# Assessment:

## Objective:

This assessment is a programming assignment to practice different functionalities of Python/Numpy. The main goal of this assessment is to get familiar with python coding.

## Desctiption:

This programming assessment is composed of 2 part. The first part would be a problem statement that you need to solve it using different concepts we've learnt during class related to Python. The second part would be a problem statement that you need to solve it using Numpy.

# Part A: Clouds

## problem statement:

A child is playing a cloud hopping game. In this game, there are sequentially numbered clouds that can be thunderheads or cumulus clouds. The character must jump from cloud to cloud until it reaches the start again.

There is an array of clouds, and an energy level . The character starts from and uses unit of energy to make a jump of size to cloud . If it lands on a thundercloud, , its energy () decreases by additional units. The game ends when the character lands back on cloud .

Given the values of , , and the configuration of the clouds as an array , determine the final value of after the game ends.

**Example:** C = [0,0,1,0]

K = 2

The indices of the path are . The energy level reduces by for each jump to . The character landed on one thunderhead at an additional cost of energy units. The final energy level is .

Note: Recall that refers to the modulo operation. In this case, it serves to make the route circular. If the character is at and jumps , it will arrive at .

## Function Description:

Complete the jumpingOnClouds function in the editor below.

jumpingOnClouds has the following parameter(s):

- int c[n]: the cloud types along the path
- int k: the length of one jump

## Return

- int: the energy level remaining

## Input Fromat:

The function should takes 2 argument, ***c*** and ***k***.

- ***k***: is the jump distance
- ***c***: is an array contains the cloud types.
  - if c[i] = 0, the cloud i is a cumuulus cloud.
  - if c[i] = 1, the cloud i is thunderhead.

## Constraints

- c[i] in {0,1}

## Sample Input:

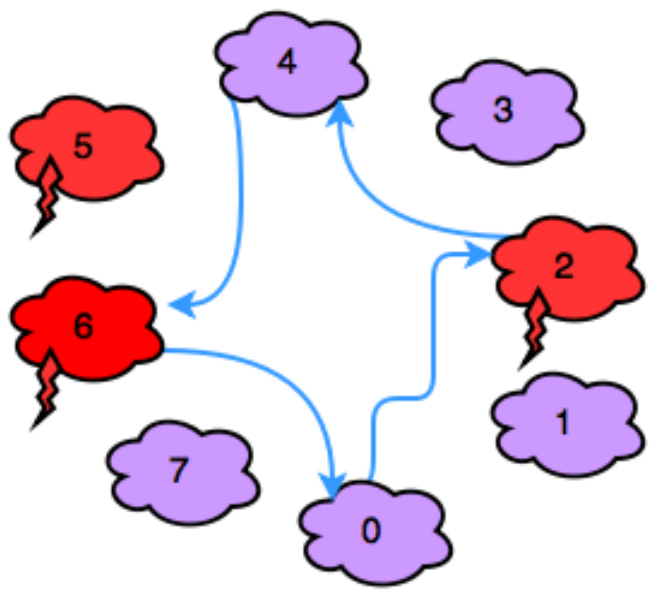
k = 2 c = [0,0,1,0,0,1,1,0]

## Sample Output

**92**

## Explanation

In the diagram below, red clouds are thunderheads and purple clouds are cumulus clouds:




Observe that our thunderheads are the clouds numbered , , and . The character makes the following sequence of moves:

1. Move:0 -> 2, Energy: e = 100 - 1 - 2 = 97
2. Move:2 -> 4, Energy: e = 97 - 1 = 96
3. Move:4 -> 6, Energy: e = 96 - 1 - 2 = 93
4. Move:6 -> 0, Energy: e = 93 - 1 = 92

# Part 2: Numpy Discovery

Create a function named `calculate()` that uses Numpy to output the mean, variance, standard deviation, max, min, and sum of the rows, columns, and elements in a 3 x 3 matrix.

The input of the function should be a list containing 9 digits. The function should convert the list into a 3 x 3 Numpy array, and then return a dictionary containing the mean, variance, standard deviation, max, min, and sum along both axes and for the flattened matrix.

The returned dictionary should follow this format:

```
{
  'mean': [axis1, axis2, flattened],
  'variance': [axis1, axis2, flattened],
  'standard deviation': [axis1, axis2, flattened],
  'max': [axis1, axis2, flattened],
  'min': [axis1, axis2, flattened],
  'sum': [axis1, axis2, flattened]
}
```

If a list containing less than 9 elements is passed into the function, it should raise a `ValueError` exception with the message: "List must contain nine numbers." The values in the returned dictionary should be lists and not Numpy arrays.

For example, `calculate([0,1,2,3,4,5,6,7,8])` should return:

```
{
  'mean': [[3.0, 4.0, 5.0], [1.0, 4.0, 7.0], 4.0],
  'variance': [[6.0, 6.0, 6.0], [0.6666666666666666, 0.6666666666666666, 0.6666666666666666],
6.666666666666667],
  'standard deviation': [[2.449489742783178, 2.449489742783178, 2.449489742783178], [0.816496580927726,
0.816496580927726, 0.816496580927726], 2.581988897471611],
  'max': [[6, 7, 8], [2, 5, 8], 8],
  'min': [[0, 1, 2], [0, 3, 6], 0],
  'sum': [[9, 12, 15], [3, 12, 21], 36]
}
```