# Lab Instructions - session 1

**Introduction to numpy and matplotlib**

## A review of numpy arrays and matrices + matplotlib

Open an interactive Python environment (python shell, ipython shell, Jupyter notebook, Google Colab), run the following commands, and see the output. Do not close the environment

### Creating numpy arrays

```
>>> l = [1,2,3]
>>> l
>>> import numpy
>>> a = numpy.array(l)
>>> a
>>> a[2] = 300
>>> a

>>> type(l)
>>> type(a)

>>> import numpy as np
>>> a = np.array(l)
>>> a

>>> a = np.zeros(10)
>>> a
>>> a.dtype
>>> a[2] = 4
>>> a

>>> a = np.zeros(10, dtype=np.int64)
>>> a
>>> a.dtype
>>> a = np.ones(10)
>>> a
>>> a = np.ones(10) * -20
>>> a
>>> np.full(10, 222)
>>> a = np.arange(10)
>>> a
>>> 2**a
```

## Numpy array basic properties

```
>>> a = np.ones(10000)
>>> len(a)
>>> a.shape
>>> type(a)
>>> a.size
>>> a.ndim
>>> a.dtype
>>> a.nbytes
>>> a.itemsize

>>> import sys
>>> sys.getsizeof(a)
```

- Why are the outputs of **`a.nbytes`** and **`sys.getsizeof(a)`** different?

## Lists vs. numpy arrays

```
>>> l1 = [1,2,3]
>>> l2 = [4,5,6]
>>> a1 = np.array(l1)
>>> a2 = np.array(l2)
>>> l1+l2
>>> a1+a2
```

## Data types

```
>>> a = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.int64)
>>> b = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.int32)
>>> c = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.int16)
>>> d = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.int8)
>>> e = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.uint8)
>>> print(a.itemsize, b.itemsize, c.itemsize, d.itemsize, e.itemsize)
>>> print(a.nbytes, b.nbytes, c.nbytes, d.nbytes, e.nbytes)

>>> d-4
>>> e-4

>>> f = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.float32)
>>> g = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.float64)
>>> h = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.float128)
>>> print(f.nbytes, g.nbytes, h.nbytes)

>>> l = np.array([False, True, True])
```

```
>>> l.dtype
>>> l = np.array([0, 1, 1], dtype=np.bool)
>>> l.dtype
>>> l.nbytes
```

## Basic operations

```
>>> a = np.array([1,2,3])
>>> b = np.array([4,5,6])
>>> a+b
>>> a-b
>>> a*b
>>> b**a
>>> a + 4
>>> a * 2
>>> a ** 2
>>> a.dtype
>>> a/b        # different in pythons 2.x and 3.x
>>> a//b

>>> a = np.array([1.0,2,3])
>>> a
>>> a.dtype
>>> a / b
>>> a//b    # different in pythons 2.x and 3.x

>>> a = np.array([1,2,3], dtype=np.float64)
>>> a
>>> a.dtype
```

## Slicing

```
>>> a = np.array([0,10,20,30,40, 50, 60, 70, 80, 90, 100])
>>> a
>>> a[2]
>>> a[-2]
>>> a[2:8]
>>> a[2:-1]
>>> a[2:]
>>> a[:8]
>>> a[2:8:2]
>>> a[2:8:-1]
>>> a[8:2:-1]
>>> a[::-1]
>>> a[[1,3,3,4,5]]
```

## 2D Arrays

```
>>>   A = np.zeros((4,6))
>>>   A
>>>   A = np.zeros((4,6), dtype=np.int32)
>>>   A
>>>   A = np.ones((3,7))
>>>   A
>>>   A = np.ones((3,8), dtype=np.uint8)
>>>   A
>>>   np.full((4,3), 50.0)
>>>
>>>   A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>>   A[1,2]
>>>   A[0,-1]
>>>   A[1,2]
>>>   A.shape
>>>   A.shape[0]
>>>   A.shape[1]
>>>   A.shape[::-1]
>>>   A.size
>>>   A.ndim
>>>
>>>   A[0]
>>>   A[1]
>>>   A[0].shape
>>>   A[0,:]
>>>   A[0,:].shape
>>>   A[[0],:]
>>>   A[[0],:].shape
>>>   A[:,2]
>>>   A[:,[2]]
>>>   A[:,2].shape
>>>   A[:,[2]].shape
>>>
>>>   A[1:3]
>>>   A[1:3, :]
>>>   A[:,:3]
>>>   A[:,::2]
>>>   A[:,::-1]
>>>
>>>   r = np.array([0, 1, 0, 2, 2])
>>>   A
>>>   r
>>>   A[r,:]
>>>
```

```
>>>
>>>  A
>>>  A[:,0] = 1
>>>  A
>>>  A[:,0] = [20,30,40]
>>>  A
>>>
>>>  A
>>>  A.T
>>>
>>>  B = np.array([[1,1,1,1], [2,2,2,2], [3,3,3,3]])
>>>  A
>>>  B
>>>  A + B
>>>  A * B
>>>
>>>  np.dot(A,B)
>>>  A.dot(B)
>>>  A @ B
>>>  A.dot(B.T)
>>>
>>>  I = np.eye(3)
>>>  I
>>>
>>>  np.random.random((2,3))
>>>  np.random.random((2,3))
>>>  np.random.random((2,3))
>>>
>>>  np.random.rand(2,3)
>>>  np.random.rand(2,3)
>>>
>>>  np.random.randn(2,3)
>>>  np.random.randn(2,3)
```

Numpy slices are references (not copies)

```
>>> A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>> b = A[:,1]
>>> b
>>> A
>>> b[1] = 10000
>>> b
>>> A

>>> b = A[:,0].copy()
>>> b
>>> b[1] = -20000
```

```
>>> b
>>> A
```

## Masks

```
>>> A = np.array([[1,2,3,4],
                  [5,6,7,8],
                  [9,10,11,12]])
>>> Mask = np.array([[True,  False,  True,  False],
>>>                  [True,  True,   False, False],
>>>                  [False, False,  False,  True]])
>>> Mask.dtype
>>> A
>>> A[Mask]
>>> A[~Mask]
>>> A[Mask] = 222
>>> A
>>> A[~Mask] *= 2
>>> A

>>> A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

>>> B = np.zeros_like(A)
>>> B[Mask] = A[Mask]
>>> B

>>> A > 2
>>> Mask = A < 8
>>> Mask
>>> A[Mask]
>>> A[A < 8]
>>> A[A < 8] += 100
>>> A
```

## Operations on arrays

```
>>> A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>> A.sum()
>>> A.sum(axis=0)
>>> A.sum(axis=1)
>>> A.min()
>>> A.min(axis=0)
>>> A.max(axis=0,keepdims=True)
>>>
>>> A.max(axis=1)
>>> A.max(axis=1,keepdims=True)
```

```
>>> A.mean(axis=1)
>>>
>>> A.prod(axis=0)
```

## Concatenation

```
>>> X = np.array([[1,2],[3,4]])
>>> Y = np.array([[10,20,30],[40,50,60]])
>>> Z = np.array([[7,7],[8,8],[9,9]])
>>> X
>>> Z
>>> np.concatenate((X,Z))
>>> np.concatenate((X,Z), axis=0)
>>> X
>>> Y
>>> np.concatenate((X,Y), axis=1)
>>> np.vstack((X,Z))
>>> np.r_[X,Z]
>>> np.hstack(X,Y)    # error
>>> np.hstack((X,Y))
>>> np.c_[X,Y]
>>> Y
>>> np.tile(Y,(4,3))
```

## Reshaping

```
>>> A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>> A.reshape((4,3))
>>> A.reshape((2,6))
>>> A.reshape((2,7))
>>> A.reshape((1,12))
>>> A.reshape((12,1))
>>> A.reshape((12,))

>>> b = A.ravel()
>>> b
>>> b.shape
>>> b.reshape((2,6))

>>> b
>>> b.shape = (2,6)
>>> b
```

```
>>> f = A.flatten()
>>> r = A.ravel()
>>> f
>>> r

>>> f[0] = 4444
>>> f
>>> A

>>> r[0] = 4444
>>> r
>>> A
```

- What is the difference between **`ravel`** and **`flatten`**? Which one do you think is faster?

## Numpy arrays vs numpy matrices

```
>>> A = np.array([[1,2,3], [1,1,1], [-1,-2,-1]])
>>> A
>>> A*A         # element-wise multiplication
>>> A.dot(A)    # matrix multiplication
>>> A @ A       # matrix multiplication (same as above)
>>>
>>> M = np.matrix([[1,2,3], [1,1,1], [-1,-2,-1]])
>>> M*M
>>> np.multiply(M,M)

>>> M=np.mat(A)
>>> M
>>> M=np.matrix(A)
>>> M
>>> M.T
>>> M.I
>>> M.I * M
>>> M * M.I
>>> M.A
>>> type(M)
>>> type(M.A)

>>> C = np.matrix("1 2; 3 4; 5 6")
>>> C
>>> M*C
```

## N-dimensional arrays

```
>>> A = np.zeros((2,4,3))
>>> A
>>> A.shape

>>> A[:,:,0].shape
>>> A[:,:,0] = [[1,2,3,4],[5,6,7,8]]
>>> A[:,:,1] = [[2,2,2,2],[4,4,4,4]]
>>> A[:,:,2] = [[10,20,30,40],[11,21,31,41]]
>>> A
>>>
>>> A[0,:,:]
>>> A[0]

>>> A[:,1,:]
>>> A[:,1]
>>> A[:,[1]]

>>> A[:,1].shape
>>> A[:,[1]].shape

>>> A[:,:,0]
>>> A[:,:,2]
>>> A[:,2:,2]
>>> A.ravel()
```

## Broadcasting

```
>>> A = np.array([0,1,2,3,4,5,6,7])
>>> A - 10
>>> a = np.array([4])
>>> A * a

>>> A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>> b = np.array([1, 0, 2,-2])
>>> A
>>> b
>>> A-b

>>> c = np.array([1,2,3])
>>> c - c
>>> c - c.reshape((3, 1))
>>> c - c.reshape((-1, 1))
>>> c - c[:, np.newaxis]
```

```
>>>  c - c[:, None]

>>>  A-c
>>>  A-c.reshape((3,1))

>>>  A = np.arange(24).reshape((2,3,4))
>>>  A.shape
>>>  A - 2
>>>  A - np.array([1,2])
>>>  A - np.array([1,2,3,4])
>>>  A - np.array([1,2,3])
>>>  A - np.array([1,2,3]).reshape((3,1))
>>>  A - np.array([1,2])
>>>  A - np.array([1,2]).reshape((2,1,1))
>>>  A - np.array([[1,2,3,4],[5,6,7,8], [9,10,11,12]])
>>>  A - np.array([[1,2,3,4],[5,6,7,8]])
>>>  A - np.array([[1,2,3,4],[5,6,7,8]]).reshape((2,1,4))
```

To get a better understanding of broadcasting, read the following (particularly the broadcasting rules) https://numpy.org/doc/stable/user/basics.broadcasting.html

## Math functions

```
>>>  x = np.arange(0, 2 * np.pi, 0.1)
>>>  x
>>>  y = np.cos(x)
>>>  y
>>>  np.sin(x)
>>>  np.tan(x)


>>>  x = np.linspace(1,8,20)
>>>  x
>>>  x.shape
>>>  np.exp(x)

>>>  np.log(x)
>>>  np.log10(x)
>>>  np.log2(x)
>>>  np.floor(x)
>>>  np.ceil(x)
>>>  np.round(x)

>>>  np.sqrt(x)
>>>  np.arctan(x)
```

- Here you can find a list of numpy math functions:
  - https://numpy.org/doc/stable/reference/routines.math.html

## Plotting with Matplotlib

```
>>> from matplotlib import pyplot as plt
>>>
>>> x = np.arange(0, 2 * np.pi, 0.1)
>>> x
>>> y = np.cos(x)
>>> y
>>> plt.plot(x,y)
>>> plt.show()

>>> plt.plot(x,np.sin(x))
>>> plt.plot(x,np.cos(x))
>>> plt.show()
```

## Reading and displaying images

```
>>> from matplotlib import pyplot as plt

>>> I = plt.imread('masoleh_gray.jpg')
>>> I.shape
>>> I.dtype
>>> plt.imshow(I)
>>> plt.show()
>>> plt.imshow(I,cmap='gray')
>>> plt.show()
>>>
>>> plt.imshow(I[100:200,50:250],cmap='gray')
>>> plt.show()
```

# Task 1 - Practice Vectorization

Consider an arbitrary **A** matrix like

```
A = np.random.rand(200,10)
```

We perform the following operation on **A** to create the matrix **B**.

```
mu = np.zeros(A.shape[1])
for i in range(A.shape[0]):
    mu += A[i]
mu /= A.shape[0]
```

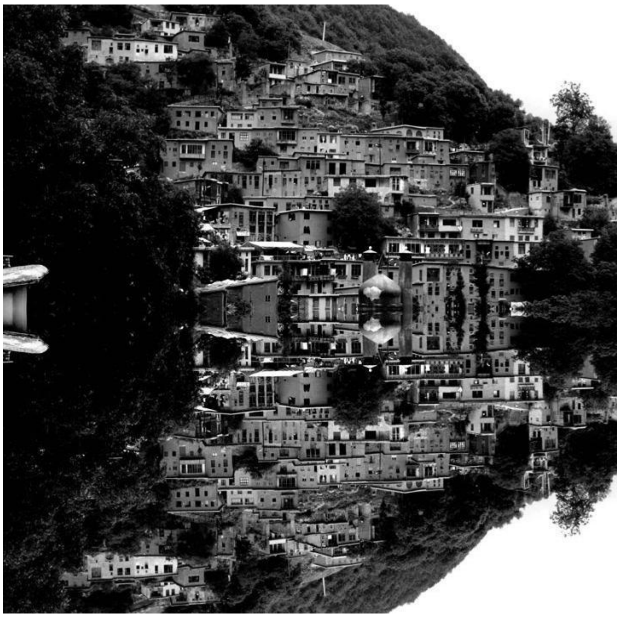
```
B = np.zeros_like(A)
for i in range(A.shape[0]):
    B[i] = A[i] - mu
```

- What does the above piece of code do?
- Write an equivalent program ***without loops***. Do it in just **a single line of code**.

```
B = ...
```

# Task 2

Read the image 'masoleh_gray.jpg' and create a new image by vertically concatenating it with its vertically inverted image (like below). Display the new image.



## References

1. Numpy Quickstart https://numpy.org/doc/stable/user/quickstart.html
2. https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html
3. https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
4. http://cs231n.github.io/python-numpy-tutorial
5. Broadcasting https://numpy.org/doc/stable/user/basics.broadcasting.html