In [1]:
```
import numpy as np
```

Matrix

A matrix is a rectangular array of numbers. We can define the dimensions of the vector using the following notation,
N X M, where N is the rows in the matrix, and M is the columns in the Matrix.

In Python we can represent the Matrix using numpy's arrays.

In [2]:
```
np.array([[1,2,3],[4,5,6],[7,8,9]])
```

Out[2]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Here we have a 3x3 Matrix. We have 3 columns, and 3 rows.

We refer to elements by using a specific syntax. We use the subscript i,j syntax like this $A_{i,j}$. If we refer to our Matrix as the Matrix A, we can say that the $A_{1,1}$ element is the number 1. I refers to the row, whereas j refers to the column

Spot Check

In [3]:
```
np.array([[1,2,3,4,5,6],[7,8,9,10,11,12],[13,14,15,16,17,18]])
```

Out[3]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18]])
```

In this Matrix, using $A_{i,j}$ syntax, How do we represent the following three values?

3 = $A_{1,3}$

11 = $A_{2,5}$

18 = $A_{3,6}$

The Vector

The vector is a special type of matrix. We can say that it is an n * 1 matrix. It will only contain 1 column, by definition.

In [4]:
```
np.array([[1],[2],[3]])
```

Out[4]:
```
array([[1],
```

```
       [2],
       [3]])
```

We refer to this as a 3 dimensional vector? Why is that? It has three layers of depth so to speak. Vectors can be 0 indexed, or they can be 1 indexed. Think of this of the way we represent arrays in python. We use similar Y_i to reference which element we are concerned with.

Matrix Addition

We can add Matrices together, only when their dimensions match. As long as that much is true, we can perform addition on them. The resultant matrix will be of the same dimensions as the two component matrices. To get the resultant matrix, you need to add the elements in the same space. Let's take a look at some examples.

In [5]:
```
first_matrix = np.array([[1],[2],[3]])
```

In [6]:
```
second_matrix = np.array([[3],[2],[1]])
```

In [7]:
```
first_matrix+second_matrix
```

Out[7]:
```
array([[4],
       [4],
       [4]])
```

What happens if we try to add together vectors with different dimensions?

In [8]:
```
first_matrix = np.array([[1],[2],[3]])
```

In [9]:
```
second_matrix = np.array([[3],[2],[1],[5]])
```

In [10]:
```
first_matrix+second_matrix
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-5b9769d07775> in <module>
----> 1 first_matrix+second_matrix

ValueError: operands could not be broadcast together with shapes (3,1) (4,1)
```

We get an error. The condition of needing the matrices to have the same dimensions must be met.

Matrix and Scalar Operations

When we say we are multiplying a matrix by a scalar, we are applying a

constant change across all values within a matrix. Let's see it in action.

In [ ]:
```
matrix_scalar = np.array([[1,2,3],[3,4,5],[6,7,8]])
```

In [ ]:
```
matrix_scalar
```

In [ ]:
```
matrix_scalar * 5
```

We can see that it took each value in the matrix, and it multiplied it by the scalar value.

As we can multiply, we can divide as well. Remember, dividing by a scalar is the same as multiplying by 1 over said scalar.

In [ ]:
```
matrix_scalar / 5
```

In [ ]:
```
matrix_scalar * .20
```

Matrix by Vector Multiplication

Now let's venture into the territory of multiplying a Matrix by a Vector. Let's see the result first.

In [ ]:
```
matrix_scalar = np.array([[1,2,3],[3,4,5],[6,7,8]])
```

In [ ]:
```
matrix_scalar
```

In [ ]:
```
vector_to_multiply = np.array([[1],[2],[3]])
```

In [ ]:
```
vector_to_multiply
```

In [ ]:
```
np.dot(matrix_scalar,vector_to_multiply)
```

Let's see how this worked out in terms of dimensions.

Matrix scalar was a 3x3 Matrix.
Vector to multiply was a 3x1 Matrix.

```
 Our result, was a 3x1 Matrix.


Let's set the rules.
Matrix A is a Matrix of M rows and N Columns.
Vector X is a Matrix of N rows and 1 Columns.
The result is a M dimensional Vector Y.
```

To get Y_i multiply A's ith row with the elements of vector X, and add them together.

In [ ]:
```
matrix_scalar
```

In [ ]:
```
vector_to_multiply
```

For example, for the first row in matrix scalar we do.

(1 *1)* + *(2* 2) + (3 * 3) = 14

That's the first component of our Vector Y.

Then we do

(3 *1)* + *(4* 2) + (5 *3) = 26

That's our second component

Our third and last component is

(6 *1)* + *(7* 2) + (8 * 3) = 44

Our result ends up being the vector

In [ ]:
```
np.array([[14],[26],[44]])
```

In [ ]:
```
Which matches the previous result.
```

Rule of Vector Matrix Multiplication. The amount of columns in the Matrix has to match the amount of rows in the Vector.

In [ ]:
```
np.dot(matrix_scalar,np.array([[1],[2]]))
```

We can see that this doesn't work.

Spot Check

In [ ]:
```
matrix_to_mult = np.array([[1,2,1,5],[0,3,0,4],[-1,-2,0,0]])
```

In [ ]:
```
vector_to_mult = np.array([[1],[3],[2],[1]])
```

In [ ]:
```
np.dot(matrix_to_mult,vector_to_mult)
```

In [ ]:
```
# Why was the result a 3 dimensional Vector?
```

Matrix Matrix Multiplication

What if we want to multiply two Matrices together? How can we do this?

In [ ]:
```
#First let's see the result.
```

In [ ]:
```
matrix_a = np.array([[1,3,2],[4,0,1]])
```

In [ ]:
```
matrix_b = np.array([[1,3],[0,1],[5,2]])
```

In [ ]:
```
np.dot(matrix_a,matrix_b)
```

Let's take a look once again at dimensions first.

matrix_a is a 2 X 3 Matrix. Let's say it has M rows, and N columns.

matrix_b is a 3 X 2 Matrix. Let's say it has N rows, and O columns.

Our result is a 2X2 Matrix. It has M rows, and O columns.

The more formal definition is this.

The ith column of the result maxtrix is obtained by multiplying Matrix_a with the ith column of B.

Let's work through it by hand first.

In [ ]:
```
matrix_a
```

In [ ]:
```
matrix_b
```

B is made up of 2 Vectors. First, we take the the first vector of B.

[[1],[0],[5]]. We will multiply Matrix A by this.

(1 *1)* + *(3* 0) + (2 * 5) = 11

That makes up the first component of the result vector.

(4 *1)* + *(0* 0) + (5 * 1) = 9

This makes up the second component of the result Vector. We are left with the Vector [[11],[9]]

Now let's multiply A by the second Vector, [[3],[1],[2]]

(3 *1)* + *(1* 3) + (2 * 2) = 10

That makes up the first component of the result vector.

(4 *3)* + *(1* 0) + (1 * 2) = 14

This makes up the second component of the result Vector. We are left with the Vector [[10],[14]]

Now we can line up those vectors and what result do we get? The same Matrix as before.

The rule of Matrix Matrix Multiplication is that the columns of the first matrix has to match with the rows of the second matrix.

Spot Check

What is the result of the multiplying these 2 matrices together?

In [ ]:
```
matrix_a = np.array([[1,2,3,4],[5,6,7,8]])
```

In [ ]:
```
matrix_b = np.array([[1,2],[3,4],[5,6],[7,8]])
```

In [ ]:
```
np.dot(matrix_a,matrix_b)
```

BREAK

Now we are going to go over some of the special cases, and some of the rules of matrices.

```
Formally we say that Matrix multiplication is Associative.
We say that Matrix multiplication is not commutative.
```

The Associative property of multiplication states that (A *B)* C is equal to A *(B*C)

```
This IS true of Matrices, and you can try it out for yourself.
```

The Commutative Property of multiplication states that A *B* = *B* A. This is NOT True for matrices, and you can try this our yourself. (In general this is not true, in specific cases it is.)

In [ ]:

Identity Matrix

This special type of matrix is a matrix that when multiplied to a matrix returns the same matrix. It's equivalent to "multiplying the matrix by 1".

It's denoted by I or I_{n*n}, where n denotes the dimensions.

In [ ]:
```
The identity matrix is as follows.
```

In [ ]:
```
np.identity(2)
```

In [ ]:
```
np.identity(3)
```

We can see that it is a 1 across the diagonals, and 1 everywhere else.

Let's formally define it.

A is some m * n Matrix.

A $I_{(n)}=I_{(m)}$ A = A

This is one of the special cases where the commutative property is true. In the case of square matrices multiplied by the identity matrix.

Inverse

```
Formally defined, the Inverse of a Matrix is the matrix we can multiply a
Matrix by in order to get 1. Let's think of it in Math. 3 * X = 1. What
is x? X is 3^-1. What about 12 * X = 1? Well it's 12^-1
```

```
It follows the same type of logic in Matrix operations. What can we
```

multiply a matrix by in order to get 1? Turns out that there are a few rules. Only Square Matrices can have inverses, and not all matrices have inverses. Why and how to get these is beyond the scope of what we're going to do.

Formally defined it is as follows.

$A\ A^{-1} = A^{-1}\ A = 1$

We will not formally go over how to calculate the inverse. It's not particularly difficult, but if you want more context look it up. In Python we can do it in numpy.

In [ ]:
```
test_matrix = np.array([[1,2],[3,4]])
```

In [ ]:
```
np.linalg.inv(test_matrix)
```

In [ ]:
```
test_inverse = np.linalg.inv(test_matrix)
```

In [ ]:
```
np.dot(test_matrix,test_inverse)
```

We can see that some values are zero, while some are very close to zero this is ok.

Why do we want to calculate the inverse?

It's because we can't divide matrices by each other. Assume this problem.

A * B = C where these are 3 Matrices.

If we have the values of Matrix A and C, how can we get the value of Matrix B?

We can multiply both sides by A^{-1} to get $BAA^{-1} = CA^{-1}$

We can reduce, and then we get. B = CA^{-1}

Some extra notes.

Matrices without an inverse are called degenerate or singular matrices.

One matrix that doesn't have an inverse is the matrix of all zeroes.

Matrix Transposing

This is a special action we can do on a Matrix. We are flipping a matrix along a 45 degree angle and mirroring it. Let's take a look

In [ ]:
```
matrix_to_transpose = np.array([[1,2,0],[3,5,9]])
```

In [ ]:
```
np.transpose(matrix_to_transpose)
```

This is a bit difficult to visualize but imagine that 45 Degree line. Then you're mirroring across it and flipping it.

More formally it is defined as follows.

Let A be an m*n matrix and let B = A^T

B is an n*m matrix, and $B_{i,j} = A_{j,i}$