CS1 Lecture 24 Mar. 20, 2017

- HW4 grades posted, HW5 scores soon.
  - always make sure to submit the specified number of .py files!
- HW6 available Wed., due next Wed.
- No discussion sections this week
- Approx. grades so far

Last time

- HW5 questions/hints
- List comprehensions
- What are we doing the rest of the semester?
- Simple image processing examples
  - plus simple steganography

Today

- Classes/object-oriented programming overview and initial example. Chapter 15 next time

The rest of the semester

Approximate schedule:

- Ch 15-18: Classes and objects / object oriented programming (1 week)
- Ch 21 (plus material from other sources): running-time analysis, searching, sorting, and other algorithms, randomization (2 weeks)
- GUIs and graphing/charting/visualization (1 week)
- Exam 2 – April 20
- Accessing web data using public APIs, json, etc. (1-2 weeks)
- Limits of computing – what's hard, what's impossible

Remaining work:

- 5 more homework assignments
- 5 more discussion section assignments
- Exam 2
- Final Exam

Ch15-18. Classes and Object-oriented (OO) programming

- This is a very important topic for modern programming.
  - Many many real-world systems are heavily object-oriented. E.g. to program iOS/iPhone/iPad, you'll have to deal with large complex OO libraries/frameworks
- It's a very big topic.
  - terms like: class, object, method, instance, inheritance, abstraction, encapsulation, information hiding, polymorphism, …
  - we'll cover the basics

Introduction to Classes

- defs lets us add new functions. Extremely useful for breaking down large program into components, building modules or libraries of computational tools
- classes let us define whole new types. Think of a class as a set of objects (the *instances* of the class) and the operations defined on them.
  - You are now familiar with: int, float, Boolean, string, list, tuple, dictionary
  - **with class definitions you can create your own types**. Programs can be much clearer, easier to understand and maintain when written in terms of appropriate types and instances of those types

Instead of using, say, a list or dictionary to represent a person:

```
p = ['jim', 56, 'blue', 'professor']
p[1]                # access age
```

and using basic list operations to extract age, define and use a Person class and related operations

```
p = Person(…)
p.age()
p.eyecolor()
p.occupation()
```

Classes provide abstraction. We can use objects without knowing details of how data is stored

- documentation tells you how to use objects but doesn't need to tell you implementation details. In fact, the implementation details can be changed without you having to worry about it

```
# assumes personLists1 and 2 have age stored at index 1
def olderThan (personList1, personList2):
    if personList1[1] > personList2[1]:
        …

# assumes person1 and 2 are objects with age method
def olderThan(person1, person2):
    if person1.age() > person2.age():
        …
```

In the first example, the olderThan function needs to understand how a person is represented – as a list in which the second element contains the age.

In the second example data in the person objects might be stored internally using lists, dictionaries, or something else. *We don't know and don't need to know.*

Classes

Basic python types are actually themselves classes.

- list objects are instances of the list class
- the operations defined for a class are called methods.

You've been using methods via the dot notation:

```
[1,2,3].append(4)
```

- Earlier I suggested you think about such methods as strange function call syntax [1,2,3].append(4) → append([1,2,3],4)
- That is useful but if try it exactly like that, you'll get an exception

```
>>> append([1,2,3],4)
```

Methods *are indeed* functions – just special ones specific to a class. The list append method is defined *as part of* the definition of the list class.

- Execute help(list) in Python shell to see things defined for the list class
- Turns out you *can* directly call append in "plain" function style, if we use append's "full" name – list.append (the append function owned by the list class)

```
>>> list.append([1,2,3],4)
```

- Similarly, see help(int). + actually shorthand for __add__ method for integers.

```
>>> a = 3
>>>.a.__add__(4)
7
```

(more on these __foo__ functions later)

Defining classes

- In Python (and other languages) to define a **class**, you define object **attributes** (also often called **properties**) and the methods (operations) that can be invoked on objects (instances) ofthat class. General form:

```
class Myclass ():
    classAttribute1 = …
    …

    def method1(self, …):
        self.objectAttribute1 = …
        self.objectAttribute2 = …
        … computation in terms of properties and arguments passed to method…
        return …

    def method2(self, …)
        … computation in terms of properties and arguments passed to method ...
```

- Note: variable name self is a convention (standard practice/usage). The first argument to a method is always the object that invoked the method. It is *legal* to name it anything but please stick to standard practive – use 'self'

- Basic examples to see how it works: Cat, Dog classes
  - Simple attributes and methods
  - Initialization via __init__ method
  - Looping over objects calling methods of same name
  - Nice, informative print form
- Read Ch 15 and 17 (you can skip 16) for next time. We'll go through them carefully