# PGE 310 2021 Fall Homework 3

This homework is divided into 2 parts:

- In Problem 1 you will practice plotting while using a fun linear algebra application, the rotation matrix.
- Problem 2 will help you practice indexing and slicing NumPy arrays in 1D, 2D, and 3D.

The link to the assignment on GitHub is posted on Canvas.

Please submit this assignment as a Python file (.py) on Github classroom. For more comprehensive instructions on how to test and submit your code, refer to the posted instructions document on Canvas.

Indexing and slicing can be tricky, even for seasoned Python pros, but it's one of the most useful skills to learn! And as always, we're here to help! Feel free to contact Dr. P, Alex, Bernie, or Ziming with problems, questions, or comments.

In [1]:
```
# Import numpy and matplotlib.pyplot
import numpy as np
import matplotlib.pyplot as plt
```

# Problem 1: Rotating the Batman Symbol

Batman needs your help! You've been hired as a software developer intern for the Batcave, and the first order of business is to make a logo that can be rotated to a specified angle!

In the first cell, we loaded the x-y data points of the logo from the batman.csv file included in the Homework 3 repository (repo).

In the proceeding cells, please complete the following:

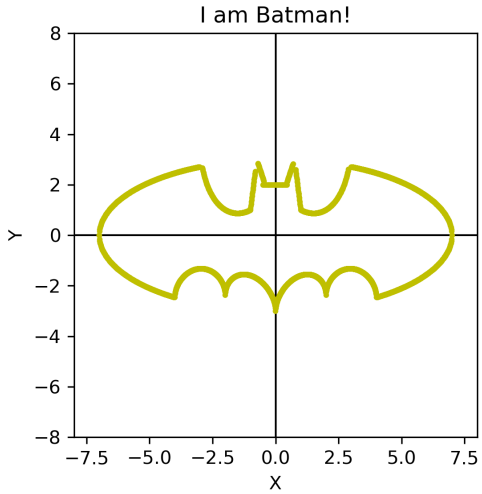
- In `rotate_image()` create a 2D rotation matrix (see below), `R`, and multiply `data` by the rotation matrix, `R`.
- Rotate the data points by 60 degrees using `rotate_image()`.
- In two separate figures, plot the original data and the rotated data with the following attributes, to recreate the example images below:
  - Plot the coordinates, `batman_coord`
  - Set the axis to 'square' formatting
  - Set both the x and y limits to [-8, 8]
  - Label both the x and y axes
  - Add a title

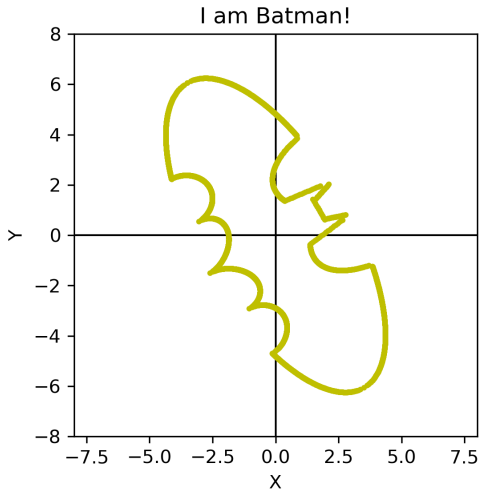
Also note how we are plotting the logo both times with a function. This will prevent us from repeating/copying code!

**The 2D rotation matrix looks like the following:**

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

This is a very useful piece of applied linear algebra. The 2D rotation matrix can be multiplied with any 2D data set with columns of x and y points, and the points will be rotated in space by the specified angle of rotation, $\theta$.




In [ ]:

```python
def rotate_image_2d(data, theta):
    # data = a 2D array with a column of x data and a column of y data
    # theta = the rotation angle in degrees

    theta = np.radians(theta)  # Convert degrees to radians

    # Create rotation matrix
    R =

    # Apply rotation matrix to the data
    rotated_image =

    return rotated_image
```

```
def batman_plot(batman_coord):
    plt.figure(dpi=300)  # Initialize new figure
    plt.plot([0, 0], [-8, 8], 'k-', linewidth=1)  # Plot vertical line for y axi
    plt.plot([-8, 8], [0, 0], 'k-', linewidth=1)  # Plot horizontal line for x a
    # Add your plotting code here

    return


# Load data for the Batman logo, columns of x and y points
batman_data = np.genfromtxt('batman.csv', delimiter=',')

# Set the rotation angle
rotation_angle =

# Call the rotation function and save the rotated data
batman_data_rotated =

# Call the plotting functions
batman_plot(batman_data)
batman_plot(batman_data_rotated)
```

# Problem 2: Slicing and Indexing in Python

Slicing and indexing are very important tools for coding in Python. This problem will guide you through indexing 1D, 2D, and 3D arrays along with the different types of elements, rows, columns, planes, and subsets. **In our questions, we will use base-1 indexing: we will only refer to elements and not indices. For example, if we ask you to index the 3rd element of `a`, this is what we mean:**

```
a = np.array([1, 4, 6, 7, 9, 10])
third_element_of_a = a[2]  # a[2] = 6
```

In other words, we want you to convert what we say to Python indexing notation (base-0).

**Please do not change the variable names, as the automatic grader will be checking the names we provided.**

## 2.1 Indexing in 1 Dimension

In the cell below, we created a 1D array using the NumPy function `np.arange()`:

In [2]:
```
array_1d = np.arange(0,100,10)
print('array_1d = \n', array_1d, '\n')
```

```
array_1d =
 [ 0 10 20 30 40 50 60 70 80 90]
```

In the next cells, please do the following:

- Problem 2.1a: index the 5th element of `array_1d`.
- Problem 2.1b: index the subset of the 4th through the 7th elements (inclusive) of `array_1d` (Please assume inclusive when the word "through" is used for the remaining the questions).

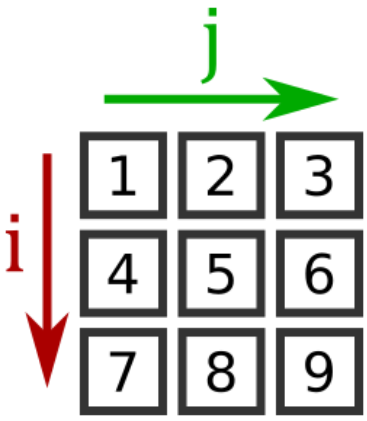
In [ ]:
```
# Problem 2.1a
array_1d_index =
print('Problem 2.1a: \n', array_1d_index, '\n')
```

In [ ]:
```
# Problem 2.1b
array_1d_slice =
print('Problem 2.1b: \n', array_1d_slice, '\n')
```

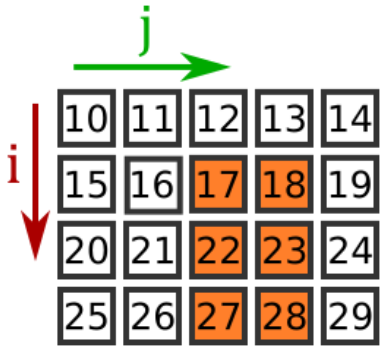
## 2.2 Indexing in 2 Dimensions

Before we get started, let's go over some figures that will help clarify 2D indexing.

Below is a figure of a 3x3 Python array with the i and j directions shown from the origin. Note that the 0 index of the array starts at the top left corner. Also remember that Python references row indices (i) first and then column indices (j) when indexing a 2D array. This can also be viewed as looking in the y direction first and then the x direction. For example, if you wanted to index the number 6 shown below, you would want the indices of the 2nd row and 3rd column; or in Python indexing the row index would be 1 and the column index would be 2.




Below is an exmple of a 4x5 array with a subset highlighted in orange. If you wanted to slice this subsection, you would want the indices for rows 2 through 4 and columns 3 through 4; or in Python indexing notation, the row indices 1 through 3 and column indices 2 through 3.




In the cell below, we created a 2D array using the NumPy function `np.linspace()`:

In [3]:
```
array_2d = np.linspace(90, 42, 25).reshape((5,5))
print('array_2d = \n', array_2d, '\n')
```

```
array_2d =
 [[90. 88. 86. 84. 82.]
 [80. 78. 76. 74. 72.]
 [70. 68. 66. 64. 62.]
 [60. 58. 56. 54. 52.]
 [50. 48. 46. 44. 42.]]
```

In the next cells, please do the following:

- Problem 2.2a: index the element of the 3rd row and 4th column `array_2d`.
- Problem 2.2b: index the entire 4th row of `array_2d`.
- Problem 2.2c: index the entire 2nd column of `array_2d`.
- Problem 2.2d: slice the subset spanning from rows 1 through 3 and columns 3 through 4 of `array_2d`.
- Problem 2.2e: index every other row and all the columns of `array_2d`.

In [ ]:
```
# Problem 2.2a
array_2d_index =
print('Problem 2.2a: \n', array_2d_index, '\n')
```

In [ ]:
```
# Problem 2.2b
array_2d_row_slice =
print('Problem 2.2b: \n', array_2d_row_slice, '\n')
```

In [ ]:
```
# Problem 2.2c
array_2d_col_slice =
print('Problem 2.2c: \n', array_2d_col_slice, '\n')
```

In [ ]:
```
# Problem 2.2d
array_2d_subset =
print('Problem 2.2d: \n', array_2d_subset, '\n')
```

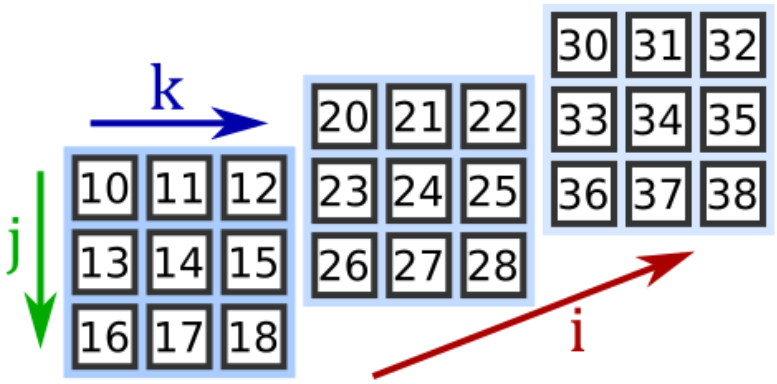
In [ ]:
```
# Problem 2.2e
```

```
array_2d_skip_rows =
print('Problem 2.2e: \n', array_2d_skip_rows, '\n')
```

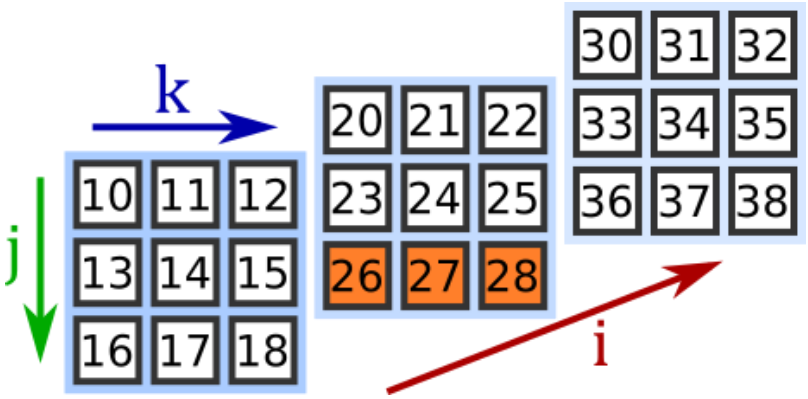
## 2.3 Indexing in 3 Dimensions

Before we start this last section, some figures about 3D indexing will be useful.

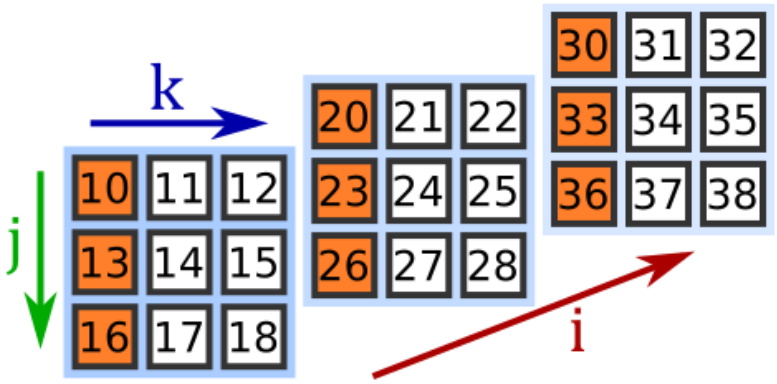
The most important thing about indexing in 3D is that Python references the coordinate directions in the following order: i, then j, then k. This can also be viewed as looking at slices, then rows, then columns. The i direction is also commonly referred to with a number of slices. For example, the 2nd slice of the 3D array below is the middle 2D array beginning with 20. The figure below shows the directions in terms of i, j, and k. Also note that the origin is the top left corner of the first slice: the origin is the number 10 located at Python indices i = 0, j = 0, k = 0. The directions i, j, k as shown can also be viewed as the z, y, and x directions respectively, and this notation is common in Python modules that handle 3D data. But, as always, the definition of the coordinate system is arbitrary and depends on how the user/programmer defines it.



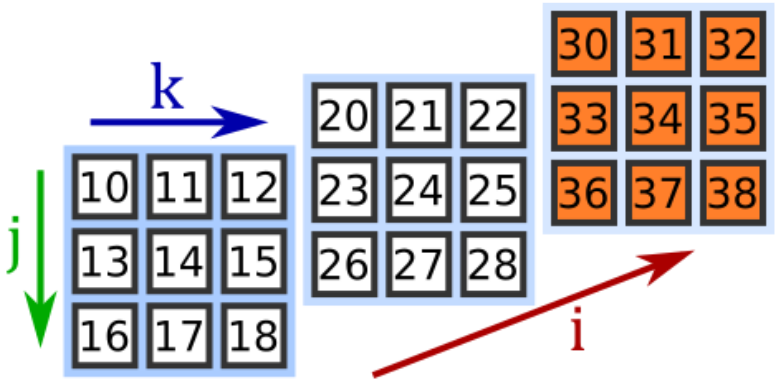

This next image references how to slice a row from a 3D array. This row is located in the 2nd slice, 3rd row, and all the columns. In Python indexing, this would be i = 1, j = 2, k = all elements.



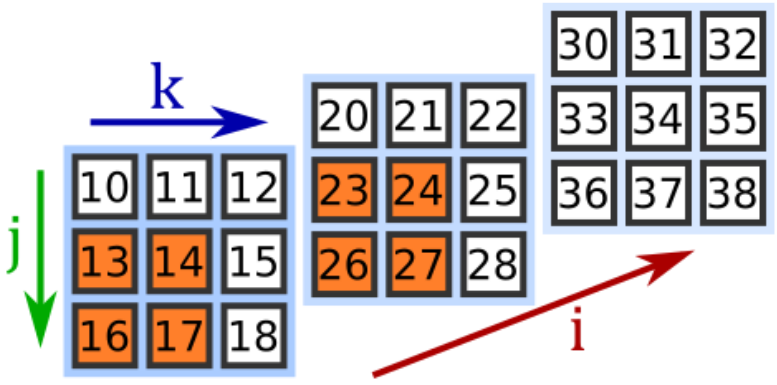

Next is an example of slicing a column slice. This highlighted section is located in all the slices, all the rows, and the 1st column. In Python indexing, this would be i = all elements, j = all elements, k =0.




Now we will show an example of slicing an entire slice (or a z-slice). The highlighted section is located in the 3rd slice, all the rows, and all the columns. In Python indexing, this would be i = 2, j = all elements, k = all elements.




Finally, here is an example of slicing a subset. The highlighted section is located in slices 1 through 2, rows 2 through 3, and columns 1 through 2. In Python indexing, this would be i = 0 through 1, j = 1 through 2, k = 0 through 1.




For this final problem, we created a 3x3x3 3D array using the NumPy function `np.arange()`:

In [4]:
```
array_3d = np.arange(1, 28).reshape((3,3,3))
print('array_3d = \n', array_3d, '\n')
```

```
array_3d =
 [[[ 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]]]
```

In the next cells, please do the following:

- Problem 2.3a: index the element of the 3rd slice, 3rd row, and 2nd column of `array_3d`.
- Problem 2.3b: slice the elements of a column slice including all the slices, all the rows, and the 3rd column of `array_3d`.
- Problem 2.3c: slice the elements of a row slice including all the slices, the 1st row, and all the columns of `array_3d`.
- Problem 2.3d: slice the elements of a z-slice including the 2nd slice, all the rows, and all the columns of `array_3d`.
- Problem 2.3e: slice the elements of a z-column which includes all the slices, the 2nd row, and the 3rd column of `array_3d`.
- Problem 2.3f: slice the elements of a subest which includes slices 2 through 3, rows 2 through 3, columns 1 through 2 of `array_3d`.

In [ ]:
```
# Problem 2.3a
array_3d_index =
print('Problem 2.3a: \n', array_3d_index, '\n')
```

In [ ]:
```
# Problem 2.3b
array_3d_x_slice =
print('Problem 2.3b: \n', array_3d_x_slice, '\n')
```

In [ ]:
```
# Problem 2.3c
array_3d_y_slice =
print('Problem 2.3c: \n', array_3d_y_slice, '\n')
```

In [ ]:
```
# Problem 2.3d
array_3d_z_slice =
print('Problem 2.3d: \n', array_3d_z_slice, '\n')
```

In [ ]:
```
# Problem 2.3e
array_3d_z_col =
print('Problem 2.3e: \n', array_3d_z_col, '\n')
```

In [ ]:
```
# Problem 2.3f
array_3d_subset =
print('Problem 2.3f: \n', array_3d_subset, '\n')
```