OBJECT ORIENTED PROGRAMING

(download slides and .py files 'fo

'follow along!)

6.0001 LECTURE 8

OBJECTS

Python supports many different kinds of data

```
1234 3.14159 "Hello" [1, 5, 7, 11, 13] {"CA": "California", "MA": "Massachusetts"}
```

- each is an object, and every object has:
 - a type
 - an internal data representation (primitive or composite)
 - a set of procedures for interaction with the object
- an object is an instance of a type
 - 1234 is an instance of an int
 - "hello" is an instance of a string

OBJECT ORIENTED PROGRAMMING (OOP)

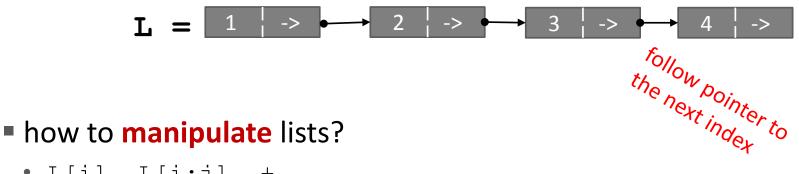
- EVERYTHING IN PYTHON IS AN OBJECT (and has a type)
- can create new objects of some type
- can manipulate objects
- can destroy objects
 - explicitly using del or just "forget" about them
 - python system will reclaim destroyed or inaccessible objects – called "garbage collection"

WHAT ARE OBJECTS?

- objects are a data abstraction that captures...
- (1) an internal representation
 - through data attributes
- (2) an **interface** for interacting with object
 - through methods (aka procedures/functions)
 - defines behaviors but hides implementation

EXAMPLE: [1,2,3,4] has type list

how are lists represented internally? linked list of cells



- L[i], L[i:j], +
- len(), min(), max(), del(L[i])
- L.append(), L.extend(), L.count(), L.index(), L.insert(), L.pop(), L.remove(), L.reverse(), L.sort()
- internal representation should be private
- correct behavior may be compromised if you manipulate internal representation directly

ADVANTAGES OF OOP

- bundle data into packages together with procedures that work on them through well-defined interfaces
- divide-and-conquer development
 - implement and test behavior of each class separately
 - increased modularity reduces complexity
- classes make it easy to reuse code
 - many Python modules define new classes
 - each class has a separate environment (no collision on function names)
 - inheritance allows subclasses to redefine or extend a selected subset of a superclass' behavior

CREATING AND USING YOUR OWN TYPES WITH CLASSES

- make a distinction between creating a class and using an instance of the class
- creating the class involves
 - defining the class name
 - defining class attributes
 - for example, someone wrote code to implement a list class
- using the class involves
 - creating new instances of objects
 - doing operations on the instances
 - for example, L=[1,2] and len(L)

DEFINE YOUR OWN TYPES

use the class keyword to define a new type

```
class Coordinate (object):

class definition #define attributes here
```

- similar to def, indent code to indicate which statements are part of the class definition
- the word object means that Coordinate is a Python object and inherits all its attributes (inheritance next lecture)
 - Coordinate is a subclass of object
 - object is a superclass of Coordinate

WHAT ARE ATTRIBUTES?

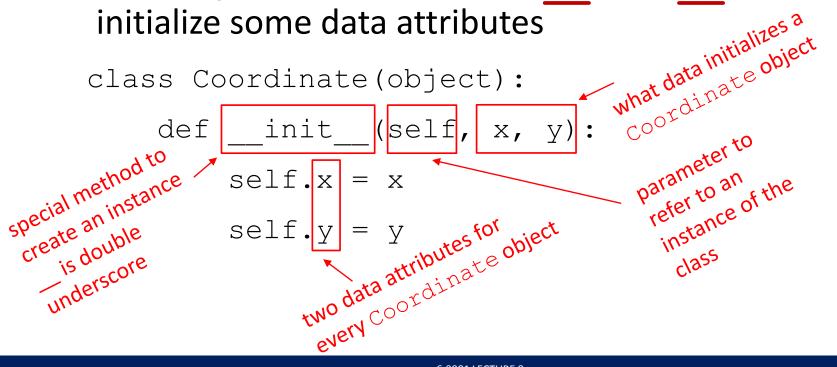
data and procedures that "belong" to the class

data attributes

- think of data as other objects that make up the class
- for example, a coordinate is made up of two numbers
- methods (procedural attributes)
 - think of methods as functions that only work with this class
 - how to interact with the object
 - for example you can define a distance between two coordinate objects but there is no meaning to a distance between two list objects

DEFINING HOW TO CREATE AN INSTANCE OF A CLASS

- first have to define how to create an instance of object
- use a special method called __init___ to initialize some data attributes



ACTUALLY CREATING AN INSTANCE OF A CLASS

```
c = Coordinate(3,4)

origin = Coordinate(0,0)

print(c.x)

print(origin.x)

use the dot to the pass in 3 and 4 to
```

- data attributes of an instance are called instance variables
- don't provide argument for self, Python does this automatically

WHAT IS A METHOD?

- procedural attribute, like a function that works only with this class
- Python always passes the object as the first argument
 - convention is to use self as the name of the first argument of all methods
- the "." operator is used to access any attribute
 - a data attribute of an object
 - a method of an object

DEFINE A METHOD FOR THE Coordinate CLASS

• other than self and dot notation, methods behave just like functions (take params, do operations, return)

HOW TO USE A METHOD

```
def distance(self, other):
    # code here
    method def
```

Using the class:

conventional way

```
c = Coordinate(3,4)
zero = Coordinate(0,0)
print(c.distance(zero))

print(c.distance(zero))

object to call
object to call
name of
method parameters not
including self
including self
including including including including including including implied to be column implied to be column.
```

equivalent to

PRINT REPRESENTATION OF AN OBJECT

```
>>> c = Coordinate(3,4)
>>> print(c)
<__main__.Coordinate object at 0x7fa918510488>
```

- uninformative print representation by default
- define a __str__ method for a class
- Python calls the __str__ method when used with print on your class object
- you choose what it does! Say that when we print a Coordinate object, want to show

```
>>> print(c) <3,4>
```

DEFINING YOUR OWN PRINT METHOD

```
class Coordinate(object):
    def init (self, x, y):
        self.x = x
        self.y = y
    def distance (self, other):
        x diff sq = (self.x-other.x)**2
        y = (self.y-other.y)**2
        return (x diff sq + y diff sq) **0.5
        str (self):
    def
        return "<"+str(self.x)+","+str(self.y)+">"
 name of
                    must return
 special
  method
```

WRAPPING YOUR HEAD AROUND TYPES AND CLASSES

can ask for the type of an object instance

```
>>> c = Coordinate(3,4)
>>> print(c)
<3,4>
>>> print(type(c))
<class main .Coordinate>
```

this makes sense since

```
>>> print(Coordinate)
<class __main__.Coordinate>
>>> print(type(Coordinate))
<type 'type'>
```

use isinstance() to check if an object is a Coordinate
>>> print(isinstance(c, Coordinate))

True

SPECIAL OPERATORS

... and others

+, -, ==, <, >, len(), print, and many others

https://docs.python.org/3/reference/datamodel.html#basic-customization

- like print, can override these to work with your class
- define them with double underscores before/after

6.0001 LECTURE 8

EXAMPLE: FRACTIONS

- create a new type to represent a number as a fraction
- internal representation is two integers
 - numerator
 - denominator
- interface a.k.a. methods a.k.a how to interact with Fraction objects
 - add, subtract
 - print representation, convert to a float
 - invert the fraction
- the code for this is in the handout, check it out!

THE POWER OF OOP

- bundle together objects that share
 - common attributes and
 - procedures that operate on those attributes
- use abstraction to make a distinction between how to implement an object vs how to use the object
- build layers of object abstractions that inherit behaviors from other classes of objects
- create our own classes of objects on top of Python's basic classes

MIT OpenCourseWare https://ocw.mit.edu

6.0001 Introduction to Computer Science and Programming in Python Fall 2016

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms.