Lab 5: Practice with Functions and Lists

This lab is intended to give you some practice using functions and lists. Unlike previous labs, no attempt is made to produce a useful program.

The Task

There is some code provided as lab5.py. Download this code to begin. The program is complete except for 4 functions:

- `second_highest`: Takes one parameter, a list, and returns the value of the second-highest number in the list. So `second_highest([1,5,3,9])` should return 5.
- `product`: Takes one parameter, a list, and returns the product of all the items multiplied together in the list. That is, for a list called L, it calculates:

$$\prod_{i=0}^{len(L)} L[i]$$

  So `product([2,4,1,8])` should return 24.
- `multiply_by_scaler`: Takes two parameters, a list and a number. Returns a list of the same length, but with each item multiplied by the given number.
  `multiply_by_scaler([1,2,3,4], 8)` should return `[8,16,24,32]`.
- `multiply_lists`: Takes two parameters, both lists. These must both be the same length. Returns a list of the same length, where each item is the corresponding items from the two parameter lists multiplied together. So `multiply_lists([1,1,2,2,], [1,2,3,4])` should return `[1,2,6,8]`.

About lab5.py

The code in the provided file (lab5.py) will use each of these four functions. Don't modify this code. Write your functions above the line:

```
if __name__ == "__main__":
```

That line, which checks to see if `__name__` happens to be `__main__`, is used to check if lab5.py is being used as a program, or as a library. If you run lab5.py in Python, the code under that if statement will be executed. But, if you write a different program, and include a line such as:

```
from lab5 import *
```

The functions will become available in your program, but `__name__` will not be set to `__main__`, and the code under that "if" statement will not be executed. This is one way to include a test routine if you are writing a library. It also provides a convenient way to re-use functions from a program you wrote previously.

Some Helpful Observations

In Python, you can find the largest item of a list with the `max` function. When designing `second_highest`, you can use `max` to find the highest, and then examine each item in the list to see if it is the highest so far but NOT the maximum. Essentially, an algorithm like this:

```
second_highest = 0
for each item in the list:
    if the item is higher than second_highest and not the maximum:
        second_highest = item
```

`product`, `multiply_by_scaler`, and `multiply_lists` can all be handled using an accumulator. For the latter two, an empty list will work (add each item as it is calculated). For `product`, a variable set to 1 as the initial value will work (multiply it by each item in the list in turn). For `product`, it will make the task more difficult if you start the accumulator off at 0, because 0 times anything is 0 - even if it is multiplied by every item in the list!

Expected Output

```
32
10
1244160
41472
[7, 70, 14, 378, 224, 84, 21]
[0.5, 5.0, 1.0, 27.0, 16.0, 6.0, 1.5]
[0, 10, 4, 162, 128, 60, 18]
[1, 100, 4, 2916, 1024, 144, 9]
[32, 320, 64, 1728, 1024, 384, 96]
[5, 500, 20, 14580, 5120, 720, 45]
```