CSC 108H: Introduction to Computer Programming

Summer 2011

Marek Janicki

Administration

- Office hours
  - Held in BA 2200 at T12-2, F2-4
  - If this changes, will be posted on announcements.
- Twice as many people in Thursday tutorials.
  - Consider switching if you can.
- Class in BA1170 on June 23rd and July 14th.
- Website typo in info sheet, there is no trailing h.
  - A redirect has been added.
  - My e-mail is quellan@cs.toronto.edu
  - Not quellan@cdf.toronto.edu

Assignment 1

- This is a short and simple assignment.
- It has been posted.
- Needs to be done on your own.
- You can write it wherever, but before you submit, make sure that it runs on the CDF machines.
- No questions about it will be accepted after June 2nd.

Programs can be adaptive.

- Last time we compared programs to recipes.
  - Not entirely accurate.
- Programs can behave differently depending on the situation.
  - We saw a very brief snippet of this last week.

Booleans: A new type.

- Can have two values True, False.
- Have three operations: not, and, or.
- not changes a True to a False and vice versa.
- and returns False unless all the arguments are True.
- or returns True unless all the arguments are False.

Truth Tables

- A way of representing boolean expressions.

x	y	not x	not y	x and y	x or y
True	True	False	False	True	True
True	False	False	True	False	True
False	True	True	False	False	True
False	False	True	True	False	False

What if we want to adapatively assign Boolean values.

- We can use relational operators.
  - <,>,<=,>=,!=, ==
- These are all comparison operators that return True or False.
- == is the equality operator.
- != is not equals.

Boolean Expressions and Representation

- Can combine boolean operators (and, or, not) and relational operators (<,>,etc) and arithmetic operators (+,-,*, etc).
  - 5+7<4*3 or 1-2 >2-4 and 15==4 is a legal expression.
  - Arithmetic goes before relational goes before boolean.
- False is represented as 0, and True is represented as 1.
  - Can lead to weirdness. Best to avoid exploiting this.

Short Circuit Evaluation

- Python only evaluates a boolean expression as long as the answer is not clear.
  - It will stop as soon as the answer is clear.
- This, combined with the nature of boolean representation can lead to strange behaviour.
- Exploiting these behaviours is bad style.

How to use boolean variables

- Recall that we want to make our code adaptive.
- To use boolean variables to selectively execute blocks of code, we use if statements.

If statement

- The general form of an if statement is:

  if *condition*:

    block

- Example:

  if grade >=50:

    print “pass”

If statement

- The general form of an if statement is:

  if *condition*:

    block

- The *condition* is a boolean expression.
- Recall that a block is a series of python statements.
- If the *condition* evaluates to true the block is executed.

Other Forms of if statement

- If we want to execute different lines of code based on the outcome of the boolean expression we can use:

```
if condition:
    block
else:
    block
```

- The block under the else is executed if the condition evaluates to false.

More general if statement.

```
if condition1:
    block
elif condition2:
    block
elif condition3:
    block
else:
    block
```

- Python evalutates the conditions in order.
- It executes the block of the first (and only the first) condition that is true.
- The final else is optional.

Style advice for booleans.

- If you are unsure of precedence, use parentheses.
  - Will make it easier for a reader.
  - Also use parentheses for complicated expressions.
- Simplify your Boolean expressions.
  - Get rid of double negatives, etc.

Break, the first

Review of Functions

- We started by looking at some of python's native functions.
- We saw how to call functions.
- Saw how to define our own.

Why functions?

- Allow us to reuse bits of code, which makes updating and testing much easier.
  - Only need to test and update the function, rather than every place that we use it.
- Chunking! Allows us to parse information much better.
  - Human mind is pretty limited in what it can do.
  - Function names allow us to have a shorthand for what a function does.

Return vs. Print

- Recall that functions end if they see a return statement, and return the value of the expression after the keyword return.
  - If there is no return statement, the function returns None.
- We've also seen snippets of the print statement.
  - Print takes one or more expressions separated by a comma, and prints them to the screen.
- This is different than a return statement, but looks identical in the shell.

Multiple Function calls

- Sometimes we want to have functions calling other functions.
  - f(g(4))
- In this case, we use the 'inside out' rule, that is we apply g first, and then we apply f to the result.
- If the functions can have local variables, this can get complicated.

How does python choose variables?

- Python has local and global variables.
- Local variables are defined inside of functions, global variables are defined outside of functions.
- What happens if a local variable is the same as a global variable?

Generally python will...

- First, check local variables defined in a function.
- Then check local variables in an enclosing function.
  - That is for f(g(4)) it will check g's local variables first, and then f's local variables.
- Then it will check global variables.
- Finally it will check built-in variables.

How to think about scope.

- We use namespaces.
- A name space is an area in which a variable is defined.
- Each time we call a function, we create a local namespace.
- We refer to that first, and go down to the enclosing functions name space or global namespace as necessary.

Style conventions for Functions.

- As we've seen, python allows us to be somewhat careless in where we initialise and call variables.
- Exploiting this is bad style.
  - It makes code hard to read and prone to errors.

Designing Functions

- Need to choose parameters.
  - Ask “what does the function need to know”.
  - Everything it needs to know should be passed as a parameter.
  - Do not rely on global parameters.
- Need to choose whether to return or not to return.
  - Functions that return information to code should return, those that show something to the user shouldn't (print, media.show(), etc).

Break, the second.

Function Documentation

- Recall that we can use the built-in function help() to get information on functions or modules.
- We can do this on functions that we've defined as well, but it doesn't give much information.
- We can add useful documentation with docstrings.
  - A docstring is surrounded by ''' and must be the first line of a module or function.

Docstrings

- If the first line of a function or module is a string, we call it a docstring.
  - Short for documentation string.
- Python saves the string to return if the help function is called.
- Convention: Leave a blank line after but not before a docstring.
- All functions should have docstrings.

Why Docstrings?

- If you write the docstring first, you have an instant sanity check.
  - That is, you can be sure that the function is doing what you want it to do.
- Makes portability and updating easier.
  - Allows other people to know what your functions do and how to use them, without having get into the code.
  - Allows for good chunking.

Writing Good Docstrings.

- '''A sunset module.'''
- '''Changes into a sunset.'''
- These are terrible docstrings.
  - They are vague and ambiguous. The don't tell us what the function expects or what it does.
- How can we make it better?

Writing Good Docstrings.

- Describes what a function does.
- '''Changes into a sunset.'''
- '''Makes a picture look like it was taken at sunset.'''
- '''Makes a picture look like it was taken at sunset by decreasing the green and blue by 70%.'''

Writing Good Docstrings.

- Describes what a function does.
- '''Changes into a sunset.'''
- **'''Makes a picture look like it was taken at sunset.'''**
- **'''Makes a picture look like it was taken at sunset by decreasing the green and blue by 70%.'''**

Writing Good Docstrings.

- Does not describe how a function works.
  - More useful for chunking, and it's unnecessary information if we're using the function.
- '"Makes a picture look like it was taken at sunset."'
- '"Makes a picture look like it was taken at sunset by decreasing the green and blue by 70%."'

Writing Good Docstrings.

- Does not describe how a function works.
  - More useful for chunking, and it's unnecessary information if we're using the function.
- **'''Makes a picture look like it was taken at sunset.'''**
- '''Makes a picture look like it was taken at sunset by decreasing the green and blue by 70%.'''

Writing Good Docstrings.

- Makes the purpose of every parameter clear and refers to the parameter by name.
- '''Makes a picture look like it was taken at sunset.'''
- '''Takes a given picture and makes it look like it was taken at sunset.'''
- '''Takes a picture pic and makes it look like it was taken at sunset.'''

Writing Good Docstrings.

- Makes the purpose of every parameter clear and refers to the parameter by name.
- '''Makes a picture look like it was taken at sunset.'''
- '''Takes a given picture and makes it look like it was taken at sunset.'''
- **'''Takes a picture pic and makes it look like it was taken at sunset.'''**

Writing Good Docstrings.

- Be clear if a function returns a value, and if so, what.

  Consider average_red(pic)
- '''Computer the average amount of red in a picture.'''
- '''Returns the average amount of red (a float) in a picture pic.'''

Writing Good Docstrings.

- Make sure to explicitly state any assumptions the function has.

  Def decrease_red(pic,percent)

- '''Decreases the amount of red per pixel in picture pic by int percent. percent must be between 0 and 100.'''

Writing Good Docstrings.

- Be concise and grammatically correct.
- Use commands rather than descriptions.
- '''Takes a picture pic and makes it appear as it if was taken at sunset.'''
- '''Take picture pic and make it appear to have been taken at sunset.'''

Writing Good Docstrings.

- Describes what a function does.
- Does not describe how a function works.
- Makes the purpose of every parameter clear and refers to the parameter by name.
- Be clear if a function returns a value, and if so, what.
- Make sure to explicitly state any assumptions the function has.
- Be concise and grammatically correct.
- Use commands rather than descriptions.

Boolean Docstrings.

- def: is_odd(x):

  return (x%2)==1

- The docstring for this might look like '"Return True if int x is odd, and False otherwise."'
- Commonly shortened to:
  - '"Return True iff int x is odd.

IFF

- iff stands for if and only if.
- So in fact we wrote:
- '"Return True if int x is odd and only iff int x is odd."'
- We didn't specify what to do if x is not odd.
- But for boolean functions, it is understood that we are to return False if we're not returning True.

Writing Good Docstrings.

- Docstrings do not include definitions or hints.
- The docstring for sqrt is not:

  '''Return the sqrt of (x). The sqrt of x is a number, that when multiplied by itself evaluates to x'.

- Is it simply:
  - Return the square root of x.