In [516…
```
import numpy as np
```

Add a cell to create a function and name it my_function_firstname, where firstname is your first name. Let the function return an integer value stored in one byte i.e. 'int8' of (4x)*(3y). Where x is the number of rows and y is the number of columns. Use np.fromfunction() to generate three elements each are two by six using the my_fuction_firstname.

In [517…
```
def my_function_renee(z, x, y):
    return np.int8((4*x) * (3*y))

result_array = np.fromfunction(my_function_renee, (3, 2, 6), dtype=int)


print(result_array)
```

```
[[[ 0  0  0  0  0  0]
  [ 0 12 24 36 48 60]]

 [[ 0  0  0  0  0  0]
  [ 0 12 24 36 48 60]]

 [[ 0  0  0  0  0  0]
  [ 0 12 24 36 48 60]]]
```

Inspect the code under this section copy it, add a cell to extract values 16,17,18

In [518…
```
b = np.arange(48).reshape(4, 12)
b
```

Out[518]:
```
array([[ 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, 32, 33, 34, 35],
       [36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]])
```

In [519…
```
b[1,4:7]
```

Out[519]:
```
array([16, 17, 18])
```

Inspect the code under this section copy it, then add a cell to iterate over c and print the Boolean values for items equivalent to zeros.

In [520…
```
c = np.arange(24).reshape(2, 3, 4)

result = np.zeros_like(c, dtype=bool)

for i in range(c.shape[0]):
    for j in range(c.shape[1]):
        for k in range(c.shape[2]):
            result[i, j, k] = (c[i, j, k] == 0)

print(result)
```

```
[[[ True False False False]
  [False False False False]
  [False False False False]]

 [[False False False False]
  [False False False False]
  [False False False False]]]
```

Inspect the code under this section copy it, then add a cell to create a variable name it q5_firstname where firstname is your firstname and vertically stack q1 and q2 and print the output.

In [521…
```
q1 = np.full((3,4), 1.0)
q2 = np.full((3,4), 2.0)
q3 = np.full((3,4), 3.0)
q5_renee = np.vstack((q1, q2))
q5_renee
```

Out[521]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.]])
```

Inspect the code under this section copy it, then add a cell to create a variable name it q8_firstname where firstname is your firstname , concatenate q1 and q3 and print the results.

In [522…
```
q8_renee = np.concatenate((q1, q3), axis=0)
q8_renee
```

Out[522]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.]])
```

Inspect the code under this section copy it, then add a cell and create a variable named t_firstname where firstname is your name, let the variable hold any ndaray size 2 by 7 with zero values, print the result then transpose and print the result.

In [523…
```
t_renee = np.zeros((2, 7))
t_renee
```

Out[523]:
```
array([[0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.]])
```

In [524…
```
t_renee_t = t_renee.transpose(1, 0)
t_renee_t
```

Out[524]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

Inspect the code under this section copy it, then add a cell to create 2 ndarys name the first a1 and the second a2. Both arrays should contain numbers in the range 0 to 8, inclusive . Print a1 and a2. Reshape a1 to a 2 by 4. Reshape a2 to a 4 by 2. Create a new variable a3 _first name where firstname is your first name which holds the dot product of a1 and a2 name it a3 and print the output of a3_firstname, then the shape of a3_first name.

In [525…
```
a1 = np.arange(1, 9).reshape(2, 4)
a1
```

Out[525]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [526…
```
a2 = np.arange(1, 9).reshape(4,2)
a2
```

Out[526]:
```
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

In [527…
```
a3_renee = np.dot(a1, a2)
a3_renee
```

Out[527]:
```
array([[ 50,  60],
       [114, 140]])
```

Add a cell to create a new 4 by 4 ndaray with values between 0 and 15, name the variable that holds the array your first name, print the array and the inverse of the array.

In [528…
```
renee = np.arange(0, 16).reshape(4, 4)
renee
```

Out[528]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [529…
```
import numpy.linalg as la
la.inv(renee)
```

Out[529]:
```
array([[ 9.00719925e+14, -4.50359963e+14, -1.80143985e+15,
         1.35107989e+15],
       [-2.40191980e+15,  2.70215978e+15,  1.80143985e+15,
        -2.10167983e+15],
       [ 2.10167983e+15, -4.05323966e+15,  1.80143985e+15,
         1.50119988e+14],
       [-6.00479950e+14,  1.80143985e+15, -1.80143985e+15,
         6.00479950e+14]])
```

Add a cell to create a 4 by 4 identity array.

In [530…
```
renee.dot(la.inv(renee))
```

Out[530]:
```
array([[ 1.5   , -1.    ,  0.5   , -0.1875],
       [ 0.    ,  0.    ,  0.5   ,  0.    ],
       [ 0.    ,  0.    , -0.5   ,  0.    ],
       [ 0.    ,  8.    , -2.5   ,  4.    ]])
```

Add a cell to create a 3 by 3 matrix with values generated randomly then printout the determinant of the matrix.

In [531…
```
# Add a cell to create a 3 by 3 matrix with values generated randomly then printout
renee_ran  = np.random.random((3, 3))
renee_ran
```

Out[531]:
```
array([[0.47608431, 0.53322432, 0.8157361 ],
       [0.7702506 , 0.351193  , 0.61968497],
       [0.82030597, 0.33480859, 0.34076741]])
```

In [532…
```
la.det(renee_ran)
```

Out[532]:
```
0.06466073417273484
```

Add a cell to create a 4 by 4 matrix with values generated randomly, assign the matrix to a variable named e_firstname. Printout the Eigenvalue and eigenvectors of the matrix.

In [533…
```
e_renee = np.random.random((4, 4))
e_renee
```

Out[533]:
```
array([[0.67169205, 0.41717689, 0.81194731, 0.50528627],
       [0.96721422, 0.87172345, 0.71162595, 0.85097868],
       [0.20192216, 0.66186834, 0.96978676, 0.18781366],
       [0.01355694, 0.05136291, 0.801518  , 0.86767052]])
```

In [534…
```
eigenvalues, eigenvectors = la.eig(e_renee)
```

In [535…
```
eigenvalues
```

Out[535]:
```
array([2.32939342+0.j        , 0.33671302+0.45400688j,
       0.33671302-0.45400688j, 0.37805332+0.j        ])
```

In [536…
```
eigenvectors
```

Out[536]:
```
array([[ 0.48080689+0.j        , -0.04758003+0.10307666j,
         -0.04758003-0.10307666j, -0.46026052+0.j        ],
       [ 0.69995113+0.j        ,  0.62120279+0.j        ,
         0.62120279-0.j        ,  0.3351084 +0.j        ],
       [ 0.45026923+0.j        , -0.30815334-0.42216823j,
        -0.30815334+0.42216823j, -0.43848766+0.j        ],
       [ 0.27595428+0.j        , -0.07877986+0.56729857j,
        -0.07877986-0.56729857j,  0.69540721+0.j        ]])
```

Add a cell to solve the following linear equations: 2x+4y+z =12 3x+8y+2z =16 X+2y+3z = 3 Check the results using the allcolse method.

In [537…
```
coeffs  = np.array([[2, 4, 1], [3, 8, 2], [1, 2, 3]])
depvars = np.array([12, 16, 3])
solution = la.solve(coeffs, depvars)
solution
```

Out[537]:
```
array([ 8. , -0.7, -1.2])
```

In [538…
```
coeffs.dot(solution), depvars
```

Out[538]:
```
(array([12., 16.,  3.]), array([12, 16,  3]))
```

In [539…
```
np.allclose(coeffs.dot(solution), depvars)
```

Out[539]:
```
True
```