INTRODUCTION TO PROGRAMMING
WEEK 2 - HELLO WORLD!

USING PYTHON

There are 2 main ways to interact with python - through a script and through the interactive mode. Please submit all homework as scripts and not as an interactive session.

Running a Python Script

In the previous lecture (and in HW1), we saw how to write a python script and run it. The command is as follows:

```
python <filename>
```

Example:

```
python hw11.py
```

A few things to note here:

- You need to be in the same directory as the python script to do this (otherwise, run the script with a relative/absolute path...`python hw1/hw1/hw1_2/hw11.py`
- Python scripts are lines of code executed from top to bottom

Interactive Mode

Python also sports an interactive mode that allows you to run individual lines of python code one at a time or in a conditional block(more on that in week 3). For now, all you need to know is that if you invoke the command `python`, it will open up the interactive mode and allow you to run individual lines.

```
alberthwang@gswitch:~$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello world'
hello world
```

A few things to note here:

- The moment you close the prompt (Hold `Control` and Hit `D`) the code you ran is lost and not saved anywhere
- The three carrots >>> are the python prompt - you enter lines of python at this prompt
- The interactive mode should be **used for testing purposes only**
- All homework submitted needs to come in the form of a script not as lines from your

interactive mode

WHY HELLO WORLD?

One time-honored tradition in computer programming is for a programmer to always print “Hello world!” in the first program that they write in a new language. Who are you actually addressing? Who knows.

COMMENTS

A comment is a note that a programmer can leave in the program to explain it. Comment syntax is different in every language, but in python, single line comments are delimited with a `#`. Try adding this:

```
# This won’t print!
```

To add a comment with multiple lines, use “”” ”””

```
“””
  This program was built to allow user to easily
  access data files associated with registration
  and person information.
”””
```

Try adding these comments to your helloworld program and run it again! What is the result? What are the use cases for comments?

VARIABLES

From high school algebra, you should remember the concept of a variable.

```
x = 2
y = 3
z = x + y

Solve for z.
```

In computer science, variables work the same way. Here is the same example in python:

```
x = 2
y = 3
z = x + y
```

```
print z
```

You declare variable names and associate them with values that you can perform operations upon. However, unlike basic algebra, in computer science variables can not only represent simple data types (primitives) like integers but also more complex objects. For instance a variable, `my_dog` can point to a dog object that can do things like `bark()`, `roll_over()`, `wag_tail()`, etc. More on that later (much later).

PRIMITIVES

Primitives are the some of most basic of computer programming data representations. They are the fundamental building blocks of more complex representations and are a great place to start learning programming. Some primitives include `char, int, double, float,` etc. While we won't hit on all of them today, we will learn about the most commonly used in python.

ASSIGNING PRIMITIVES TO VARIABLES

The assignment operator or = basically tell us that the variable on the left of the operator is to be assigned to the object on the right of the operator. This should be already be a familiar concept to you.

What will happen when you run this program?

```
my_str = 'hello, world!'
print my_str
```

What about now?

```
my_str = 'hello, world!'
my_str2 = 'hello there!'
my_str = 'hello, albert!'
print my_str
```

Next, let's go through some primitive types that are extremely important in Python.

STRINGS

Strings are sets of characters strung together, usually forming words or sentences. In some languages, strings are considered primitives, in others they are considered more complex objects. Some examples of strings are: `'hello'`, `'world'`, `'hello, world!'`, etc.

- A **substring** is any part of a string. For instance, `'he'` is a substring of the string `'hello'` (the first 2 characters).
- In computer science, strings are always demarcated with quotes `''`. This is important

because it distinguishes string literals from variables. For instance `'hello'` and `hello` are two different things (a string and a variable). This would be legal - `hello = 2`, while this `'hello' = 2` would be non-sensical.

- In python, strings can be declared using single or double quotes, but the Google style guide prefers that you use single quotes, as in `'hello'`
- As you have seen in the previous weeks, printing something shows the value in the prompt, so:
    - `print 'hello, world!'`
    - This displays `hello, world!` in your command prompt

ESCAPING SPECIAL CHARACTERS

What happens if you want to print this:

```
I really wonder about this so called 'cheese' in the cafeteria...
```

This will cause your computer to get quite confused, after all, the quotes are used to show where a string begins and ends. As a result, you will need to "escape" the characters so that the computer knows to print literal quotes - `''` to the screen.

The way to do this is with backslashes:

```
print 'I really wonder about this so called \'cheese\' in...'
```

CONCATENATION

Concatenation is the process of qualitatively joining two strings. This is one of those things that might seem intuitive but need to be spelled out explicitly for the computer. For instance, when I run this, what will display?:

```
new_str = 'one' + 'two'
print new_str
```

If you answered `'one two'` or `'three'` or `3`, you are wrong. The correct answer is `'onetwo'`

In python, concatenating two strings is as simple as using the addition sign. `'hi' + ' ' + 'there'` equals `'hi there'`.

You can also concatenate with variables as well. For example:

```
name = 'albert'
greeting = 'hi' + ' ' + 'there' + ' ' + name + '!'
print greeting # hi there albert!
```

STRING FUNCTIONS

Here are some extremely useful string functions for manipulating strings.

Substring

Substrings work exactly as they sound, `'super'` is a substring of the string `'superfly'` because `'super'` is a part of `'superfly'`. To get a substring, use square brackets and specify either the specific character place or a range of characters that you want to see from the string.

Specific character:

```
old_str = 'superfluous'

new_str = old_str[0]
print new_str # 's'

new_str = old_str[3]
print new_str # 'e'
```

Notice here, that when I say `old_str[0]` I am asking for the **first** character of the string. The reason for this is because **in computer science, counting always begins with 0 (0,1,2,3...).** When I say `old_str[3]` I am asking for the **fourth** character of `new_str`.

Range:

To create a substring from a range of characters in a string, use the following convention:

```
<string variable>[<first character position>: <+1 character over last in range>]

old_str = 'superfluous'

new_str = old_str[0:2]
print new_str # 'su'

new_str = old_str[2:6]
print new_str # 'perf'
```

Here are a couple handy shortcuts:

```
old_str[3:] # character position 3 to the end - 'erfluous'
```

```
old_str[:5] # From the beginning to character position 4 - 'super'
```

Replace()

The replace function replaces a substring of a string with another substring. Here are some examples

```
old_str = 'superfluous'
new_str = old_str.replace('super', 'SUPA')
print new_str # 'SUPAfluous'
```

This works on any character or set of characters including punctuation and individual characters.

```
old_str = 'superfluous!'
new_str1 = old_str.replace('u', 't')
new_str2 = old_str.replace('!', '?')

print new_str1 # 'stperfltots'
print new_str2 # 'superfluous?'
```

Lower() and Upper()

`lower()` and `upper()` make strings lowercase and uppercase respectively.

```
old_str = 'SuperFluous'
print old_str.lower() # 'superfluous'
print old_str.upper() # 'SUPERFLUOUS'
```

Strip()

`strip()` is a super-useful function that allows you to remove white space before and after a string.

```
old_str = ' superfluous '
old_str = old_str.strip()
print old_str # 'superfluous'
```

This function comes in handy when working with user input. Which is what we are covering next!

BASIC USER I/O WITH RAW_INPUT

A key element to any useful computer program is the ability to interact with the user of the program. To prompt the user to input a value and assign the value to a variable, use the raw_input function like so:

```
<user response variable> = raw_input(<string prompt>)
```

Try running the below example to see how it works:

```
name = raw_input('Your name: ')
print 'Hi ' + name
```

Raw Input takes the user input (from the prompt) and converts it to a string for use in the program.

INTEGERS

Most of you should be familiar with the concept of an integer, so I won't go into here. Try running a few of these operations below:

- `int1 = 3, int2 = 4, int3 = int1 + int2, print int3`
- `int1 = 3, int2 = 4, int3 = int1 * int2, print int3`
- `int1 = 3, int2 = 4, int3 = int1 + 10, print int3`

MODULUS

One operation you might not be familiar with is the modulus operator (`%`). Try running the following sets of code and see what you get:

```
int1 = 10
int2 = 3
int3 = int1 % int2
print int3 # 1

int1 = 11
int2 = 3
int3 = int1 % int2
print int3 # 2

int1 = 12
int2 = 3
int3 = int1 % int2
```

```
print int3 # 0

int1 = 13
int2 = 3
int3 = int1 % int2
print int3 # 1
```

What is the modulus operator?

Modulus returns the remainder if the first integer is divided by the second:

```
10 % 3 # 1
11 % 3 # 2
12 % 3 # 0
13 % 3 # 1
```

What use cases are there for the modulus operator?

This is good for processes that need to be iterated every n-th time where n is an integer (think of alternating colors on table rows).

CASTING

Try running the following:

```
str1 = 'I have'
int1 = 99
str2 = 'bottles of beer on the wall'

print str1 + ' ' + int1 + ' ' + str2
```

This *should* print `I have 99 bottles of beer on the wall`, but what goes wrong?

What we have here is a classic type error. While it might make sense (in your mind) to add 99 and "bottles of beer" to get "99 bottles of beer", python refuses to do this because there are two differnet object types here - integers and strings. It's not sure if you want it to convert the string "bottles of beer" into a numeric representation and then add it to the integer 99 or to convert 99 into a string and concatenate it with "bottles of beer" to get a new phrase.

To fix this, we must do something called *casting*. Essentially what this means is converting the data type of a variable or value. To cast, just invoke the `str()` or `int()` functions which will return an object of `String` of `Integer` data type (respectively).

```
str(3) # This will return '3'
int('3') # This will return 3
```

Now try this:

```
str1 = 'I have'
int1 = 99
str2 = 'bottles of beer on the wall'

print str1 + ' ' + str(int1) + ' ' + str2
```

What did we do differently? Why did this work?

Can you guess how we would fix this?

```
str1 = '1'
int1 = 99

# This should print 100
print str1 + int1
```

If you said this, then you are correct:

```
str1 = '1'
int1 = 99

#This should print 100
print int(str1) + int1
```