

Programming in Python

CU Research Computing and CRDDS

Nick Featherstone
feathern@colorado.edu

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**

class keyword

self parameter name

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

init method

associated
methods

Instantiation

- Objects may be initialized 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 may be called by prepending the object name to the method name, with a DOT in between
- The **self** parameter is “*silent*.”

```
obj1 = myclass( )
```

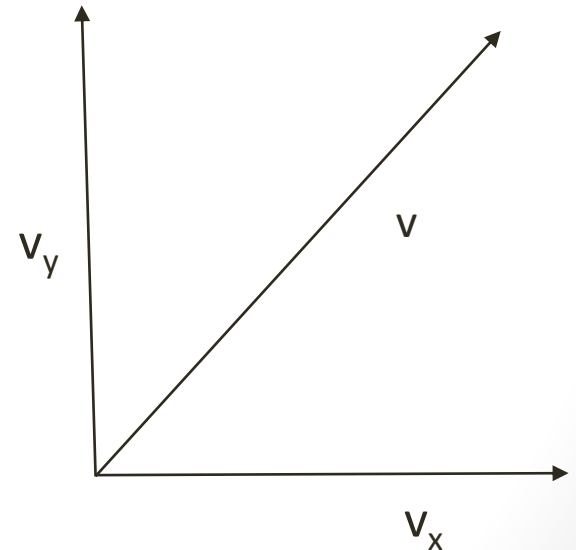
```
print( obj1.val )
```

```
obj1.setval(42)
```

```
obj1.display( )
```

Object Example: Vectors (physics)

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

- Addition/Subtraction
 - $\mathbf{v} - \mathbf{w} = (v_x - w_x)\mathbf{x} - (v_y - w_y)\mathbf{y} - (v_z - w_z)\mathbf{z}$
- Vector magnitude $|\mathbf{v}|$:
 - $|\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$
- Vector dot product $\mathbf{v} \cdot \mathbf{w}$
 - $\mathbf{v} \cdot \mathbf{w} = v_x w_x + v_y w_y + v_z w_z$
- Vector cross product $\mathbf{v} \times \mathbf{w}$
 - if $\mathbf{b} = \mathbf{v} \times \mathbf{w}$ 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

- 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'

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`.
-
- 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 `*` (for two of the same object)
 - `__rmul__` : replaces `*` (for object and scalar)
 - `__truediv__` : replaces `/`
 - `__floordiv__` : replaces `//`
 - `__pow__` : replaces `**`

Exercise

- Following our `__add__` example, overload the remaining methods in the `vector` class as follows:
 - `minus` : `-`
 - `dot` : `*`
 - `cross` : `/`

Modules

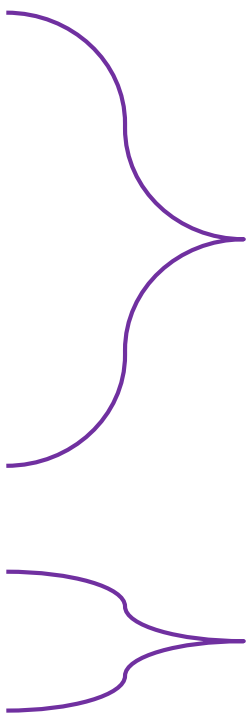
- Python allows us to collect associated functions and variables into modules
- Modules may be imported into other modules or into your main program
- Essentially any .py file can be imported as a module
- Let's have a look at `my_module.py`

Defining Modules in Python

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

```
val1 = 1  
val2 = 2
```

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



Executed whenever
module is imported

Executed only if
module is being run
as the main program

Importing Modules in Python

- We can import an entire module, or only certain things from a module
- When importing entire module, module name is prepended to variable names as `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

- We can selectively import only certain functions or variables from a module using the **from** keyword
- Syntax is “from module_name import variable name”
- We can import everything using * in lieu of variable name (be careful!)
- When using from, the module name is not prepended

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

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