

Object Oriented Programing

Joe

Outline

DSI Standards

- You need to leave knowing the following information:
 - Given the code for a python class, instantiate an instance and call it's methods
 - Match key “magic” methods to their syntactic sugar
 - Design a program in an object oriented fashion
 - Write the python code for a simple class
 - Compare and contrast functional and object oriented programming

Morning Objectives

- Explain the three properties of object oriented programming (OOP)
- Create a class for a line with basic functionality

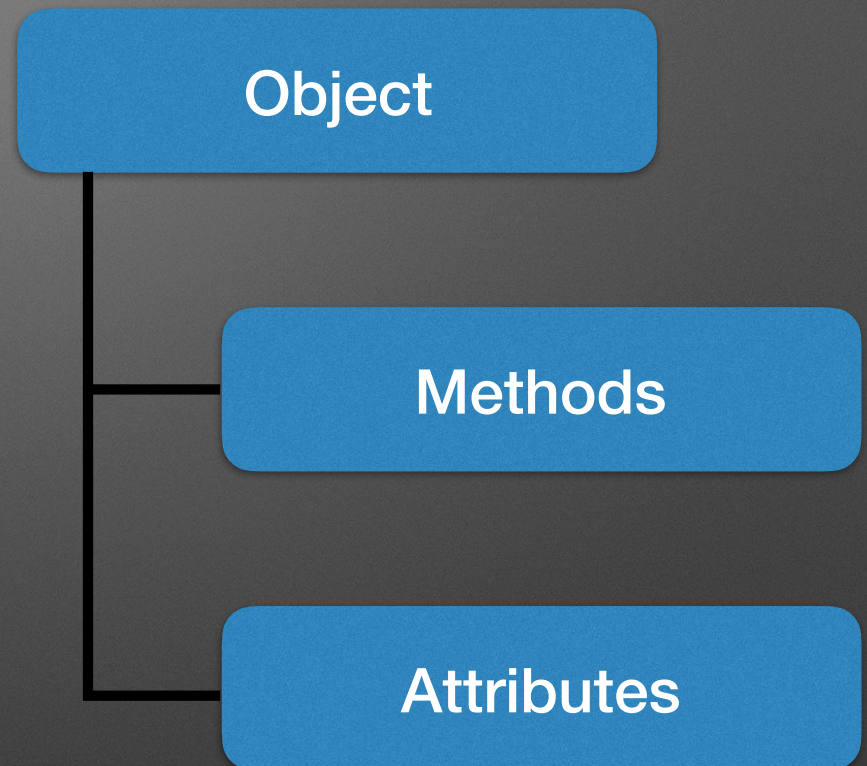
Recommended Reading

- Writing Idiomatic Python by Jeff Knupp
- Python 3 Object-Oriented Programming by Dusty Philips

Definitions

The Three Properties of OOP

1. Encapsulation - Objects should contain both the data and functions associated with a task. Data should only be modified through methods associated with a class.
2. Inheritance - Objects can be created from other objects that have the same properties
3. Polymorphism - Objects with the same interface should be able to be used in the same way.



Classes - user defined types with methods and attributes

Object - an instance of a class

Example: Creating an sklearn Object

```
from sklearn.linear_model import LinearRegression

X = [[x] for x in range(10)]
y = [x if x%2==1 else x+1 for x in range(10)]

1 model = LinearRegression()
2 model.fit(X, y)
3 print(model.coef_)
  print(model.intercept )

[ 0.96969697]
0.636363636364
```

- sklearn is the machine learning package we will use for most models in this program
- Here we
 1. Create an instance of a linear regression class
 2. Fit the model to data
 3. Access the attributes for the object

Why OOP?

- OOP is a paradigm in which functions and data are organized in a way so that clients (i.e. users of the program that didn't write them) can leverage with ease
- OOP promotes standard APIs so that similar functions are called in similar ways
- The basic concept is that OOP is used to make code easier to use correctly by more people.

Drawbacks of OOP

- OOP makes code easier to use for more people, at the cost of being more time consuming to write and test.
- We discussed that premature optimization is a major time drag on progress
 - Have an idea if something should work before you code it up, and fail fast in the case you are wrong
- The alternative is called functional programming
 - Functional programming has its place and is what we've done up to this point

Example Class - Demonstrating Encapsulation

OOP in Python

```
class Point:
```

OOP in python is implemented via classes (similar to C++ & Java).

Classes give us the ability to organize data and functions that are related in a meaningful way. This is the first property of OOP - Encapsulation.

OOP in Python

```
class Point:
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def __str__(self):
        return "(x=" + self._x.__str__() + ", y=" + self._y.__str__() + ")"
```

Classes have a set of functions called “special methods”, that define behaviors of the class. There are a lot of these, but the most important is the `__init__` function, which you use to create an instance of the class.

N.B. use ‘`_`’ prefixes to attributes to denote they should not be called by users.

OOP in Python

```
class Point:
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def __str__(self):
        return "(x=" + self._x.__str__() + ", y=" + self._y.__str__() + ")"

    def get(self):
        return (self._x, self._y)
```

One type of function is called an ‘accessor’ which gives clients access to the object attributes.

For those coming from Java/C++, python does not have the notion of public/private data, “*We’re all consenting adults*”.

OOP in Python

```
class Point:
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def __str__(self):
        return "(x=" + self._x.__str__() + ", y=" + self._y.__str__() + ")"

    def get(self):
        return (self._x, self._y)

    def transpose(self, dx=0, dy=0):
        self._x = self._x + dx
        self._y = self._y + dy
```

Another type of function allows you to modify the attributes of a class; often times you will have logical checks to assure that no internal variables have leagal values.

Example Class - Demonstrating Inheritance

Inherited Class

```
class Origin(Point):
```

Classes Inherit from one another by passing in “parent” classes in the declaration.

By default, the derived class will have all the same functions as the parent class

Inherited Class

```
class Origin(Point):  
    def __init__(self):  
        self._x = 0  
        self._y = 0
```

In the case that methods behaviors should be modified, you can define alternative behaviors that will supersede the parent class's behavior

Inherited Class

```
class Origin(Point):  
    def __init__(self):  
        self._x = 0  
        self._y = 0  
  
    def __str__(self):  
        return "Origin->" + super(Origin, self).__str__()
```

Similarly, the 'super' function will call the behavior of the inherited class so that it can be modified, but still reuse that piece of code.

Inherited Class

```
class Origin(Point):  
    def __init__(self):  
        self._x = 0  
        self._y = 0  
  
    def __str__(self):  
        return "Origin->" + super(Origin, self).__str__()  
  
    def transpose(self, dx=0, dy=0):  
        raise ValueError("Cannot transpose the origin")
```

Classes should have a common interface, but throw errors when used incorrectly.

Demonstrating Polymorphism

Python Polymorphism

- Python uses what's called 'duck typing', i.e. "if it looks like a duck and quacks like a duck, it's a duck"
- Polymorphism in C++ is a lot more complicated involving pointers based on base classes to call inherited classes in regular ways
- Python is awesome and doesn't make you deal with memory management, so polymorphism here is pretty straight forward

```
p1 = Point(3,2)
p2 = Origin()
p3 = Point(2,1)

for P in [p1, p2, p3]:
    print(P)

(x=3, y=2)
Origin->(x=0, y=0)
(x=2, y=1)
```

Let's work together to build a python class for a line.

Morning Exercises

- Our objectives were:
 - Explain the three properties of object oriented programming (OOP)
 - Create a class for a line with basic functionality
- Time to break out for individual sprints!

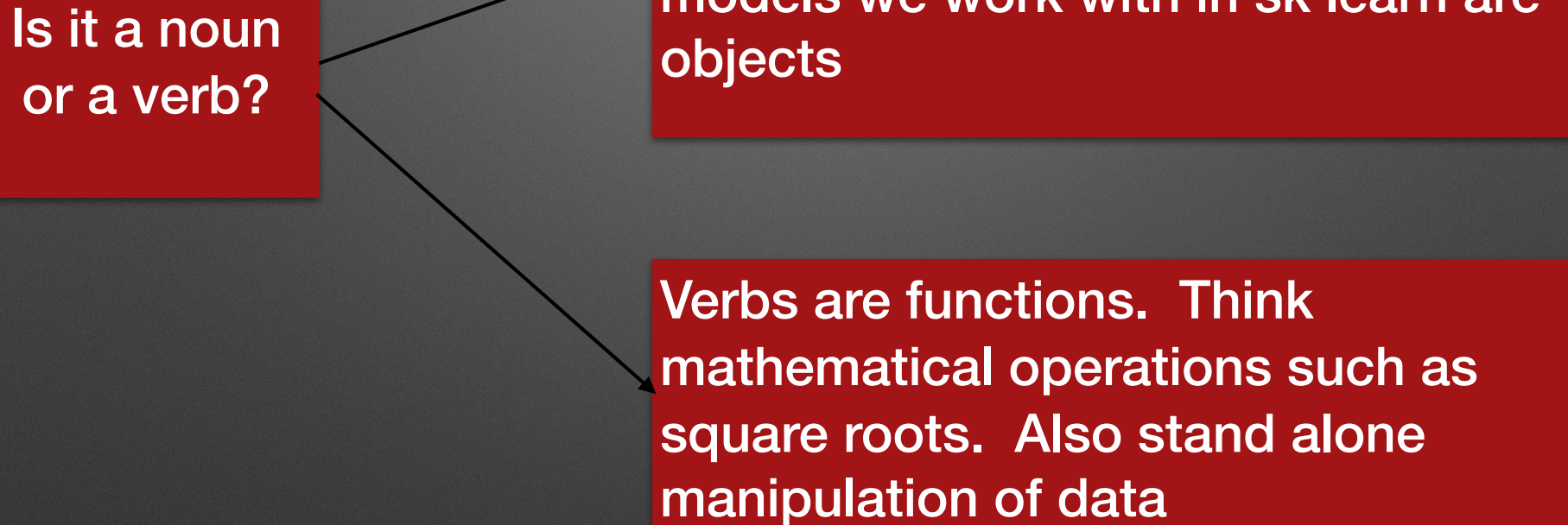
Afternoon Objectives

- State best design practices and identify usage is mornings demos
- Use special input types to define objects of variable size

Design

Design Principals 1 - When to Use OOP

Is it a noun
or a verb?



```
graph LR; A[Is it a noun or a verb?] --> B[Nouns are objects. For example, the models we work with in sklearn are objects]; A --> C[Verbs are functions. Think mathematical operations such as square roots. Also stand alone manipulation of data];
```

Nouns are objects. For example, the models we work with in sklearn are objects

Verbs are functions. Think mathematical operations such as square roots. Also stand alone manipulation of data

Design Principals 2 - Start simple

Rec 1 - Point

Attributes: x, y

Methods:

Init in cart. coords

String

getters

Transpose

Rec 1 - Point

Attributes: x, y, z (optional), r, theta, rho, phi, default coord

Methods:

- * Init in cart., cylindrical, & spherical coords

- * Strings, with switch for coordinate system

- * getters

- * Transpose

- * Rotate about origin

- * Distance to other point.....

From the Textbooks

- Points from “Effective C++” (my favorite programming book)

Chapter 4: Designs and Declarations	78
--	-----------

Item 18: Make interfaces easy to use correctly and hard to use incorrectly.	78
---	----

Item 19: Treat class design as type design.	84
---	----

Item 22: Declare data members private.	94
--	----

Chapter 6: Inheritance and Object-Oriented Design	149
--	------------

Item 32: Make sure public inheritance models “is-a.”	150
--	-----

Item 37: Never redefine a function’s inherited default parameter value.	180
---	-----

Item 38: Model “has-a” or “is-implemented-in-terms-of” through composition.	184
---	-----

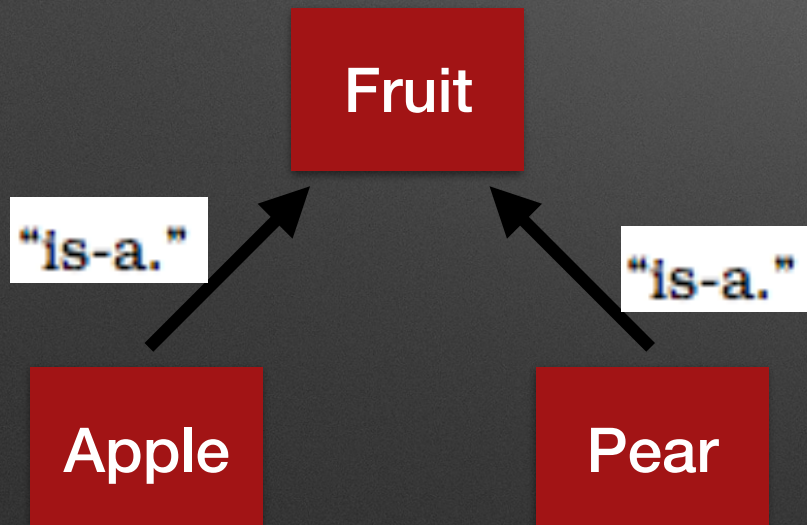
Type Design

Item 19: Treat class design as type design.

84

Item 32: Make sure public inheritance models “is-a.”

150



Fruit - Abstract base class:

<https://docs.python.org/3/library/abc.html>

No private variables

Item 22: Declare data members private.

94

Python does not have the concept of public or private variables.

If you don't have getters and setters, people will access data in unusual ways; providing setters in particular will allow you to put basic sanity checks in place.

Default Parameters

Item 37: Never redefine a function's inherited default parameter value.

180

Look at Fruits.py
in repo

Model Composition

Item 38: Model “has-a” or “is-implemented-in-terms-of” through composition.

Composition is the relationship between types that arises when objects of one type contain objects of another type. For example:

```
class Address { ... };           // where someone lives
class PhoneNumber { ... };
class Person {
public:
    ...
private:
    std::string name;             // composed object
    Address address;              // ditto
    PhoneNumber voiceNumber;      // ditto
    PhoneNumber faxNumber;        // ditto
};
```


Class Specific Decorators

Static methods

```
@staticmethod
def describe():
    print('''
Polygons are collections of points, created either from
PointCollection object called "point_collection". It has
valid collection.
''')
```


Class methods

```
class Polygon(PointCollection):  
    origin = Origin()
```

```
@classmethod  
def print_origin(cls):  
    print(cls.origin)
```


Afternoon Objectives

- State best design practices and identify usage is mornings demos
- Use special input types to define objects of variable size