In [ ]:
```
# Name: Sujey Rajesh Naik
# Reg No: 24-27-27
# Course: Data Science Tools & Techniques (AM609)
# Programme: MTech Data Science
# Assignment Number: 02
```

In [3]:
```
import numpy as np
```

In [26]:
```
#Question 1a
#Array is created with values 0 to 30 both inclusive
var1 = np.arange(0,31)
print(var1)
print(var1.shape)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
(31,)
```

In [28]:
```
#Question 1b
#As size of var1 is 31, we are skipping element 0 so that we can create 2d matrix
var2 = var1[1:].reshape(6,5)
print(var2)
print(var2.shape)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]
 [26 27 28 29 30]]
(6, 5)
```

In [30]:
```
#Question 1c
var3 = var2.reshape(5,2,3)
print(var3)
print(var3.shape)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]]

 [[25 26 27]
  [28 29 30]]]
(5, 2, 3)
```

In [44]:
```
#Question 1d
var2[1,0] = -1
```

In [46]:
```
print(var1)
print(var2)
print(var3)
```

```
[ 0  1  2  3  4  5 -1  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
[[ 1  2  3  4  5]
 [-1  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]
 [26 27 28 29 30]]
[[[ 1  2  3]
  [ 4  5 -1]]

 [[ 7  8  9]
  [10 11 12]]

 [[13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]]

 [[25 26 27]
  [28 29 30]]]
```

In [35]:
```
# when we change a value in var2, the corresponding value in var1 and var3 are also changed.
# This is because the reshape() function in NumPy returns a view of the original array whenever possible, which
# Since var2 and var3 are views of the original array var1, they all point to the same data in memory.
# Therfore,when we change the value in var2 using array indexing, i.e, setting the first value of the second ro
```

In [48]:
```
#Question 1e
#i)
result1 = var3.sum(axis=1) #Sum var3 over its second dimension
print(result1)
```

```
[[ 5  7  2]
 [17 19 21]
 [29 31 33]
 [41 43 45]
 [53 55 57]]
```

In [50]:
```
#ii)
result2 = var3.sum(axis=2) #Sum var3 over its third dimension
print(result2)
```

```
[[ 6  8]
 [24 33]
 [42 51]
 [60 69]
 [78 87]]
```

In [52]:
```
#iii)
result3 = var3.sum(axis=(0,2)) #Sum var3 over both its first and third dimensions
print(result3)
```

```
[210 248]
```

In [54]:
```
#Question 1f
#i)
print(var2[1,:]) #Second row of var2 with shape= (6,5)
```

```
[-1  7  8  9 10]
```

In [56]:
```
#ii)
print(var2[:,-1]) #last column of var2
```

```
[ 5 10 15 20 25 30]
```

In [58]:
```
#iii)
print(var2[:2,-2:]) #top right 2*2 submatrix of var2
```

```
[[ 4  5]
 [ 9 10]]
```

In [60]:
```
#Question 2a
array1 = np.arange(10)+1
print(array1)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

In [62]:
```
#Question 2b
array2 = np.arange(10)
A = array2.reshape(10,1) + array2
print(A)
```

```
[[ 0  1  2  3  4  5  6  7  8  9]
 [ 1  2  3  4  5  6  7  8  9 10]
 [ 2  3  4  5  6  7  8  9 10 11]
 [ 3  4  5  6  7  8  9 10 11 12]
 [ 4  5  6  7  8  9 10 11 12 13]
 [ 5  6  7  8  9 10 11 12 13 14]
 [ 6  7  8  9 10 11 12 13 14 15]
 [ 7  8  9 10 11 12 13 14 15 16]
 [ 8  9 10 11 12 13 14 15 16 17]
 [ 9 10 11 12 13 14 15 16 17 18]]
```

In [64]:
```
#Question 2c
import numpy.random as npr
data = np.exp(npr.randn ( 50 , 5 ) )
```

In [66]:
```
#Question 2d & 2e
mean = data.mean(axis=0)
print("Mean:",mean)
```

```
Mean: [1.99487516 1.60206809 1.6673333  1.67504494 1.47492236]
```

In [68]:
```
std = data.std(axis=0)
print("std:",std)
```

```
std: [2.55556899 1.68299729 1.71657705 2.46181126 1.37037642]
```

In [70]:
```
#Question 2f
```

```
normalized = data - mean
normalized = normalized / std
```

In [72]:
```
normalized.mean(axis=0).round()
```

Out[72]:
```
array([ 0., -0.,  0.,  0.,  0.])
```

In [74]:
```
normalized.std(axis=0).round()
```

Out[74]:
```
array([1., 1., 1., 1., 1.])
```

In [76]:
```
#From the above results, we can tell that we have standardized the data as mean is equal to zero
#And standard deviation is equal to 1
```

In [78]:
```
#Question 3a
#Function for creating vandermonde matrix
def vandermonde (N):
    vec = np.arange (N) +1
    vander = vec.reshape(N,1) ** (vec-1)
    return vander
```

In [80]:
```
vander = vandermonde(12)
print(vander)
```

```
[[          1           1           1           1           1           1
            1           1           1           1           1           1]
 [          1           2           4           8          16          32
           64         128         256         512        1024        2048]
 [          1           3           9          27          81         243
          729        2187        6561       19683       59049      177147]
 [          1           4          16          64         256        1024
         4096       16384       65536      262144     1048576     4194304]
 [          1           5          25         125         625        3125
        15625       78125      390625     1953125     9765625    48828125]
 [          1           6          36         216        1296        7776
        46656      279936     1679616    10077696    60466176   362797056]
 [          1           7          49         343        2401       16807
       117649      823543     5764801    40353607   282475249  1977326743]
 [          1           8          64         512        4096       32768
       262144     2097152    16777216   134217728  1073741824           0]
 [          1           9          81         729        6561       59049
       531441     4782969    43046721   387420489  -808182895  1316288537]
 [          1          10         100        1000       10000      100000
      1000000    10000000   100000000  1000000000  1410065408  1215752192]
 [          1          11         121        1331       14641      161051
      1771561    19487171   214358881 -1937019605   167620825  1843829075]
 [          1          12         144        1728       20736      248832
      2985984    35831808   429981696   864813056  1787822080   -20971520]]
```

In [82]:
```
#Question 3b
x = np.ones(12)
print(x)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [84]:
```
b = np.dot(vander,x)
print(b)
```

```
[1.20000000e+01 4.09500000e+03 2.65720000e+05 5.59240500e+06
 6.10351560e+07 4.35356467e+08 2.30688120e+09 1.22713351e+09
 9.43953692e+08 3.73692871e+09 3.10225064e+08 3.10073456e+09]
```

In [92]:
```
#Question 3c
import numpy.linalg as nplg
inverted_vander = nplg.inv(vander)
result = np.dot(inverted_vander,b)
print(result)
```

```
[1.00000572 0.99735641 1.00311279 0.999506   1.00002861 0.99999857
 1.0000001  1.         1.         1.         1.         1.        ]
```

In [100…
```
#In the above result, we are solving linear equation as we are solving Ax=B. So if we do inverse(A)*B we should
#Even though the above result is approximately equal to 1, there is slight difference only because of floating
```

In [94]:
```
#Question 3d
result_solved = nplg.solve(vander,b)
```

In [96]:
```
print(result_solved)
```

```
[1.00000067 0.99999715 1.00000411 0.99999733 1.00000089 0.99999984
 1.00000002 1.         1.         1.         1.         1.        ]
```

In [102…
```
#Here we are doing the same thing as above and the result in 3c and 3d should match as we solving for x with sa
```

```
#We can see in the above result the values are approximately equal to 1. There is slight difference only becaus
```

In [104…

```
#https://github.com/Sujey-Nk/Sujey_Rajesh_Naik_24-27-27
```