CSC 108H: Introduction to Computer Programming

Summer 2012

Marek Janicki
Administration

- Help Centre is open.
  - BA 2270 M-R 2-4.
- CDF is closed from M Jun 4th 5pm to 11am T June 5th.
- Exercise 1 deadline extended to Sunday.
- Exercise 2 will be posted before next Lecture.

May 31 2012
Last Week

• More Functions.
  • `print` makes the computer show something on the screen.
  • `return` ends a function and causes it to return the value of the expression.

• Function documentation.
  - The first line after a function should be a description of what it does enclosed in `'''`.
  - Returned by `help(function_name)`.

• Function design.
Function Review

- What is this function missing?
  ```python
def foo():
    return 10
  ```

- What gets printed to the screen, and in what order?
  ```python
def foo(x):
    print x + 10
    return 15

y = 12
foo(y)
print foo(y+4)
```
Function Review

• What is this function missing?
  ```python
def foo():
    '''NoneType -> int
    returns 10.'''
    return 10
  ```

• What gets printed to the screen, and in what order?
  ```python
def foo(x):
    print x + 10
    return 15

y = 12
foo(y)
print foo(y+4)
```

22
26
15
Last Week

- **Scope.**
  - Variable scope is used to determine which variable is used when there are multiple variables with the same name.
  - Variables can be global and local.
    - local variables are defined within functions.
    - global variables are defined in the body of code.
  - To determine which variable is used if there are multiple function calls we use a call stack.
    - Each time there is a function call, a new namespace is created on the call stack.
Scope Review

What does the call stack look like at the indicated points?

```python
def foo(x):
    if x < 10:
        return 10
    else:
        return goo(x-5)

def goo(x):
    if x < 10:
        return 10
    else:
        return foo(x-5)

x = 20
foo(x)
```
Scope Review

- What does the call stack look like at the indicated points?
  
def foo(x):
      if x < 10:
          return 10
      else:
          return goo(x-5)

  def goo(x):
      if x < 10:
          return 10
      else:
          return foo(x-5)

x = 20
foo(x)
Scope Review

• What does the call stack look like at the indicated points?

def foo(x):
    if x < 10:
        return 10
    else:
        return goo(x-5)

def goo(x):
    if x < 10:
        return 10
    else:
        return foo(x-5)

x = 20
foo(x)

May 31 2012
Last Week

- Booleans.
  - New type.
  - Can be True or False.
  - Can compare booleans with and, or, not.
  - Can use relational operators to generate booleans.
    - <, >, <=, >=, !=, ==.

- Conditionals.
  - Used to selectively execute blocks of code based on booleans.
    - if, else, elif.
Booleans

What do these expressions evaluate to?

(bool(x) and not(bool(x))

(True or False) and bool(-10)

True != False

What values does x need to execute each print statement?

x = ?

if (x == 50):
    print 'a'

elif (x < 50):
    print 'b'

elif (x > 25):
    print 'c'

else:
    print 'd'
Booleans

What do these expressions evaluate to?

(bool(x) and not(bool(x)))

False

(True or False) and bool(-10)

True

True != False

True

What values does x need to execute each print statement?

x = 50, 24, 60, NA

if (x == 50):
    print 'a'

elif (x < 50):
    print 'b'

elif (x > 25):
    print 'c'

else:
    print 'd'
Using text

- So far we've seen three types:
  - ints, floats, and booleans.
- Allow for number manipulation and logic manipulation
- Don't allow for text manipulation.
- Text manipulation needs a new type - strings.
  - A string is a sequence of characters.
  - A character is a single letter/punctuation mark/etc.
Strings

- **Two types:** `str` and `unicode`.
  - We'll use `str` in this course.
  - It contains the roman alphabet, numbers a few symbols.

- **Use `str` to refer to the type in docstrings.**
  - `'NoneType -> str'`

- Strings are denoted by single or double quotes.
  - "This is a string"
  - 'This is not'
  - "" is an empty string.
String operations

- Strings can be 'added'.
  - We call this concatenation.
  - "str" + "ing" results in "string".
- Can also be multiplied, sort of.
  - You can't multiply a string with itself, but the multiplication operator functions as a copy.
  - So "copy"*3 results in "copycopycopy".
- None of the other arithmetic operators are defined for strings.
  - so /, -, **, and % generate errors.
String questions

- Which of the following expressions evaluate to legal strings?
  - 'abab'
  - '" abababe''
  - '"ababab"'
  - 'avvrr' + "bab"
  - 4 + 'abb' + "ab"

- "a" + "b" - "b"
- 3 * "abab" + "vbr"
- '"abe"" * 99
- '""bb"" * '99'
- 'string'
- "string"
String questions

- Which of the following expressions evaluate to legal strings?
  - 'abab''
  - " abababe'  
  - ""ababab"
  - 'avvrr' + "bab"
  - 4 + 'abb' + "ab"

- "a” + “b” - “b”
- 3 * “abab” + “vbr”
- "'abe'"" * 99
- ""bbb"" * '99'
- 'string'
- “string”
String operations

• Can also compare strings using relational operators.
  • So two strings can be compared using <,>, !=, etc.
  • If the letters are all upper case or all lower case, the order is lexicographic (dictionary style).
  • Upper case letters are 'smaller' than lower case letters, which can cause odd behaviour.
    - 'aaa' < 'ab'
    - 'aaa' < 'aB'
• Can compare punctuation marks, but there's no intuition for the results.
String operations

- Can check if substrings are in a string using `in`.
  - `possible_substring in big_string` returns `True` iff `possible_substring` is in `big_string`.
  - `possible_substring` needs to be contiguously within `big_string` for this to return `True`, it will return `False` otherwise.

- Long strings that span multiple lines can be made using `''`
  - Note that this relates to docstrings.
Escape Characters

- Denoted by a backslash, they indicate to python that the next character is a special character.
  - \n - a new line
  - \' - a single quote
  - \" - a double quote
  - \\
  - a backslash
  - \t - a tab.
String functions

- `len(string)` will return an int that is the number of characters in the string.
- `ord(char)` will return the integer code of that character.
- `chr(x)` will return a character that corresponds to the integer `x`.  
  - `x` should be between 0 and 255.
Type Conversions

- If we want to add a number or boolean to a string, we need to convert it to a string first.
- `str(x)` converts `x` to a `str`.
- This is automatically done when `print` is used.
- Strings can be converted to booleans.
  - `False` iff string is empty.
- Strings of numbers can be converted to floats or integers.
- Strings of numbers with one decimal point can be converted to floats.
String Questions

- What do the following strings look like?
- '\n\n\n Hi, things'

- str(True) + '\n" This is true'

- str(34) + '" +str(44)

- str(bool(''))
String Questions

- What do the following strings look like?
- '\n\n\n Hi, things'
  
  \ Hi, things"

- str(True) + '\n" This is true'
  ""True
  ""This is true"

- str(34) + '\" +str(44)  ""34'44"

- str(bool(''))  ""False""
Mixing strings with other types

- Print can display mixed types.
  - They must be separated with a comma.
    - `print "string", x, " ", real_num`
- Can be awkward.
  - `print "Person", name, "has height", height, "age", age, "weight", weight`
String formatting

- Can use special characters to tell python to insert a type into a string.
- `print “My age is %d.” % age`
- The `%d` tells python to take age, and format it as an integer.
- `%s` says to take a value and format it as a string.
- `%f` says to take a value and format it as a float.
- `%.2f` says to pad the float to 2 decimal places.
Multiple variables

• What if we want multiple variables in our string?
  • print “Person”, name, “has height”, \height, “age”, age, “weight”, weight

• We put them in parentheses separated by commas.
  • print “Person %s has weight %.2f \ and age %d and height %d.” \% (name, weight, age, height)
Break, the first
Break Question.

- Given the variables age, height, name; write out formatted strings that evaluate to:
  - 'My name is <name>.'
  - 'My name is <name> and I am called <name>.'
  - 'I am <age> years old and this tall: <height>,'
  - ""My name is <name>
    I am <age> years old.""
  - 'My height is %s <height> %s'
- Do not use "".
Break Question.

- Given the variables age, height, name; write out formatted strings that evaluate to:
  'My name is %s' % name
  'My name is %s and I am called %s' % (name, name)
  'I am %d years old and this tall: %d' % (age, height)
  'My name is %s
I am %s years old.' % (name, age)
  'My height is %s %s %s' % ('%s', height, '%s')

- Do not use "".
User input

- Thus far, the only way we've had of giving input to a program is to hardcode it in the code.
- Inefficient and not user-friendly.
- Python allows us to ask for user input using `raw_input()`.
- Returns a string!
  - So it may need to be converted.
Modules

- Sometimes we want to use other people's code.
- Or make our own code available for use.
- But we don't want to mix our code with that of others.
- Modules allow us to do this.
- A Module is a group of related functions and variables.
  - Each file in python is a module.
Using modules

- To use a module, one needs to **import** it.
  - At the top of a file by convention.
- Importing a module causes python to run each line of code in the module.
- To use a function in a module one uses:
  
  ```python
  module_name.function_name()
  ```
- We can also run a module. Then we just use:
  ```python
  function_name()
  ```
Using modules

- Note that we can run files, and each file is a module.
  - If we are just running a file, then we only use the function name, not `module_name.function_name`.
  - Functions defined within a module are local functions, in the same way that variables within a function are local variables.
  - Global variables within a module can be accessed by `module_name.variable_name`.
    - Rare that this is necessary.
Which are legal?

import foo
foo.foo(12)
goo(12)

import foo
def goo(x):
    return x
foo.foo(12)
goo(12)

import foo
def goo(x):
    return x
foo(12)
goo(12)
Which are legal?

import foo
foo.foo(12)
goo(12)

import foo
def goo(x):
    return x
foo.foo(12)
goo(12)

import foo
def goo(x):
    return x
foo(12)
goo(12)
Importing Modules

• When a file is imported, every line in the file is run.
  • It is just function definitions this doesn't cause much trouble.
  • But it can be annoying if there is code that you don't care about or testing code in the module.
__name__

- In addition to variables that are defined in the module, each module has a variable that is called __name__.

- If we import a module called module_m, then
  
  module_m.__name__ == "module_m"

- But if we run a module, then
  
  __name__ == "__main__"

- Recall that if we are running a module, we don't need the module name as a prefix.
if __name__ == '__main__'

• It is very common to see modules that have the following code:
  
  if __name__ == '__main__':
      block

• The block will be executed if the module is being run.

• A useful place to put testing code.
Another way to import things.

- `from module_name import fn_name1(), fn_name2()`
  - Will import `fn_name1` and `fn_name2`
  - These functions are referenced by just `fn_name1()`
- Can also use `*` as a wildcard to import all the functions.
  - `from module_name import *`
- What if two modules have a function with the same name?
- The most recent one stays.
Break, the second.
Break, the second.

- When will these modules print 'running'

```python
if __name__ == '__main__':
    print 'running'

print 'running'

if __name__ == '__main__':
    print 'running'
```

May 31 2012
Break, the second.

- When will these modules print 'running'

  ```python
  if __name__ == '__main__':
      print 'running'
  ```

- All the time.

  ```python
  __name__ == '__main__'
  if __name__ == '__main__':
      print 'running'
  ```

- All the time
Methods

- We've seen that modules can have their own functions.
- A similar thing is true of values.
- Values contain functions that assume one of the inputs is the value. We call these methods.
- These are called by `value.fn_name()`.
- Or, if we've assigned a value to a variable we can use `variable_name.fn_name()`.
- We can call `help(type)` to figure out what methods a type has available to it.
String methods

• Can find them by using `help(str)`.
• Useful ones include:
• `s.replace(old, new)` - a copy of `s` with all instances of `old` replaced by `new`.
• `s.count(substr)` – return the number of instances of `substr` in the string.
• `s.lower()` - shift to lower case letters.
• `s.upper()` - shift to capitalised letters.
• None of these change `s`.
String methods

- `s.strip()` - returns a copy of `s` with leading and trailing whitespace removed.
  - Note, doesn't touch middle whitespace.
  - Whitespace refers to spaces, tabs and new lines.
  - Essentially, anything that doesn't contain a visible character.

- `s.strip(chars)` - strips all of the characters in the given string instead.
String method questions

- What do the following statements evaluate to?
- \( x = 'AAAAbb bb' \)
- \( x.count('b') \)
- \( x.count('B') \)
- \( x.count('Ab') \)
- \( x.lower() \)
- \( x.strip() \)
- \( 'bb'.replace('b','bb') \)
String method questions

- What do the following statements evaluate to?
  - x = ' AAAAbb bb'
    - None
  - x.count('b')
    - 4
  - x.count('B')
    - 0
  - x.count('Ab')
    - 1
  - x.lower()
    - ' aaaabbb bb'
  - x.strip()
    - 'AAAAbb bb'
  - 'bb'.replace('b','bb')
    - 'bbbb'
Getting method information

- Most direct way is to use help().
- But help isn't searchable. Can use dir() to browse.
  - Sometimes you know what you want, and you think it might already exist.
- An alternative is to check the standard library:
  - http://docs.python.org/library/
  - Being able to browse this is useful skill.
- Modules are found in:
  - http://docs.python.org/py-modindex.html
Remember!

- Functions belong to modules.
- Methods belong to objects.
  - All of the basic types in python are objects.
  - We will learn how to make our own later.
  - This is covered in greater detail in 148.
- `len(str)` is a function
- `str.lower()` is a method.
- Subtle but important distinction.
Lab Review

• Next weeks lab covers Booleans and conditionals.

• You need to:
  • Be comfortable with using boolean operators (and, or, not) on booleans.
  • Using if statements to selectively execute blocks of code based on the value of boolean expressions.