## 17 Nov 2018, 11 Nov 2020, 17 Apr 2021, 13 Nov 2021

Floyd's triangle is a right-angled triangular array of natural numbers, used in computer science education. It is named after Robert Floyd. It is defined by filling the rows of the triangle with consecutive numbers, starting with a 1 in the top left corner.

Implement the function floyd_row(n) which takes in a positive integer n and returns a tuple of integers that represent the nth row of the Floyd's triangle.

In [1]:
```
def floyd_row(n):

    if n == 1:
        return [1]
    else:
        newRow = []
        lastRow = floyd_row(n-1)
        for i in range(len(lastRow)):
            newRow.append(lastRow[i] + n-1)
            if i == len(lastRow)-1:
                newRow.append(newRow[i]+1)

        return tuple(newRow)
```

In [2]:
```
floyd_row(2)
```

Out[2]:
```
(2, 3)
```

In [3]:
```
floyd_row(5)
```

Out[3]:
```
(11, 12, 13, 14, 15)
```

Implement the function floyd_sum(n) which takes in a positive integer n and returns the sum of all the numbers in the Floyd's triangle until and including row n. You should solve it computationally taking at least O(n) time, i.e., not simply using a formula.

In [4]:
```
def sumRows(array):

    count = 0

    for num in array:

        count += num

    return count
```

In [5]:
```
def floyd_sum(n):

    count = 0
    final = 0

    for i in range(n+1):

        count = count + i
```

```
    for j in range(count+1):

        final = final + j

    return final
```

In [6]:
```
floyd_sum(1)
```

Out[6]:
```
1
```

In [7]:
```
floyd_sum(2)
```

Out[7]:
```
6
```

In [8]:
```
floyd_sum(5)
```

Out[8]:
```
120
```

You have been asked to babysit your 5-year-old nephew for the weekend and he likes to play with blocks. The blocks come in two forms: (i) cubes; and (ii) 2-cube cubiods (equivalent to 2 cubes glued together). He wants to build pyramids, which has n layers. The bottom layer has length n and each subsequent layer above is 1 cube shorter. At the very top is a cube.

In [9]:
```
def bottomRow(n):
    if n == 1:
        return 1
    elif n == 2:
        return 2
    else:
        return (bottomRow(n-1) + bottomRow(n-2))
```

In [10]:
```
def pyramids(n):

    if n == 1:
        return 1
    elif n == 2:
        return 2
    else:
        return bottomRow(n)*pyramids(n-1)
```

In [11]:
```
pyramids(3)
```

Out[11]:
```
6
```

In [12]:
```
pyramids(4)
```

Out[12]:
```
30
```

In [13]:
```
def new1(curr):

    if curr%2 == 0:

        return (curr/2) + 1 + curr

    else:

        return (curr-1)/2 + 1 + curr
```

In [14]:
```
def new2(curr, dest):

    x = dest - curr

    if x%2 == 0:

        return curr + x/2 + 1

    else:

        return curr + (x-1)/2 + 1
```

In [15]:
```
def nextjump(curr, dest):

    if abs(curr - dest) == 0:

        return 0

    elif new2(curr, dest) == dest:

        return nextjump(new2(curr,dest), dest) + 1

    elif new1(curr) == dest:

        return nextjump(new1(curr), dest) + 1

    elif new1(curr) < dest and new2(curr,dest) > dest:

        return nextjump(new1(curr), dest) + 1

    elif new1(curr) > dest and new2(curr,dest) < dest:

        return nextjump(new2(curr,dest), dest) + 1

    else:

        min1 = nextjump(new1(curr), dest) + 1
        min2 = nextjump(new2(curr,dest), dest) + 1
        return min(min1,min2)

#must put in more conditional checks. Cannot allow tree to expand indefinately
```

In [16]:
```
def jump(n):

    return nextjump(0, n)
```

In [17]:
```
for i in range(1,21):

    print(i,jump(i))
```

```
1 1
2 1
3 2
4 2
5 2
6 2
7 2
8 2
9 3
10 2
11 3
12 3
13 3
14 3
15 3
16 3
17 4
18 3
19 4
20 4
```

Implement the function num_walls that takes as input size and returns the number of walls in the honeycomb of the input size

In [18]:
```
def num_walls(n):

    if n == 0:

        return 0

    elif n == 1:

        return 6

    else:

        return num_walls(n-1) + 4 + 4 + 4 + 6*(n-2)+1
```

In [19]:
```
for i in range(21):
    print(num_walls(i))
```

```
0
6
19
38
63
94
131
174
223
278
339
406
479
558
643
734
831
934
1043
1158
1279
```

In [20]:
```
num_walls(50)
```

Out[20]:
```
7699
```