Simple Looping

Many of the functions we have looked at can also be written using an *iterative* approach instead of recursion or list comprehensions. This can be particularly useful in situations where we are concerned with an item position within a list.

Multiplying with Iteration

Recall the simplified multiplication function `mult()` that we wrote in Lab 3. It could have been written iteratively as show below, (and provided as `lab6.py`).

```
def mult(n,m):
    ''' returns the product of n and m
        note that the multiplier n must be
          a non-negative inter value  '''
    product = 0
    for i in range(n):
        product = product + m
    return product
```

Note that due to the way the `range()` function works the *multiplier* `n` is still limited to non-negative, integer values when using this approach.

Now consider the dot product function, also from Lab 3. To restate the definition, the dot product of two lists is the sum of the products of the elements at the same index in each of the two lists. For example, if we have L = [5,3] and K = [6,4] then the dot product of L and K is 42, which comes from 5∗6 + 3∗4. As before, if the two lists are not the same length, or when both lists are empty, then 0.0 should be returned.

Write a function, `dot(L,K)`, using an iterative approach, (i.e. a `for`-based loop), to compute the dot product of two lists. Provide several printed tests cases in your program showing that your function works correctly.

Representing Polynomials

Consider a polynomial expression of the form

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

If we represent such an expression with a list of the coefficients, $a_0 \cdots a_n$, we can then find the value of the expression for a given *indeterminate*, x, by multiplying each coefficient by the appropriate power of x.

For example, $x^2 - 4x + 7$ could be represented by the list `[7, -4, 1]` and could be evaluated at $x = 3$ by computing $7 + (-4) * 3 + 1 * 3^2$, for an answer of 4.

Write a function called `poly_eval(coeff, x)` which takes as input a list of coefficients, `coeff`, and a value, `x`, and returns the result of evaluating the represented polynomial at the given value of x.

As usual, provide printed tests demonstrating that your function is correct.

Submitting your work

Once you have completed the two functions, and provided suitable printed tests, you are ready to submit your work.

Check that your file is named correctly, `lab6.py`, and that both your and your partner's names are included in the header comment.

Upload your file to the Lab 6 page on Moodle. Make sure that both partners will have access to the work you did today for when you each want to review the lab material.

Have one of the TAs in the lab come over and briefly evaluate your work.