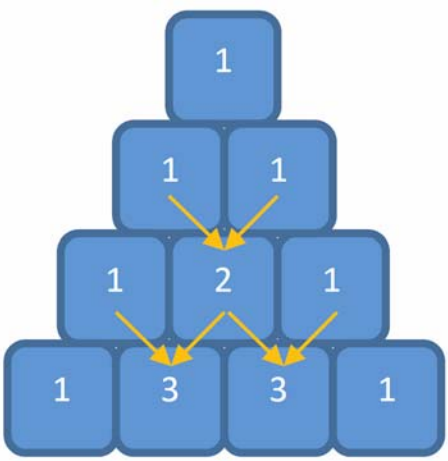
Pascal's Triangle

Pascal's Triangle was developed by a French mathematician named Blaise Pascal. The rows are numbered starting with 0 and go all the way to n, just like list indices in Python. The triangle always starts with a single term in the 0^{th} row, which is a [1]. The nth row of the triangle has n+1 terms, and each term is found by adding the values of the two terms above it. For example, row number 2 (actually the third row) has three terms: [1, 2, 1]. The row above it, row number 1, is [1, 1], and the row above that one, row 0, is just [1]. Pascal's Triangle has several uses in mathematics, such as binomial expansions. (http://en.wikipedia.org/wiki/Pascal%27s_triangle contains more information.) See how the terms are obtained in the figure below.



Task 1:

Your first task is to create a function, `pascal_row`, that outputs a list of elements found in a certain row of Pascal's Triangle. The `pascal_row` function should take a single integer as input, which represents the row number. Row number 0 indicates the single term row, [1]. The input to the `pascal_row` function will always be an integer greater than or equal to 0. Hopefully you have noticed that each row in Pascal's Triangle can be computed by recursively calculating the row above it. Be sure to think carefully about the base case in this problem. Also, it may be helpful to write a helper function that creates a new list of sums of adjacent terms in the original list.

Here are some examples of calling the `pascal_row` function:

```
>>> pascal_row(0)
[1]
>>> pascal_row(1)
[1, 1]
>>> pascal_row(5)
[1, 5, 10, 10, 5, 1]
```

Task 2:

Write a second function called `pascal_triangle` that takes as input a single integer n and returns a list of lists containing the values of the all the rows up to and including row n.

Here are some examples of calling the `pascal_triangle` function:

```
>>> pascal_triangle(0)
[[1]]
>>> pascal_triangle(1)
[[1], [1, 1]]
>>> pascal_triangle(5)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1]]
```

Task 3:

Write your own PyUnit test script. Be sure to test your `pascal_row` and `pascal_triangle` functions thoroughly. Write 5 tests for each function. A template is given below.

```
import unittest
import hw4

class Test(unittest.TestCase):

    def test01(self):
        # Replace pass with real test code. Use self.assertEqual.
        pass

    # Write more tests...

    def test10(self):
        pass

if __name__ == "__main__":
    unittest.main()
```

Submission:

Create a zip file called hw4.zip. It should contain only hw4.py and test_hw4.py. Upload that zip file to Canvas. For each file that is named incorrectly, you will earn 0 points on that part of the assignment. Also make sure that your test_hw4.py works on ANY correct implementation of the homework, not just your own.

[3:56]

also if possible I'm going to implement the bytecode cheating detection this week

[3:57]

but that's secondary to making sure the engine handles pyunit/zips as well

[3:59]

also we should note that their unit test's class Test must be named Test and not any other name