ASSIGNMENT 6.1

Student: K. Anandaranga

---

1. Introduction

This assignment will help you to consolidate the concepts learnt in the session.

2. Problem Statement

- Write a function so that the columns of the output matrix are powers of the input Vector. The order of the powers is determined by the increasing boolean argument. Specifically, when increasing is False, the i-th output column is the input vector raised element-wise to the power of N - i - 1.

HINT: Such a matrix with a geometric progression in each row is named for Alexandre-Theophile Vandermonde.

3. Output

ACD MDS Assignment 6.1

Student: K. Anandaranga - Mar 2018 batch

In [74]:
```
def vanderMatrixGen(a, N, b_option):
    ''' Function replicates the built-in function called 'vander'; Gives the output as a matrix with a geometric progression in each row
        a = array with integer values; N = number of columns, where each column value is from the geometric progression
        b_option is True if ascending and False if descending order
    '''
    import numpy as np
    o = []              # Empty list to store the geometric progression values
    x1 = len(a)         # Number of rows in array is based on the number of vaues

    for i in range(len(a)):        # Create the geometric progression and append to the empty list
        for j in range (0,N):
            b = a[i]**j
            o.append(b)
    o_rev = o[: : -1]              # Create the reverse array if b_option is False

    if b_option == True:
        np_arr = np.array(o)       # Convert the geometric progression list into an array
        c = np_arr.reshape(x1, N)  # reshape the array per 'a' and 'N' values
        print (c)
    else:
        np_arr = np.array(o_rev)   # Same as above, expect the reverse list is used
        c = np_arr.reshape(x1, N)  # reshape the array
        c_flip = np.flipud(c)      # Use the flipud function to reverse the row order
        print (c_flip)

a = [2,3,5,6]                                  # Use the same inputs for both new and built-in function to test results
N = 4
b_option = False

print ('*'*10, "Results using NEW function", '*'*10)
vanderMatrixGen(a, N, b_option)
print ('*'*10, "Results using BUILT-IN 'vander' function", '*'*10)
np.vander(a, N, b_option)
```

```
********** Results using NEW function **********
[[  8   4   2   1]
 [ 27   9   3   1]
 [125  25   5   1]
 [216  36   6   1]]
********** Results using BUILT-IN 'vander' function **********
```

Out[74]:
```
array([[  8,   4,   2,   1],
       [ 27,   9,   3,   1],
       [125,  25,   5,   1],
       [216,  36,   6,   1]])
```