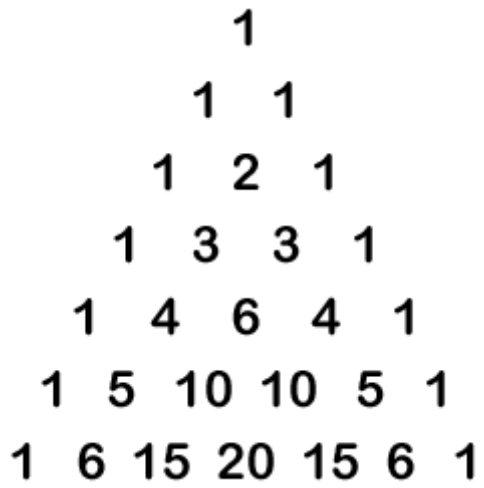
# Lab Assignment 6
# Pascal's Triangle

**Assignment Overview**

The goal of this lab is to practice with Lists. In Python, the List type is a container that holds a number of other objects, in a given order. This data structure is so useful and powerful that you can find it in almost every algorithm.

In this lab, we want to generate a Pascal's triangle. You can get the detailed explanation from Wikipedia ( http://en.wikipedia.org/wiki/Pascal's_triangle ), and the figure below is a simple Pascal's triangle whose height is 7.




You are going to write a program that asks for the height of Pascal's triangle, then generate the triangle and output it in the same style as above example.

Pascal's triangle rules:

1. To generate the triangle, you start from the first row, which is always 1, and the second row which is always 1 1.
2. After the first two rows, each row at level *h* is generated from the values at row *h*-1. Note that the leftmost number and the rightmost number in any row are always 1. Note that, in row *h*, there are *h* numbers.

**Part 1**

**(A)** Write a function named `make_new_row` that takes one argument `old_row` and generates the next row of Pascal's triangle (starting with `old_row = [1,1]`). In the lab directory is a file named `lab0.py` that contains a function header and requirements. For example:

```
>>> make_new_row([1,1])
[1, 2, 1]
>>> make_new_row([1,2,1])
```

```
[1, 3, 3, 1]
>>> make_new_row([1,3,3,1])
[1, 4, 6, 4, 1]
>>> make_new_row([1,4,6,4,1])
[1, 5, 10, 10, 5, 1]
```

Hints:

1. Append new values to a `new_row` list. After the initial 1, the values will be the sum of two elements of `old_row`.
2. It is easiest to use a `for` loop that loops through indices, e.g. `for i in range(n):` so you can refer to adjacent items using indices `i` and `i + 1`, but be careful that you do not get too large an index, i.e. out of range.
3. Don't forget the last 1.

**(B)** Refactor your function to handle the following special cases:

1. If `old_row` is `[]`, then return `[1]`.
2. If `old_row` is `[1]`, then return `[1,1]`.

Adjust the contract comments to reflect handling of special cases.

**Part 2**

Write a program that prompts for a height greater than 0 and prints lists of Pascal's triangle rows to that height (no error checking needed). For example:

```
>>>
Enter the desired height of Pascal's triangle: 7
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
```

Hints:

1. Use the function from part 1 in a loop

**Part 3**

Rewrite your program so that you collect all the rows into a master list, that is, a list of lists. You can do that by starting with a list and then appending lists onto the list. Print your list of lists two ways:

1. If your master list is named L, simply `print(L)`
2. Then print each list on its own line

The output is shown below:

```
Enter the desired height of Pascal's triangle: 7
```

```
Printing whole list of lists:
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10,
10, 5, 1], [1, 6, 15, 20, 15, 6, 1]]
Printing list of lists, one list at a time:
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
```