Homework 1 Problem 1 `hw1pr1.py` (Lab)

Copied from: https://www.cs.hmc.edu/twiki/bin/view/CS5/Lab1A on 3/20/2017

Welcome to Lab 1!

Please be sure that you've signed one of the attendance pages for this week's lab!

There are two parts to this lab:

- the first part is `hw1pr1.py`: you'll gain experience splicing and interacting with Python data
- the second part is `hw1pr2.py`: you'll write a number of *functions*, the fundamental building blocks of software

Trying out the Python interpreter (or *shell*) — ipython!

There's no file nor anything to hand in for this part -- just open ipython's shell by typing `ipython` at the command prompt and try out the Python commands below...

You can tell if you're in the ipython shell by the prompt: `In [n]` (where n is some number) or, on its own line,

```
In [1]:
```

Note that you don't have to type that prompt - it's already there!

Using VS Code's terminal

The text editor many of us are using, VS Code, has a built-in terminal. To use it, go to the *View* menu and then choose *Integrated Terminal*. The keyboard shortcut is control-` (control-backtick). Once you've opened the built-in terminal, you can type `ipython` to start Python.

Using ipython's numbered prompts

The numbered prompts are great, because you can access all inputs (as strings) and outputs (as whatever type they produce) via their numbers. To see this, try these examples

```
In [1]: 6*7
Out[1]: 42

In [2]: Out[1]*2
Out[2]: 84

In [3]: In[1]*2
Out[3]: '6*76*7'
```

Cool!
Python's `eval` will evaluate strings. Try `eval( Out[3] )` in the example above!

However, our lab guidelines won't necessarily match which number you're currently on, so **don't worry if your prompt number doesn't match ours - *experiment*!**

Arithmetic with numbers, lists, strings, booleans, ...

To get started, try a few arithmetic, string, and list expressions in the Python interpreter, e.g.,

```
In [1]: 40 + 2
Out[1]: 42

In [2]: 40 ** 2
Out[2]: 1600

In [3]: 40 % 7    # 40 "mod" 7
Out[3]: 5

In [3]: 40 // 11    # integer division
Out[3]: 3

In [4]: 'hi there!'
```

```
Out[4]: 'hi there!' # (notice Python's politeness!)

In [5]: 'who are you?'
Out[5]: 'who are you?' # (though sometimes it's a bit touchy.)

In [6]: L = [0,1,2,3] # You can label data (here, a list) with a
name (here, the name L)
(no response from Python)

In [7]: L
Out[7]: [0,1,2,3] # You can see the data (here, a list) referred
to by a name (here, L)

In [8]: L[1:]
Out[8]: [1,2,3] # You can slice lists (here, using the name L)

In [9]: L[::-1]
Out[9]: [3,2,1,0] # You can reverse lists (<i>or strings!</i>)
using "skip"-slicing with a =-1= asthe amount to skip.

In [10]: [6,7,8,9][1:]
Out[10]: [7,8,9] # You can slice lists using the raw list
instead of the name (Not that this would be very useful,
admittedly!)

In [11]: 100*L + [42]*100
Out[11]: (a list with 500 elements)

In [12]: L = 42 # You can reassign the name L to another value,
even of a different type. Now, L names the integer 42, instead
of the list it used to represent.
(no response from Python)

In [13]: L == 42 # Two equals are different than 1! This
<i>tests for equality</i>.
Out[13]: True

In [14]: L != 42 # This tests for "not equal."
Out[14]: False
```

Errors and *Exceptions*

Mistakes are unavoidable! So, you'll encounter Python errors. They're sometimes called *exceptions*, as well.

One of the most important habits we hope you'll practice in CS 5 is this:
if an error happens, consider it an opportunity, not a problem!

It's true, in that an error is a chance to

1. improve your intuition about how computation works, i.e., the "machine's mindset,"
2. improve the software you're developing (or your understanding of it), and
3. build on your debugging skills... .

So, let's create some errors, which Python calls *exceptions*:

Give yourself two minutes. In that time, see how many of these Python exceptions you can cause!

If you create others, all the better—let an instructor or tutor know, and we'll add them to this list:

- `NameError` (an unrecognized variable!)
- `TypeError` (try slicing an integer for example!)
- `ZeroDivisionError` (perhaps clear from its name)
- `SyntaxError` (the error that kittens most often produce when walking over the keyboard)
- `IndexError` (try an out-of-bounds index into a sequence)
- `OverflowError`
  Remember that integers won't overflow—if they get too big to fit in memory, they'll simply crash Python. To obtain this error, therefore, you'll need to use a floating-point value, such as `42.0`, in some mathematical expression that produces very large values! For example, use the power `**` operator!

Lists! Challenges with slicing and indexing

This problem will exercise your slicing-and-indexing skills.

First, create a new text file and save it with the name `hw1pr1.py` .

To do: Then, copy the following starting lines into your new plain-text file:

```
# CS5 Gold, hw1pr1
# Filename: hw1pr1.py
# Name:
# Problem description: First Python lab!

pi = [3,1,4,1,5,9]
e = [2,7,1]

# Example problem (problem 0):  [2,7,5,9]
answer0 = e[0:2] + pi[-2:]
print(answer0)
```

A couple of notes on this code:

- Be sure to save this as a plain-text file named `hw1pr1.py` *You'll need the `.py` extension.*
- After the initial comment, this code defines the list named `pi` and the list named `e`.
- When you run the file, the line `answer0 = e[0:2] + pi[-2:]` will define the value held by the variable `answer0`.
- Then, the code will print the value of the variable `answer0`.

To run the code:

- First, at your command-line, make sure you're in the folder/location where `hw1pr1.py` is located. Perhaps it's the desktop or another folder.
- Run `ipython`
- Then, within Python, enter `run hw1pr1.py`
- (Tab-completion and up-arrow can make things easier.)

Composing new lists from `pi`, `e`, and list operations

The problems below ask you to create several lists using **only** the list named `pi`, the list named `e`, and these list operations:

- list indexing, such as `pi[0]`
- list slicing, such as `e[1:]`
- skip-slicing, such as `pi[0:6:2]`

- list concatenation with `+`, such as `e[0:2] + pi[-2:]`
  (for this problem, we ask you *not* to use `+` to add values numerically)

- Please **leave a blank line** or two between your answers (to keep things readable - this makes the graders happy)!
- Once you've run the file once, you can experiment at Python's command-line - try it by typing `e[0:1] + pi[0:1]`
- ***For fun only***, you might try using as few operations as possible, to keep your answers elegant and efficient.

- Here are the problems:

- **0**. Use `pi` and/or `e` to create the list `[2,7,5,9]`. This is the example above, stored in the variable `answer0`.

- **1**. Use `pi` and/or `e` to create the list `[7,1]`.
  - As above, store this list in the variable `answer1`. Here is a start, to copy-and-paste:

```
o
o  # Problem 1: creating [7,1]
o  answer1 =   e[1:2]           # not the right answer, but a
   start...
o  print(answer1)
o
```

- **2**. Use `pi` and/or `e` to create the list `[9,1,1]`. Store this list in the variable `answer2`.

- **3**. Use `pi` and/or `e` to create the list `[1,4,1,5,9]`. Store this list in the variable `answer3`.

- **4**. Use `pi` and/or `e` to create the list `[1,2,3,4,5]`. Store this list in the variable `answer4`.

Strings! Slicing and indexing

This problem continues in the style of the last one, but uses strings rather than lists.
 First, copy these lines into your `hw1pr1.py` file underneath the previous problems (with some blank lines to keep things apart!):

```
# Lab1 string practice
```

```
h = 'harvey'
m = 'mudd'
c = 'college'
```

You may use any combination of these four string operations:

- String indexing, e.g., `h[0]`
- String slicing, e.g., `m[1:]`
- String concatenation, `+`, e.g., `h + m`
- Repetition, *, e.g., `42*c`

Again, the number of operations in the shortest answers that we know about are in parentheses. If you'd like, you might see if your answers are equally or more concise.
However, **any correct answer is OK** - there's no requirement to use a small number of operations.

Example problem (#5): Use `h`, `m`, and `c` to create `'hey'`. Store this string in the variable `answer5`. We used 3 operations.

Answer to example 5 - please copy and paste this into your file:

```
# Problem 5:  'hey'
answer5 = h[0] + h[4:6]
print(answer5)
```

The 3 operations are 1 use of list indexing, 1 slice, and 1 concatenation with `+`.

Here are the string-creation challenges (and, in parens, our most efficient answers, at least so far):

- *Remember that the "most efficient answers" are not at all needed* (they may be fun, but ***any*** working answer is 100% OK!)

- **5**. (The example from above) Create `hey` and store this string in the variable `answer5`. (3 ops.)

- **6**. Create `collude` and store this string in the variable `answer6`. (our best: 5 ops.)

- **7**. Create `arveyudd` and store this string in the variable `answer7`. (our best: 3 ops.)
- **8**. Create `hardeharharhar` and store this string in the variable `answer8`. (our best: 8 ops.)
- **9**. Create `legomyego` and store this string in the variable `answer9`. (our best: 8 ops.)
- **10**. Create `clearcall` and store this string in the variable `answer10`. (our best: 9 ops.)

If you have gotten to this point, you have completed the first half of Lab 1! You should submit your `hw1pr1.py` file at the Submission Site .

Excellent! Now, on to Lab 1, Problem 2: *functions*