Python Workshop Series Session 4: *Objects and Modules*

Nick Featherstone
Applied Mathematics

Daniel Trahan
Research Computing

Slides: https://github.com/ResearchComputing/Python_Spring_2019

Outline

- Objects & Methods
- Operator Overloading
- Modules

- Note: Due to time constraints, we will not discuss inheritance. See online text, chapter 23 for a concise overview

Classes & Objects in Python

- Class refers to a complex data type that may contain both associated values and associated functions
- Distinct instances of a class are referred to as objects
- Methods are defined as functions within class definition
- Class Definition syntax (try this out):

class keyword

init method
(double underscores)

additional
methods

```
class myclass:
    def __init__(self):
        self.val = 2
    def setval(  self , x):
        self.val = x
    def display(self):
        print(self.val)
```

self parameter

Instantiation

- Initialize objects by calling the class name as a function.
- The init method is run at instantiation time

```
obj1 = myclass( )
```

- Object attributes are referred to by prepending the object name to the attribute, with a DOT in between

```
print( obj1.val )
```

Using Methods

- Class methods are called by prepending the object name to the method name, with a DOT in between
- The self parameter is *"silent"* (not explicitly passed).
- Self is understood to refer to the particular instance of the class calling the method

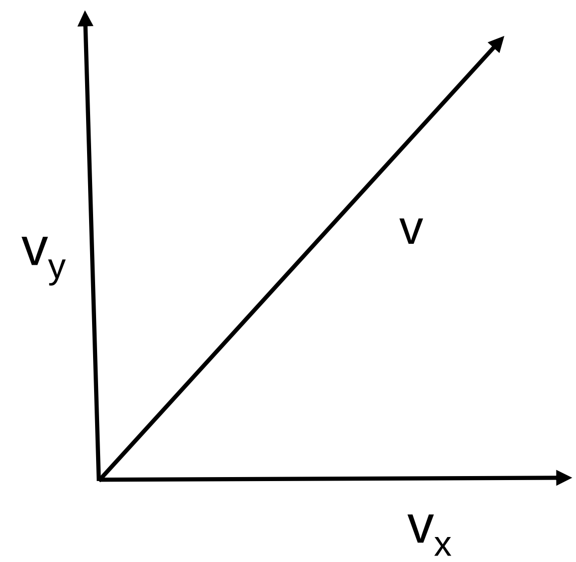
```
obj1 = myclass( )
obj2 = myclass( )

obj1.setval(42)

obj1.display( )

obj2.setval(37)
```

self refers to obj1

self refers to obj2

Object Example: Vectors

- Recall that a vector in N-dimensional space is a combination of N numbers.
- The *ith* number represents the magnitude of *something* in the *i*-direction
- Example: Velocity (miles per hour)
  - $\mathbf{v} = v_x \boldsymbol{x} + v_y \boldsymbol{y} + v_z \boldsymbol{z}$
  - $\mathbf{v} = 1\boldsymbol{x} + 12\boldsymbol{y} + 3\boldsymbol{z}$
    - Speed in x-direction (v_x):   1 mph
    - Speed in y-direction (v_y): 12 mph
    - Speed in z-direction (v_z):   3 mph



Some Vector Properties

- Addition and Subtraction is component-wise:
  - $\mathbf{v} - \mathbf{w} = (v_x - w_x)\mathbf{x} - (v_y - w_y)\mathbf{y} - (v_z - w_z)\mathbf{z}$
- Vector magnitude $|\mathbf{v}|$:
  - $|\boldsymbol{v}| = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2}$
- Vector dot product $\boldsymbol{v} \cdot \boldsymbol{w}$
  - $\boldsymbol{v} \cdot \boldsymbol{w} = v_x w_x + v_y w_y + v_z w_z$
- Vector cross product $\boldsymbol{v} \times \boldsymbol{w}$
  - if $\boldsymbol{b} = \boldsymbol{v} \times \boldsymbol{w}$ then:
    - $b_x = v_y w_z - v_z w_y$
    - $b_y = v_z w_x - v_x w_z$
    - $b_z = v_x w_y - v_y w_x$

Exercise 1

- Let's have a look at vectors.py
- Add a method named mag to the vector class that accepts no parameters (other than self).
- Have your method return the vector's magnitude (a scalar value)
- Recall that exponentiation in Python is done via **
- A**2 = 'A squared'
- A**(0.5) = 'square root of A'

- Vector magnitude |**v**|:
  - $|\boldsymbol{v}| = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2}$

Exercise 2

- Add a method named plus to the vector class that accepts an additional parameter named other.
- Assume that other is an object of type "vector"
- The method should return a new vector which is created by taking the vector sum of self and other.

- Once you've done that, create another method named minus that returns the difference of self and other.

Exercise 3

- Add a method named dot to the vector class that accepts an additional parameter named other.
- Assume that other is an object of type "vector"
- The method should return the vector dot product of self and other.

> - Vector dot product $\boldsymbol{v} \cdot \boldsymbol{w}$
>   - $\boldsymbol{v} \cdot \boldsymbol{w} = v_x w_x + v_y w_y + v_z w_z$

- Finally, when that's finished, add a similarly-structured method named cross that returns the vector cross product of two vectors.

Operator Overloading

- v.add(w) is concise, but non-intuitive

- Is there a way to say “v +w” ? Yes!
- Follow these steps:
  - Open vectors_completed.py
  - Create a COPY of the plus function
  - Name the new function __add__ (two underscores on each side)
  - Try using v + w in your code now

Operator Overloading

- Several special method names exist:
  - __sub__ : replaces –
  - __mul__ : replaces * (two of the same object)
  - __rmul__ : replaces * (object and scalar)
  - __truediv__ : replaces /
  - __floordiv__ : replaces //
  - __pow__ : replaces **

Exercise 4

- Following our __add__ example, overload operators with the remaining methods in the vector class as follows:
  - minus : - (__sub__)
  - dot : * (__mul__)
  - cross : ** (__pow__)

Modules

- Python allows us to collect associated functions, class, and variables into modules
- Modules may be imported into other modules or into your main program
- Essentially any .py file can be imported as a module
- Let’s have a look at my_module.py

Defining Modules

Any .py file with function definitions etc. works as a module.

```
def myfunc():
        print('my function')
def main( ):
    print("hello world")

val1 = 1
val2 = 2

if __name__ == "__main__":
    main( )
```

Executed when module is imported

Executed only if module is being run as the main program

Importing Modules

- We can import an entire module, or only certain items
- To reference a module variable, use the syntax:
  module_name (DOT) variable_name
- We can assign an alias to our module name at import time using the as keyword
- See import_module.py

```
import my_module
print(  my_module.val1 )
my_module.myfunc()
```

```
import my_module as mm
print(  mm.val1 )
mm.myfunc()
```

Selective importing

- Selectively import specific items using the from keyword
- Syntax:

  from ‘module name’ import ‘variable name’
- Can import everything using * (take care!)
- When using from, the module name is not prepended

```
from my_module import val1
print(  val1 )
```

```
from my_module import *
print( val2 )
myfunc( )
```

Intrinsic Python Modules

- https://docs.python.org/3/py-modindex.html
- Some particularly useful modules:
  - math – provides sine, cosinie, sqrt etc.
  - random – for random number generation
  - time – useful for measuring execution time
  - sys – system/ info (e.g., getrecursionlimit , argv)
  - os -- various system routines (ls, mkdir, etc.)
  - tkinter – Python GUI utilities

Agument Lists

- sys.argv is particularly useful for scripting
- Lists all command-line arguments passed to program
- sys.argv[0] = program name
- Open / examine argv.py

Where do modules live?

- Python places modules deep within its directory structure.
- Best not to place your custom modules here
- Let's have a quick look. (Bash commands follow)

```
which python
```

/custom/software/miniconda3/envs/idp3/bin/python

```
export PYDIR=/custom/software/miniconda3/envs/idp3
```

```
ls $PYDIR/lib/python3.6/site-packages/
```

PYTHONPATH

- Python refers to the environment variable, PYTHONPATH for possible module locations.

- We can manipulate PYTHONPATH within our program.

```
import sys
sys.path.append('/path/to/my/modules')
```

- More on PYTHONPATH and package management next time.

RC Jupyterhub

- Web-based access to your data on Summit and the Petalibrary
- https://jupyter.rc.colorado.edu (note ‘https’)
- Can test upcoming interface at:
  - https://tutorials-jupyter.rc.colorado.edu

JupyterLab

- More sophisticated notebook interface
- https://jupyterlab.readthedocs.io/en/stable/