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

Welcome to Lab 1! 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*)

There's no file nor anything to hand in for this part — just open Python's shell and try out the Python commands below...

The Python shell is the window in which programs execute. You will see a prompt with three greater-than signs at which you can enter short snippets of Python in order to try things out:

```
>>>
```

Apr 24-9:45 AM

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

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

```
>>> 40 + 2
42

>>> 40 ** 2
1600

>>> 40 % 2
0

>>> 'hi there!'
hi there!
```
(notice Python's politeness!)

```
>>> 'who are you?'
who are you?
```
(though sometimes it's a bit touchy.)

```
>>> list = [0, 1, 2, 3]
(no response from Python)
```
You can label data (here, a *list*) with a name (here, the name `list`)

```
>>> list
[0, 1, 2, 3]
```
You can see the data (here, a list) referred to by a name (here, `list`)

```
>>> list[1:]
[1, 2, 3]
```
You can slice lists (here, using the name `list`)

Apr 24-9:45 AM

```
>>> list[::-1]
[3, 2, 1, 0]
```
You can reverse lists (*or strings!*) using "skip"-slicing with a `-1` as the amount to skip.

```
>>> [0, 1, 2, 3][1:]
[1, 2, 3]
```
You can slice lists using the raw list instead of the name you gave the list
(Not that this would be very useful, admittedly!)

```
>>> 100*list + [42]*100
(a list with 500 elements that I'm too lazy to type here)
```

```
>>> list = 42
(no response from Python)
```
You can reassign the name `list` to another value, even of a different type—now, `list` names the integer 42, instead of the list it used to represent.

```
>>> list == 42
True
```
Two equals are different than 1! This *tests for equality*.

```
>>> list != 42
False
```
This tests for "not equal."

Apr 24-9:46 AM

Errors or Exceptions

If you didn't type things in perfectly, Python will reply with an error or an *exception*, as it is often called. See if you can make Python create the following exceptions, but don't spend more than a minute or so in total!

`SyntaxError`
`TypeError` (try slicing an integer for example!)
`ZeroDivisionError`
`IndexError` (try an out-of-bounds index)
`OverflowError`

Remember that integers will crash the machine before overflowing — so to obtain this error you'll need to use floating-point values in some large mathematical expression!

Apr 24-9:46 AM

Challenges with slicing and indexing... Lists!

This problem will exercise your slicing-and-indexing skills. First, use the `File->New Window` menu options to create an editor window. You will use this window to create your `hw1pr1.py` file. **To do:** Copy the following lines into the editor window:

```
#
# Name:
#
# hw1pr1.py (Lab 1, part 1)
# slicing and indexing challenges
#
# Author:       Enter author name (or names if pair programming)
# Lab Section: Enter your lab section
# Lab Time:     Enter the day/time of your lab
#

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

Apr 24-9:46 AM

Fill in the appropriate information at the top of your lab. Then save the contents under the name `hw1pr1.py`. Now, when you press **F5**, these two lists will be recognized by the Python shell. The challenge is to create several lists using **only** the list labeled `pi`, the list labeled `e`, and the four list operations here:

- list indexing such as `pi[0]`
- list slicing such as `e[1:]`
- list concatenation, +, such as `pi[:1] + e[1:]` (do *not* use + to add values numerically)
- the list-making operator, `[ , ]` for example: `[e[2], e[0]]`

For each one, place your answer into an appropriate variable in your `hw1pr1.py` file as you see in the example below. Include a comment on the line above and a print statement on the line below. That way, each time you use F5 to reload the file, the results will print — it's much easier to check that way!

Though not mandatory, you might try to use as few operations as possible, to keep your answers elegant and efficient. Here is an example to follow:

Example problem Use `pi` and/or `e` to create the list `[2, 5, 9]`. Store this list in the variable `answer0`.

Answer to the example problem

```
# Creating the list [2, 5, 9] from pi and e
answer0 = [e[0]] + pi[-2:]      # adds the lists:  [2] + [5, 9]
print(answer0)
```

Please leave a blank line or two between your answers (to keep the graders happy)!

Apr 24-9:46 AM

Remember that you can use the python interpreter to try things out!

Here are the problems:

- Use `pi` and/or `e` to create the list `[7, 1]`. Store this list in the variable `answer1`.
- Use `pi` and/or `e` to create the list `[9, 1, 1]`. Store this list in the variable `answer2`.
- Use `pi` and/or `e` to create the list `[1, 4, 1, 5, 9]`. Store this list in the variable `answer3`.
- Use `pi` and/or `e` to create the list `[1, 2, 3, 4, 5]`. Store this list in the variable `answer4`.

Apr 24-9:46 AM

Practicing with strings

This problem continues in the style of the last one, but uses strings rather than lists. So, these challenges ask you to create specified strings that result from using *only* the following three string literals, which you should copy into your `hw1pr1.py` file at this point:

```
# starting strings for Lab 1:
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` (using integers is OK here)

Apr 24-9:47 AM

Again, less is more: the number of operations in our shortest answers are in parentheses—you might find even more efficient ones...! However, **any answer is OK** — there's no requirement to use a small number of operations.

Example problem: Use `h`, `m`, and `c` to create `'heyyou'`. Store this string in the variable `answer42`. We used 9 operations.

Answer to example

```
# Creating the string 'heyyou'
answer42 = h[0] + h[4:] + h[-1] + c[1] + m[1]
print(answer42)
```

The 9 operations are 1 slice, 4 concatenations with + and 4 uses of list indexing. Here are the string-creation challenges (and our most efficient answers):

- Create `collude` Store this string in the variable `answer5`. (5 ops.)
- Create `arveyudd` Store this string in the variable `answer6`. (3 ops.)
- Create `hardeharharhar` Store this string in the variable `answer7`. (8 ops.)
- Create `legomyego` Store this string in the variable `answer8`. (9 ops.)
- Create `clearcall` Store this string in the variable `answer9`. (8 ops.)

Apr 24-9:47 AM

Apr 24-9:48 AM