CSC 108H: Introduction to Computer Programming

Summer 2012

Marek Janicki

Administration

- Assignment 1 grades will be up this weekend.
  - Automarker is taking longer to write than I thought.
  - Comments.
- Assignment 2 comments.
  - Check piazza for clarification.
  - You are responsible for good variable name choice and docstrings!
- Assignment 3 will be up this weekend.
  - Will be a two part assignment.
  - Test cases will be due earlier.
- No office hours this Monday.

Speed Review

- Two ways of judging speed of code.
- Hard testing the speed.
  - literally timing the time it takes code to run.
- Analysing the code and roughly seeing how it scales with input.
  - More generally used at a planning stage.
  - Used to choose between several possibly solutions.
  - Consider worst case of input.

Type Review

- So far we've seen lots of types.
  - int, bool, float, str, list, dict.
- A lot of these types have methods.
  - str.isupper, list.sort, dict.update, etc.
- This allows us to organise our code much more efficiently.
  - x.upper() is more readable than set_to_upper(x).
  - Allows us to save time.

Function and Module Review

- Python comes with some built-in functions.
- We make our own based on code that we want to reuse.
- We group similar functions in a file.
  - These can be shared with other users through modules.
  - So other people can build larger things ontop of our functions.

Types are useful

- Types allow us to separate our code conceptually.
  - Even though on some level it is all 0s and 1s, it's useful to imagine some of them representing integers and some of them representing strings.
- Often times when we design code, we imagine certain data structures to represent some meta-information.
  - It would be useful to encode this.

Classes

- Python allows us to build our own types.
- These are called classes.
- A class is analgous to a type.
- Keep in mind that types have instances.
  - Think of the difference between `str` and `'aaa'`.
  - Each type can have lots of possible instances.
  - When we create our own classes, they will have the same property.

Objects

- An object is an instance of a class.
- An object is to class as 'aaa' is to str.
- Objects allow for good chunking of code.

Class/Object syntax.

- To make a class we just do:

```
class Class_name(object):
    block
```

- `class` is a keyword and `object` is a type.
- Class names start with Capital letters by convention.
- To create objects or instances of Class_name we use:

```
x = Class_name()
```

What do we want from our classes?

- Need to store data.
  - like a string stores the actual values, or an integer has some value, or a list contains elements, etc.
- Need to be able to easily generate values based on the object.
  - Like dict.has_key, str.isupper, str.upper, etc.
- Need to be able to easily modify the data in the object.
  - Like dict.update, list.pop, etc.

Class data

- Each object can have instance variables.
  - And instance variable is some data that is associated with a class.
- These variables are created with the following syntax
  - `object_name.instance_var_name = value`
  - `some_patient.name = 'Marek'`
- Instance variables are variables, and must obey all of the variable naming conventions.
- They also obey all the same mutability rules.

Class data

- Being able to name instance variables means that we can now alter objects.
  - Objects are mutable!
  - If a function changes an object, that change persists after the function call.
- Instance variable names are important, because they contribute to the legibility of code.

Class Methods

- So far our objects are only convenient naming devices for related bits of data.
- We have no way of doing computation on them without writing specific functions to do it.
  - So we can to is_upper(x) but not x.isupper().
- To do the latter we need methods.
- These are put in the block of code under the class definition.

Class syntax redux

```
class Class_name(object):
    block
```

- The block contains methods.
- A method has the syntax:

```
def method_name(self, parameters):
    block
```

- Where `block` is any legal python code.
- What is self?

Self

- What is `self`?
- Imagine writing code for list.count.
  - Recall `L.count(1)` returns the number of occurrences of 1 in the list L.
  - So obviously the result depends on the values of L.
- If we just have `def count(x):` we have no way of determining the values of L.
  - `self` is a keyword that refers to the object that is being used to call the method.

Class methods

- self is always implicitly passed.
- This means we don't need to pass it explicitly.
- Consider having the following method:

```
class Patient(object):
    def set_age(self, age):
        self.age = age
```

- To call set_age we use `p1.set_age(10)`, not `p1.set_age(p1,10)`.

Class Methods vs. Functions

- Any method can be rewritten as a function with more than one variable.
- When to choose methods, and when to choose functions?
  - No hard and fast rule.
  - Generally, any task that has one 'core' object should be written as a method.

Class conventions.

- Class names start with upper case letters.
- Class methods and instances start with lower case letters.
- Method definitions should have docstrings just like function definitions.
  - Methods that you don't want users calling should begin with an underscore.
- Classes should have docstrings just like modules have docstrings that describe what the class does.

Break, the first.

Can we initialise data?

- Thus far we've needed to design a class, and then add all the data we want for each instance by hand.
- This is not necessary, we can create instances of classes with data already added.

Class Constructors.

- The special method __init__ is called whenever you try to create an instance of an object.
- Such methods are called constructors.

```
class Patient(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

- Called by `x = Patient("Joey", 10)`
  - Then our patient already has these two instance variables set and initialised.

Object Oriented-Programming

- OO programming is the idea that when you have a problem, the first step to solving it is to decide on what objects you will need to solve it.
  - In some sense it means putting the emphasis on the data structures needed to solve the problem.
- It means that the main solution to the problem tends to be some object that is initialised and has methods callsed.
- Three commonly stated advantages to OO programing - polymorphism, encapsulation, and inheritance.

Polymorphism

- Polymorphism is the idea of getting different types to behave in similar ways.
  - We saw that we can loop over lists and strings.
  - Also in labs we saw that we can loop over pixels.
  - Objects of different classes can be initialised in the same way.
- Objects allow polymorphism because there are 'special' methods that when implemented cause types to behave like native python types.
  - We won't really focus on this.

Class methods: Special Methods.

- __ indicates that the method is a special method.
- These are used to make our classes work more like Python's built-in types.
- For example:
  - `__str__` is used when printing
  - `__cmp__` is used to allow boolean operations.
  - `__add__` is used to allow the + operator.
  - `__iter__` is used to allow your type to be used in for loops.

Classes - Encapsulation

- One of the big benefits of classes is that they hide implementation details from the user.
- We call this encapsulation.
- A well designed class has methods that allow the user to get out all the information they need out of it.
  - This allows a user to concentrate on their code rather than on your code.
- This also frees you to change the internal implementation of the class.

Classes - Encapsulation

- Encapsulation is one of the reasons why docstrings are important.
  - If docstrings are poorly written, encapsulation not longer works properly.
  - And documentation may need to be rewritten if the code is rewritten.
- Encapsulating well means understanding the difference between what the code is representing, and how the code is written.

The Structure of Programming.

- We want our programs to be both reusable and extendable.
- Reusable means that other people can easily take our code and use it for their problems.
- Extendable means that it's easy to modify our code to handle new issues that come up.
- How do we resolve the tension between the two?

Classes - Inheritance

- We want a way to allow modifications to existing code, that don't alter the ability of existing code to run.
- One way we could do this is to write a new class that copies the old class plus has some new functions.
- This is a lot of work, especially if you decide to change the old class down the road.
- Also breaks encapsulation.

Classes - Inheritance

- Instead we can use Inheritance.
- Classes are allowed to inherit methods and variables from other classes.
- If class A inherits from class B, then class B is called the superclass, and class A the subclass.
- Classes inherit all of the methods and variables in the superclass.
- One can overwrite or add new methods in the subclass as appropriate.

Classes – Inheritance

- The syntax for creating subclasses is:
- `class Class_name(Superclass_name):`

  `block`
- Note that this means our previous class is a subclass of the class `object`.
- If you define a method with the same name as one in the superclass, you overwrite it.

Classes - Inheritance

- Inheritance is a really powerful tool that is easy to abuse.
- Inheritance should be used to represent 'is-a' relations.
  - So a PhysioPatient is a type of Patient.
  - A mammal is a type of animal.
  - A party is a type of event.
- When coming up on to a new problem, a common first step is to think about class structures and what objects you'll need.

Classes - Inheritance

- It may be that the best structure for a program has several different classes, only some of which inherit from eachother.
- It maybe that you just need a bunch of classes that all inherit from object,
- It maybe that you will have one super class from which everything inherits.
- But you should never

Classes - summary

- Classes are user-built types.
  - Objects are instances of those types.
  - These behave like every other type in python.
  - You can add objects to lists, dictionaries and so on.
- Classes have 3 main advantages.
  - polymorphism
  - encapsulation
  - inheritance