

There is much to discuss but little time.

I recommended taking a look at the source code of how a library you use is organized. Such as numpy
https://github.com/numpy/numpy/tree/master/numpy

Python standard libraries

`https://docs.python.org/3/library/index.html` The Python libraries available with any Python installation.

- math - Mathematical function
- cmath - Mathematical functions for complex numbers
- itertools - functions for creating iterators for efficient looping
- pickle - Python object serialization (storing objects)
- csv - csv file read and write
- os - miscellaneous operating system interface
- and many more!

Special Python libraries we'll use in this course

- numpy - fundamental package for scientific computing with Python
- matplotlib - Python 2D plotting library
- scipy - Python-based ecosystem of open-source software for mathematics, science, and engineering
- sympy - symbolic math with Python
- sklearn - scikit-learn machine learning in Python

Libraries and namespace - basic import

```
import math

math.cos(math.pi)
```

- explicit import of the math library
- this is the most recommended type of import
- the functions of the math library exist in the math namespace
- run dir(math) to view all of the functions
- you access the functions using .

Libraries and namespace - import as alias

```
import numpy as np
```

```
np.cos(np.pi)
```

- sometimes it's inconvient to use the entire name of a library
- in this example we assign an alias np using as
- use the convention when importing libraries
- it is the convention to import numpy as np

Libraries and namespace - explicit import of specific functions

```
from math import cos, pi

cos(pi)
```

- sometimes it is useful to import just a few functions of a library
- in this case there will be no math namespace
- instead the functions cos and pi will occur in the local namespace
- from library import function1, function2, function3

Libraries and namespace - implicit import of all functions

```
from sympy import *
```

- this imports all of the functions of sympy into the local namespace
- if this isn't officially recommended by your library, it could break default Python functions
- for instance from numpy import * would override max() and min() functions with the numpy max() and min() functions

import os - useful for importing your operating system functions

```
import os
os.getcwd()    # returns the current working directory as a stri
os.chdir(path) # changes the working directory to path
os.listdir()   # returns a list of the entries in the current di
os.listdir(path)# a list of the entries in the path directory

os.system()    # lets you run commands from your system terminal
'''
os.system(command)
    Execute the command in a subshell.
    '''
```

The PyPA recommended tool for installing Python packages

pip is a tool for installing python packages – execute pip from the anaconda prompt/terminal or the canopy prompt/terminal

https://pip.pypa.io/en/stable/quickstart/

Install a package:

```
$ pip install numpy
```

Upgrade a package:

```
$ pip install --upgrade numpy
```

Upgrade pip:

```
$ pip install --upgrade pip
```

List what packages are outdated:

```
$ pip list --outdated
```

conda - if you installed Anaconda

conda is part of the Anaconda distribution and is a package, dependency manager for multiple languages `https://conda.io/docs/`

You access conda from the Anaconda prompt/terminal

To install a package:

```
$ conda install <package-name>
```

To list the packages you have installed:

```
$ conda list
```

To update all packages:

```
$ conda update --all
```

To list all packages that are available

```
$ conda search
```

HW 03 - turn in one week from today in Canvas

Turn in the 5 questions as a single .py file onto canvas. Use comments to clearly indicate which question you are working on. Your filename should end as _py2.py if you use Python2 and _py3.py if you use Python3.

1. Open an Anaconda/Canopy prompt/terminal. Enter the command *pip install pydoe* or (*conda install pydoe* if you've installed anaconda) to install the pydoe package. This is a Python Design of Experiments library. In your .py file import pyDOE.
2. Use the os library to print the current Python working directory. It is very useful to run system commands using os.system(). Import os. Run a system command to use the system's *ping* program. If you are using Windows run the command *ping -n 2 ufl.edu* or if you are using Linux/OS *ping -c 2 ufl.edu*. Hint: your command should be a string in Python.
3. Compare math.pi to numpy.pi. Are these two representations of π equivalent? Print the boolean statement True if they are, otherwise print False.

HW 03 - turn in one week from today in Canvas

4. Create a class called sphere. The object sphere requires a radius and mass to initialize. The attributes of the sphere should include the radius (r), mass (m), volume (v), surface area (A), and density (ρ). Initiate a new sphere name red with $r = 1.7$ and $m = 0.25$. Print dir(red). Print the volume, surface area, and density of red.

HW 03 - turn in one week from today in Canvas

5. The Python 3 print function adds some incredibly useful functionality

```
x = 1.0; y = 2.0;
print(x,y,sep = ' & ')
```

will print *1.0 & 2.0*

Given x

```
x =   [[ 0,  1,  2,  3],[ 4,  5,  6,  7],
[ 8,  9, 10, 11],[12, 13, 14, 15]]
```

Use a for loop to iterate through the four lists in x. Each item in the list should be printed and separated by an &. The following should be the output of your print.

```
0 & 1 & 2 & 3
4 & 5 & 6 & 7
8 & 9 & 10 & 11
12 & 13 & 14 & 15
```

Hint: from __future__ should go at the top of your script if you are using Python 2.