```python
import numpy as np

# Q1 a) Create a variable var1 that stores an array of numbers from 0
to 29 and print it
var1 = np.arange(0, 30)  # Changed the range to 0 to 29 (30 elements)
print("var1:", var1)

# Q1 b) Change var1 to a 2D array with 5 rows using reshape function
var2 = var1.reshape(5, 6)
print("\nvar2 (reshaped):\n", var2)

# Q1 c) Reshape var2 into a 3D array
var3 = var2.reshape(2, 3, 5)
print("\nvar3 (reshaped to 3D):\n", var3)

# Q1 d) Use two-dimensional array indexing
var2[1, 0] = -1
print("\nModified var2:\n", var2)
print("\nModified var3 (after changing var2):\n", var3)

# Q1 e) Using NumPy sum functions over specified dimensions
print("\nSum var3 over its second dimension:\n", np.sum(var3, axis=1))
print("\nSum var3 over its third dimension:\n", np.sum(var3, axis=2))
print("\nSum var3 over both first and third dimensions:\n",
np.sum(var3, axis=(0, 2)))

# Q1 f) Slicing var2
print("\nSecond row of var2:\n", var2[1, :])
print("\nLast column of var2:\n", var2[:, -1])
print("\nTop right 2x2 submatrix of var2:\n", var2[:2, -2:])
```

```
var1: [ 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]

var2 (reshaped):
 [[ 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]]

var3 (reshaped to 3D):
 [[[ 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]]]
```

```
Modified var2:
 [[ 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]]

Modified var3 (after changing var2):
 [[[ 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]]]

Sum var3 over its second dimension:
 [[15 11 21 24 27]
 [60 63 66 69 72]]

Sum var3 over its third dimension:
 [[ 10  28  60]
 [ 85 110 135]]

Sum var3 over both first and third dimensions:
 [ 95 138 195]

Second row of var2:
 [-1  7  8  9 10 11]

Last column of var2:
 [ 5 11 17 23 29]

Top right 2x2 submatrix of var2:
 [[ 4  5]
 [10 11]]
```

```
import numpy as np
import numpy.linalg as la

def vandermonde(N):
    vec = np.arange(N) + 1
    vander = vec[:, np.newaxis] ** vec[np.newaxis, :]
    return vander

N = 12
vander = vandermonde(N)
print("Vandermonde matrix:")
print(vander)
```

```
x = np.ones(N)
b = vander @ x
print("Vector b:")
print(b)

# Naive solution using matrix inversion
x_naive = la.inv(vander) @ b
print("Naive solution:")
print(x_naive)

# Solution using np.linalg.solve
x_solve = la.solve(vander, b)
print("Solution using np.linalg.solve:")
print(x_solve)

Vandermonde matrix:
[[          1           1           1           1           1
1
            1           1           1           1           1
1]
 [          2           4           8          16          32
64
          128         256         512        1024        2048
4096]
 [          3           9          27          81         243
729
         2187        6561       19683       59049      177147
531441]
 [          4          16          64         256        1024
4096
        16384       65536      262144     1048576     4194304
16777216]
 [          5          25         125         625        3125
15625
        78125      390625     1953125     9765625    48828125
244140625]
 [          6          36         216        1296        7776
46656
       279936     1679616    10077696    60466176   362797056 -
2118184960]
 [          7          49         343        2401       16807
117649
       823543     5764801    40353607   282475249  1977326743
956385313]
 [          8          64         512        4096       32768
262144
      2097152    16777216   134217728  1073741824           0
0]
 [          9          81         729        6561       59049
531441
```

```
      4782969    43046721   387420489  -808182895  1316288537 -
1038305055]
 [         10         100        1000       10000      100000
1000000
     10000000   100000000  1000000000  1410065408  1215752192  -
727379968]
 [         11         121        1331       14641      161051
1771561
     19487171   214358881 -1937019605   167620825  1843829075 -
1192716655]
 [         12         144        1728       20736      248832
2985984
     35831808   429981696   864813056  1787822080   -20971520  -
251658240]]
Vector b:
[ 1.20000000e+01  8.19000000e+03  7.97160000e+05  2.23696200e+07
  3.05175780e+08 -1.68282849e+09  3.26326651e+09  1.22713351e+09
 -9.43513640e+07  3.00954874e+09 -8.82491592e+08  2.84907632e+09]
Naive solution:
[1.00036895 0.99989367 1.00001907 0.99999964 0.99999973 1.00000004
 1.         1.         1.         1.         1.         1.        ]
Solution using np.linalg.solve:
[1.00000086 0.99999879 1.00000058 0.99999989 1.         1.
 1.         1.         1.         1.         1.         1.        ]

import numpy as np
import numpy.random as npr

# Part (a) - Create a vector 1, 2, ..., 10 by adding 1 to the result
of arange function
vector = np.arange(10) + 1
print("Vector 1 to 10:", vector)

# Part (b) - Create a 10 x 10 matrix A where A[i, j] = i + j
A = vector.reshape(10, 1) + vector
print("\n10x10 Matrix A (A[i,j] = i + j):")
print(A)

# Part (c) - Create a fake dataset of 50 examples, each with 5
dimensions
data = np.exp(npr.randn(50, 5))
print("\nFake dataset (50x5) created:")

# Part (e) - Compute mean and standard deviation of each column
mean = np.mean(data, axis=0)
std = np.std(data, axis=0)
print("\nMean of each column:", mean)
print("Standard deviation of each column:", std)

# Part (f) - Standardize the data matrix
```

```
normalized = (data - mean) / std
print("\nStandardized data matrix:")

# Verify the standardization by computing the mean and std of the
normalized data
normalized_mean = np.mean(normalized, axis=0)
normalized_std = np.std(normalized, axis=0)
print("\nMean of standardized columns (should be close to 0):",
normalized_mean)
print("Standard deviation of standardized columns (should be close to
1):", normalized_std)
```

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

10x10 Matrix A (A[i,j] = i + j):
[[ 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]
 [10 11 12 13 14 15 16 17 18 19]
 [11 12 13 14 15 16 17 18 19 20]]

Fake dataset (50x5) created:

Mean of each column: [1.42888415 1.72598083 1.68493738 1.75926977
2.23499822]
Standard deviation of each column: [1.50325164 1.85361678 2.90820094
2.03346829 2.58967922]

Standardized data matrix:

Mean of standardized columns (should be close to 0): [ 1.75415238e-16
-2.70894418e-16 -5.88418203e-17 -3.55271368e-17
  3.81916720e-16]
Standard deviation of standardized columns (should be close to 1): [1.
1. 1. 1. 1.]
```