

Computational Physics

Object Oriented Programming in Python

Outline

- What is Object Oriented Programming?
- Why design programs to be Object Oriented?
- Features of Object Oriented Programs
- Implementation in Python

Procedural Programming

- Design emphasis is on setting up the logic of the program and its supporting functions.
 - We can improve things if we structure the program to make use of functions to do things we do a lot.
- Define or input the data you want to operate on.
 - Write read and write functions
 - Set up structures for the data
- Generally, you need to know a lot about the procedures and the data structures to make use of the program or modify it.
- Ultimately provide some output....

Procedural Example

```
import numpy as np
import matplotlib.pyplot as pl

def falling_ball(x0,v0,g,t):
    return x0 + v0*t + 0.5*g*t**2

def main():
    # set variables
    x0 = 100.
    v0 = 0.
    g = 9.8
    t = np.arange(101.)

    # define output array
    x = np.zeros(101)

    # compute
    for i,tt in enumerate(t):
        x[i] = falling_ball(x0,v0,g,tt)

    # now do something, like plot... e.g.
    pl.plot(t,x)

main()
```

Define a function we'll call a lot.

Initialize variables (data) we need to do the calculation.

Set up a structure to hold the result. Note that we have to be careful about Making x the correct size.

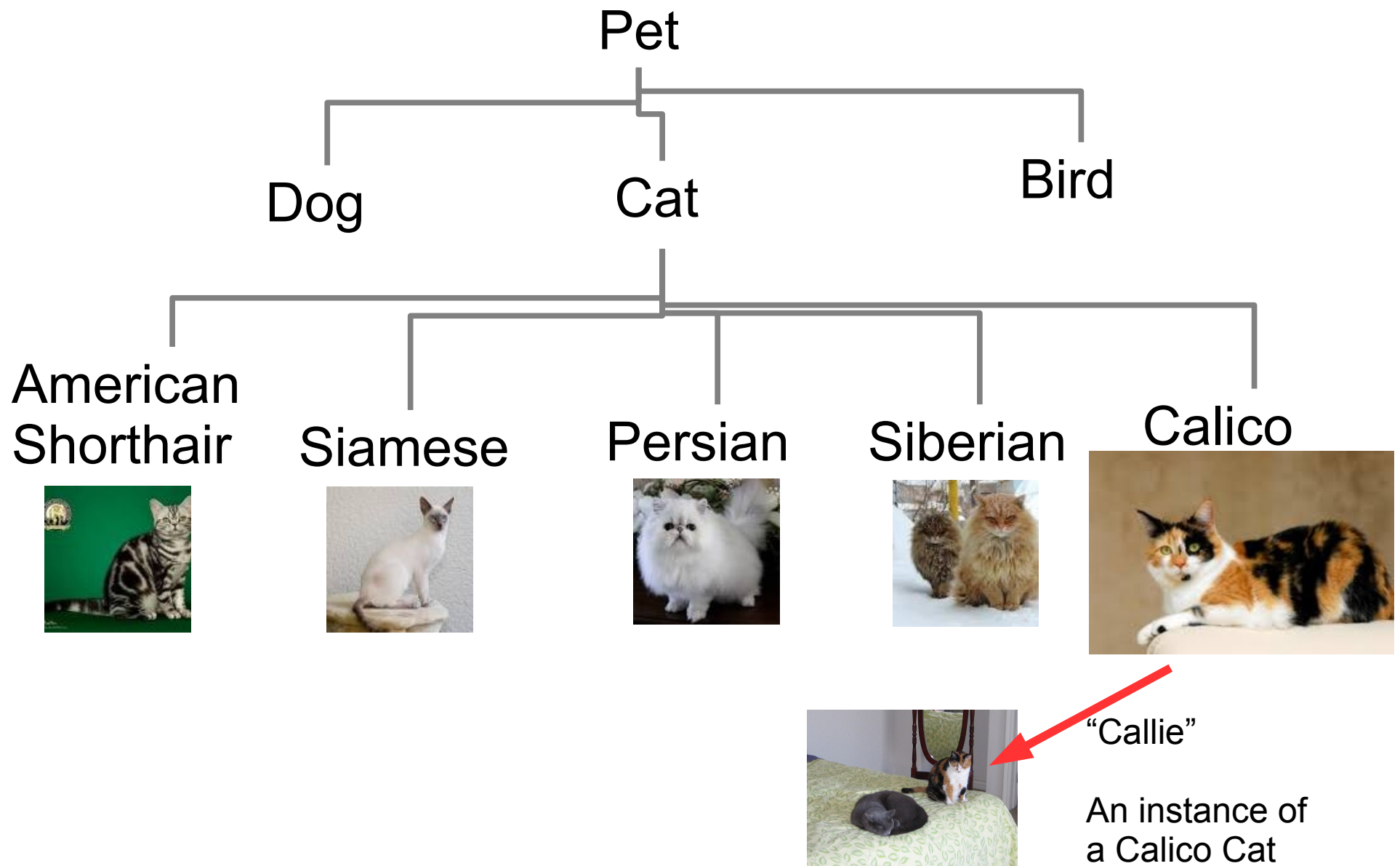
Now do the calculation. Again we have to be careful to write our program so that things come out even.

Finally we want to do something with the result

Object Oriented Programming

- Emphasis is on writing “objects”
- Objects contain data; data is maintained in a structure
- Objects contain “methods” which operate on the data.
- Objects have well defined ways to interface to other objects and programmers.
- Individual Objects can belong to the same “class” of objects. The same thing ... just different data and properties.

Classes, Objects, Instances



Stuff about Cats

- Cats have attributes (data)
 - Name
 - Breed
 - DNA
- Cats do stuff (methods)
 - Eat
 - Purr
 - Play with string
- Cats are really complicated inside, but still easy to use. You don't have to know about DNA to play with a cat!

Why Design OO Programs?

- Objects are easy to use, extend, and reuse
- Objects “encapsulate” their data
 - The inner workings are hidden to avoid confusion and only accessed through a well defined interface.
 - User does not have to know about manipulation of data structures necessary to use the data in an object.
- Objects may be extended to create new object which inherits the workings of an existing object.
 - You don't have to rewrite a lot of existing code to define something that is similar to something that already exists.

Criticisms of OOP

- Efficiency
 - More time to design
 - More things to code, so your program has to lug around more stuff....
- Everything is not an object, why force that?
 - Algorithms and computation are important too.
 - Why emphasize the nouns over the verbs?
 - How do you model time?
 - How do you handle parallel processing?

Features of Software Objects

- Attributes or Properties (Data)
 - Abstraction – How does the object look from the outside? What are the essential details of the object that will allow it to be used easily?
 - Encapsulation – How is the data managed on the inside to achieve the desired behavior? What are the necessary details which are not needed in order to use the object?
- Methods - functions which belong to the object
- “Data Hiding”
 - Public - available for use from outside; define what the object does.
 - Protected or Private - used inside and nominally unseen from the outside; used to achieve desired behavior.
- Inheritance – ability to extend old objects and create new ones.

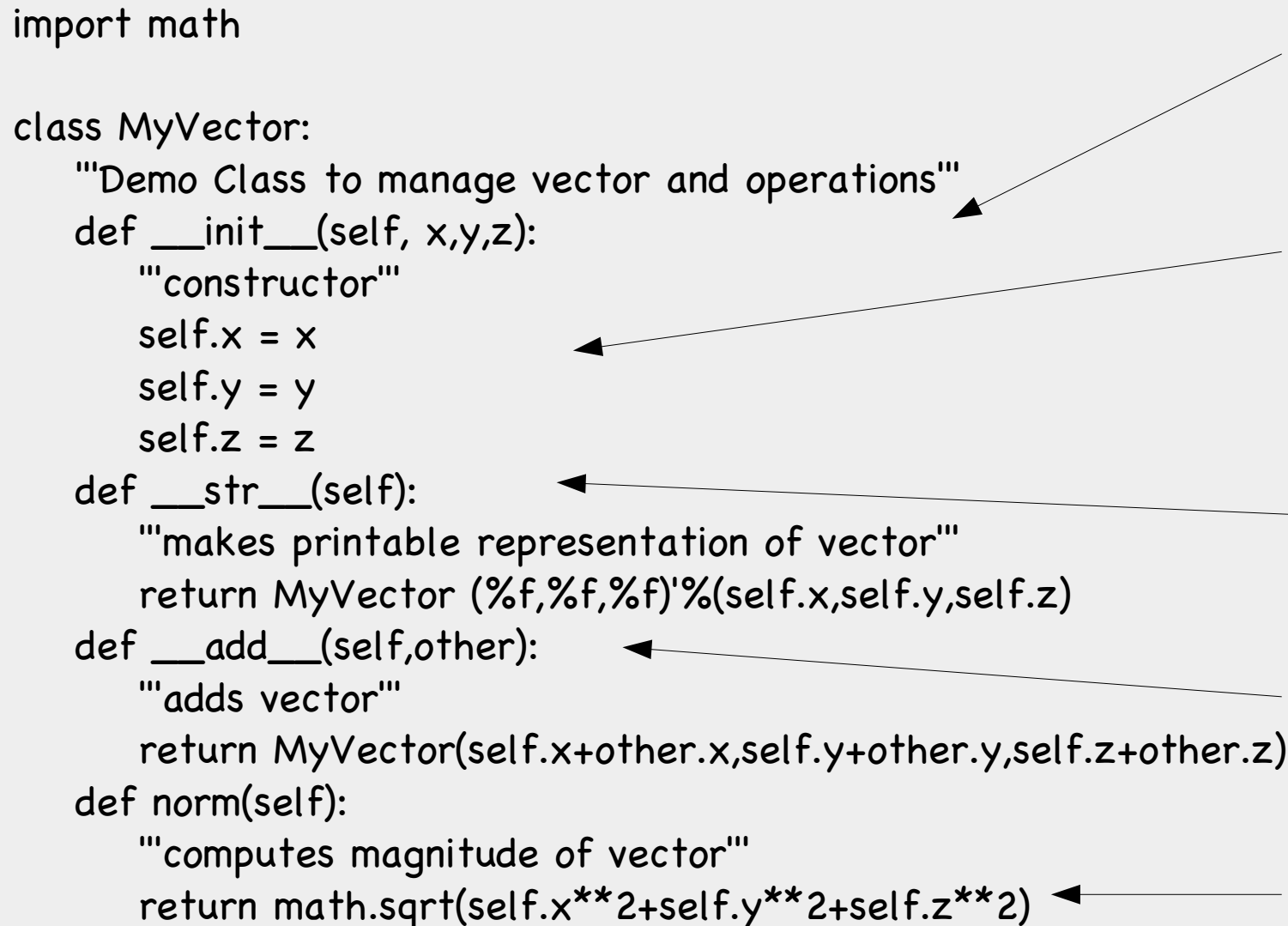
OOP and Python

- Python is Object Oriented by design.
 - Everything in Python is a class
 - Python has ability to make new classes that allow OOP features to be incorporated easily
- We need to learn about writing and using Python “Classes”
 - Consider an example --- “MyVector” --- which will deal with vectors and vector operations.
 - See the MyVector.py module on the web site for the complete example.

Implementation of a python class

```
import math

class MyVector:
    """Demo Class to manage vector and operations"""
    def __init__(self, x,y,z):
        """constructor"""
        self.x = x
        self.y = y
        self.z = z
    def __str__(self):
        """makes printable representation of vector"""
        return MyVector (%f,%f,%f'%(self.x,self.y,self.z)
    def __add__(self,other):
        """adds vector"""
        return MyVector(self.x+other.x,self.y+other.y,self.z+other.z)
    def norm(self):
        """computes magnitude of vector"""
        return math.sqrt(self.x**2+self.y**2+self.z**2)
```

The diagram consists of five arrows pointing from the explanatory text on the right to specific parts of the Python code on the left. The first arrow points from the text '___init___ is the constructor method creates an instance of "MyVector"' to the 'def __init__(self, x,y,z):' line. The second arrow points from 'x,y,z are the data for the vector. "self" references this instance of MyVector' to the 'self.x = x', 'self.y = y', and 'self.z = z' lines. The third arrow points from '___str___ is available in all python classes to make a string used by "print".' to the 'def __str__(self):' line. The fourth arrow points from '___add___ tells what to do with the + operation.' to the 'def __add__(self,other):' line. The fifth arrow points from 'norm is our own special method for this class' to the 'def norm(self):' line.

`__init__` is the constructor method creates an instance of "MyVector"

x,y,z are the data for the vector. "self" references this instance of MyVector

`__str__` is available in all python classes to make a string used by "print".

`__add__` tells what to do with the + operation.

norm is our own special method for this class

See module **MyVector.py** for complete implementation of example

Using MyVector

In [1]: `from MyVector import MyVector`

Import the MyVector class from the file MyVector.py

In [2]: `a = MyVector(3.,1.,0.)`

Create an instance of MyVector

In [3]: `print a`
`MyVector (3.000000,1.000000,0.000000)`

Print a using print command, which uses `__str__` method.

In [4]: `b = MyVector(2.,2.,2.)`

Create another vector

In [5]: `print a+b`
`MyVector (5.000000,3.000000,2.000000)`

Add vectors a and b and print result

In [6]: `print a.norm()`
`3.16227766017`

Use norm method and print magnitude of the vector a

Extending MyVector

```
class MyPolarVector(MyVector):  
    "vector in polar coordinates"  
    def __init__(self,r,t,p):  
        "constructor"  
        MyVector.__init__(self,  
                            r*math.cos(t)*math.cos(p),  
                            r*math.cos(t)*math.sin(p),  
                            r*math.sin(t))  
  
    def r(self):  
        "returns the radius component"  
        return self.norm()  
    def phi(self):  
        "returns azimuth"  
        return math.atan2(self.y,self.x)  
    def theta(self):  
        "returns polar angle"  
        return math.asin(self.z/self.norm())
```

New class "MyPolarVector" inherits all the functionality of "MyVector". We can then add new methods for this class.

Constructor must first create an instance of the MyVector class.

Then we can create other useful methods for the new class.

Using MyPolarVector

```
In [1]: from MyVector import MyVector, MyPolarVector
```

Import MyVector and
MyPolarVector

```
In [2]: a = MyVector(3.,1.,0.)
```

```
In [3]: b = MyPolarVector(3.,1.,0.)
```

Create instances of each

```
In [4]: print b
```

```
MyVector (1.620907,0.000000,2.524413)
```

Print MyPolarVector uses
The `__str__` method from
MyVector

```
In [5]: print a+b
```

```
MyVector (4.620907,1.000000,2.524413)
```

Addition uses method from
MyVector

```
In [6]: print b.r()
```

```
3.0
```

```
In [7]: print b.theta()
```

```
1.0
```

```
In [8]: print b.phi()
```

```
0.0
```

Here are the new methods for
MyPolarVector

Features of the *MyVector* class

- We deal with vectors as an entity. This simplifies our use of them.
- Abstraction
 - Operations are standard vector operations. We don't sweat the details of what happens to components.
- Encapsulation
 - After definition, no need to worry about components and their description inside the class. For example, we could have used NumPy arrays and operations to deal with the vectors and users would not know the difference....