CSC 108H: Introduction to Computer Programming

Summer 2011

Marek Janicki

Administration

- Class is in BA1140 Next week.
- Midterms will be given out in the second break.
- The mean was a 28.8.
- Median was 31.
- Assignment 1 autotesting results have been mailed to your cdf e-mails.
- I am aware that Assignment 2 was too difficult, and the grading will reflect that.
- Assignment 3 comes out tomorrow or on the weekend.

The Structure of Programming.

- At the most basic level there is Machine Code.
  - We don't do this.
- Then we have lines of code in a programming language.
  - But often times we use the same code.
- Then we have functions on top of that.
  - We can reuse code, but often code is very use specific.
  - There are also built-in functions.

The Structure of Programming.

- So we can divide our code into Modules.
  - This means we don't need to evaluate massive amounts of code any time we need to get something done.
- Modules have their own functions.
  - Helps with separating programs.
- In a similar way, we saw that types have methods.

Classes

- Python allows us to build our own types.
- These are called classes.
- When we have a class, we can create instances/objects of that class.
- Recall the difference between a type (str, int) and a value ('this is a string', 10).
- This is analogous to the difference between a class and an object.

Classes

- 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()
```

Class methods.

- So far our class objects can't do a whole lot.
- One way to make them more useful is to add methods to class objects.
- We do this by putting them in the block of code under the class name.
- But we want our methods to work on each individual class object.
- Ie. when we call 'aaa'.isUpper() we want it to only work on 'aaa' not on all strings.
- How can we get our method to refer to our class object rather than to all classes?

Class methods.

- To solve this issue we use the keyword `self`
- Say we have a class Patient that is meant to store information about patients in a hospital.
- We may want to update the patient's age.
- To do this we need a method `set_age`.

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

Class methods.

- Note that these are methods, so that if we have a patient `p1` to set his age we use `p1.set_age(10)` and not `p1.set_age(p1,10)`
- Self should always be the first parameter in a method but it is passed for free when we call the method.

Class Constructors.

- There is also away to set class variables when you construct class instances.
- The special method __init__ is called whenever you try and construct an instance of an object.

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

- Called by `x = Patient("Joey", 10)`

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.

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.
- Classes should have docstrings just like modules have docstrings that describe what the class does.

Break, the first.

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.

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(Subclass_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 Surgery Patient 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.

Break, the second

Midterm Review

July 7 2011