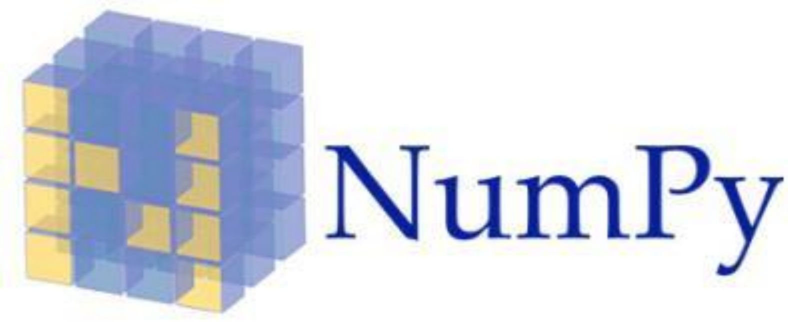
NumPy - a library for matrices



What does NumPy allow you to do?

Numpy is a Python library for manipulation of matrices:

Matrix manipulation: instantiation, edition, reshaping, ...

Matrix operations: addition, multiplication, inverse, determinant, ...

Linear algebra operations: equation solving, matrix decomposition, eigenvalues, ...

Other basic math functions: trigonometry, random number generation, …

A lot of very powerful math functions for statistics, polynomials, logic, financial, … optimized for Python.

NumPy Arrays

A vector (4 5 6) can be created by calling `np.array`:

```
>>> A = np.array([4, 5, 6])
```

You can also create a matrix by passing a multidimensional array to `np.array`:

```
>>> A = np.array([[4, 5], [5, 6]])
```

Basic operations

You can add/subtract two matrices with + and −:

```
>>> a = np.array([20, 30, 40, 50])
>>> b = np.array([0, 1, 2, 3])
>>> c = a - b
    array([20, 29, 38, 47])
```

You can also multiply a matrix by a scalar

```
>>> A = np.array([[1, 2],[2, 3]])
>>> B = 10*A
    array([[ 10,  20],
           [ 20,  30]])
```

Basic operations

You can add/subtract two matrices with + and −:

```
>>> a = np.array([20, 30, 40, 50])
>>> b = np.array([0, 1, 2, 3])
>>> c = a - b
    array([20, 29, 38, 47])
```

You can also divide by a scalar, compute the square or the square root of each element, etc.

You can also multiply a matrix by a scalar

```
>>> A = np.array([[1, 2],[2, 3]])
>>> B = 10*A
    array([[ 10,  20],
           [ 20,  30]])
```

Your turn!

Compute ***C*** with:

$$A = \begin{pmatrix} 1 & 3 & -1 \\ 2 & 0 & -1 \\ 0 & -2 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 3 & 0 \\ 1 & -2 & 4 \end{pmatrix}$$

$$C = 2\,A - 0.5\,B$$

Special matrix creation

A matrix full of 0s can be created by calling `np.zeros` :

```
>>> np.zeros((3,4))
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

The same goes with ones

```
>>> A = np.ones((2,3))
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

You can also call an incrementing vector by calling `np.arange`:

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
```

Axis-wise operations

Each dimension is an axis, some operations can be performed axis-wise:

```
>>> A = np.array([[ 0,  1,  2,  3],
                  [ 4,  5,  6,  7],
                  [ 8,  9, 10, 11]])

>>> np.sum(A, axis=0)
array([12, 15, 18, 21])

>>> np.max(A, axis=1)
array([ 3,  7, 11])
```

Axis-wise operations

Each dimension is an axis, some operations can be performed axis-wise:

```
>>> A = np.array([[ 0,  1,  2,  3],
                  [ 4,  5,  6,  7],
                  [ 8,  9, 10, 11]])

>>> np.sum(A, axis=0)
array([12, 15, 18, 21])

>>> np.max(A, axis=1)
array([ 3,  7, 11])
```

You can also compute the np.min() and np.mean()

Your turn!

Instantiating ONLY one matrix ***M***, compute ***a*** and ***b*** where

- ***a*** is the sum of all odd numbers from 0 to 500
- ***b*** is the sum of all even numbers from 0 to 500

Useful features

Number of dimensions:

```
>>> A = np.array([[4, 5, 6], [5, 6, 7]])
>>> np.ndim(A)
2
```

Shape:

```
>>> np.shape(A)
(2, 3)
```

Number of items:

```
>>> np.size(A)
6
```

Random (1/2)

A random number between 0 (inclusive) and 1 (exclusive) can be generated using `np.random.rand()`:

```
>>> np.random.rand()
0.22689366747057116
```

Similarly, use this to generate a matrix of numbers by specifying the shape:

```
>>> np.random.rand(2, 3)
array([[ 0.512432,  0.725554, 0.904192],
       [ 0.223564,  0.424663, 0.681788]])
```

Random (2/2)

A single integer, or matrix of integers, can also be generated by specifying the minimum value, maximum value, and shape:

```
>>> np.random.randint(0, 4)
3

>>> np.random.randint(0, 4, (2, 2))
array([[0, 2],
       [1, 1]])
```

Your turn!

1) Instantiate a random matrix ***M*** of numbers between 0 and 5, with random numbers of rows ***I*** and columns ***L*** between 1 and 10.

2) Compute ***A***, the vector of column averages, and ***a***, the total average of the matrix.

3) Output ***I*** and ***L***.

Matrix products

Element-wise product

```
>>> A = np.array( [[1,1],[0,1]] )
>>> B = np.array( [[2,0],[3,4]] )
>>> C = A*B
array([[2, 0],
       [0, 4]])
```

Dot product:

```
>>> D = np.dot(A, B)
array([[5, 4],
       [3, 4]])
```

Indexing

1 dimensional matrix

```
>>> A = np.array([1,1,0,1])
>>> B = A[1]
1
>>> C = A[2]
0
```

2 dimensional matrix

```
>>> D = np.array( [[1,2], [3,4]] )
>>> E = D[1,0]
3
```

Slicing (1/2)

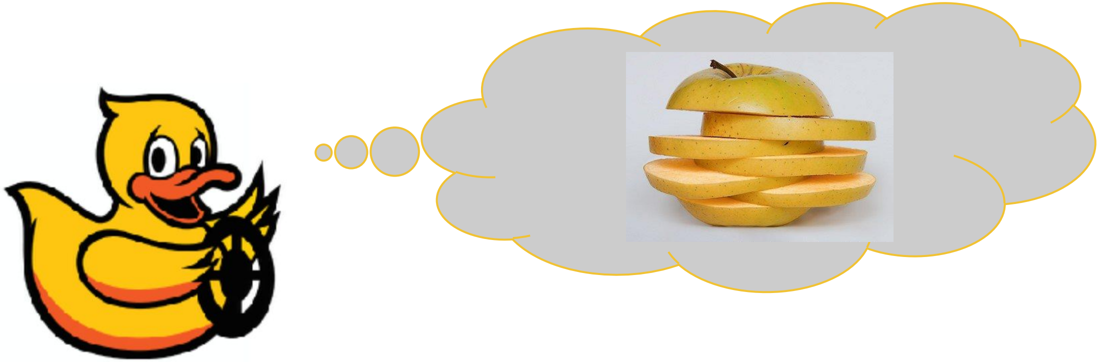
This is used to select a contiguous subset of an array along any dimensions, the “`:`” operator performs this

```
>>> A = np.array([1,1,0,1])
>>> A[1:3]
array([1, 0])
>>> A[:2]
array([1, 1])
```

Slicing (1/2)

This is used to select a contiguous subset of an array along any dimensions, the "`:`" operator performs this

```
>>> A = np.array([1,1,0,1])
>>> A[1:3]
array([1, 0])
>>> A[:2]
array([1, 1])
```



Slicing (2/2)

Multidimensional slicing: what does this do?

```
>>> A = np.random.randint(0, 5, (3, 4))
>>> print(A)
[[2 3 1 4]
 [3 1 1 1]
 [0 3 3 4]]

>>> A[:2,1:3]
array([[3, 1],
       [1, 1]])
```

Your turn!

With:

$$A = \begin{pmatrix} 1 & 3 & -1 \\ 2 & 0 & -1 \\ 0 & -2 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 3 & 0 \\ 1 & -2 & 4 \end{pmatrix}$$

Compute *C*, the elementwise product of *A* and *B*. Output *C_last*, the bottom right element of *C*n

Change *C_last* to 0.

Compute *D*, the dot product of *B* and *C*. Output *D'*, the right-top 2x2 matrix of *D*.