Problem Wk.4.4.2: Implement LTISM

Now we will implement LTISM.

Recall that difference equations have the form:

$$y[n] = c_0 y[n-1] + c_1 y[n-2] + \ldots + c_{k-1} y[n-k] + d_0 x[n] + d_1 x[n-1] + \ldots + d_j x[n-j]$$

Part 1: Simulate LTISM

Consider an LTISM defined as follows

m = LTISM([1, 2], [1], [3], [4])

Find the values returned by this machine for this input sequence:

o = m.transduce([1, 2, 3, 4, 5])

1. o[0] =
2. o[1] =
3. o[2] =
4. o[3] =
5. o[4] =

Part 2: Code

Below is a skeleton of the `LTISM` class. Complete the definition of the `getNextValues` method. You may find this utility procedure helpful. It returns the dot product of two lists of numbers (which should be of the same length).

```
def dotProd(a, b):
    if len(a) == 0 or len(b) == 0: return 0
    if len(a) != len(b):
            print 'dotProd mismatch error ' + str(len(a)) + '  !=  ' + str(len(b))
    return sum([ai*bi for (ai,bi) in zip(a,b)])
```

It is already defined for your use.

Think through the cases when `cCoeffs` or `dCoeffs` are empty.

You should write your code in Idle. In the design lab folder, create a file with

```
import lib601.sm as sm
```

at the top. You can then place your definitions and test cases in the file. Evaluate them with Run Module. Use the example in the first part of this problem to do your testing.

Recall that the `transduce` method of the `sm.SM` class takes a list of input values, $x[0], \ldots, x[n]$, and generates a list of output values, $y[0], \ldots, y[n]$.

```
class LTISM (sm.SM):
    def __init__(self, dCoeffs, cCoeffs,
                 previousInputs = [], previousOutputs = []):
        self.cCoeffs = cCoeffs
        self.dCoeffs = dCoeffs
        # State is last j input values and last k output values
        self.startState = (previousInputs, previousOutputs)

    def getNextValues(self, state, input):
        (inputs, outputs) = state
        # Your code here
```