Solutions Lecture 1 (Sections 3.1-3.5)

Make sure to import Numpy to be able to use all its functionality.

```
import numpy as np
```

Note that all the questions below should be solved without using for-loops.

Question 1

Create a 3×2 array M of ones. Make sure the elements have data type `int`.

```
# If you don't specify the dtype, then the array will have
# dtype 'float64' (You can check this with the command: x.dtype)
x = np.ones((3,2),dtype=int)

print(x)
```

```
[[1 1]
 [1 1]
 [1 1]]
```

Question 2

Create the array $x = [2, 4, 6, 8, ..., 100]$ once with `arange()`, and once with `linspace()`.

```
# With arange()
x = np.arange(2,101,2)
print(x)

# With linspace()
x = np.linspace(2,100,50)
print(x)
```

```
[  2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36
  38  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68  70  72
  74  76  78  80  82  84  86  88  90  92  94  96  98 100]
[  2.   4.   6.   8.  10.  12.  14.  16.  18.  20.  22.  24.  26.  28.
  30.  32.  34.  36.  38.  40.  42.  44.  46.  48.  50.  52.  54.  56.
  58.  60.  62.  64.  66.  68.  70.  72.  74.  76.  78.  80.  82.  84.
```

```
  86.   88.   90.   92.   94.   96.   98.  100.]
```

Question 3

Create the array $x = [-1, -0.9, \dots, -0.1, 0, 0.1, \dots, 0.9, 1]$.

```
# With arange()
x = np.arange(-1,1.1,0.1)
print(x)

# With linspace()
x = np.linspace(-1,1,21)
print(x)
```

```
[-1.00000000e+00 -9.00000000e-01 -8.00000000e-01 -7.00000000e-01
 -6.00000000e-01 -5.00000000e-01 -4.00000000e-01 -3.00000000e-01
 -2.00000000e-01 -1.00000000e-01 -2.22044605e-16  1.00000000e-01
  2.00000000e-01  3.00000000e-01  4.00000000e-01  5.00000000e-01
  6.00000000e-01  7.00000000e-01  8.00000000e-01  9.00000000e-01
  1.00000000e+00]
[-1.  -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.   0.1  0.2  0.3
  0.4  0.5  0.6  0.7  0.8  0.9  1. ]
```

Question 4

Consider the two-dimension array below and answer the following questions by using indexing.

```
M = np.array([[1,1,1,1,1],[2,1,2,1,2],[2,1,2,1,2],[2,1,2,1,2],[2,1,2,1,2],[1,1,1,1,1]])

print(M)
```

```
[[1 1 1 1 1]
 [2 1 2 1 2]
 [2 1 2 1 2]
 [2 1 2 1 2]
 [2 1 2 1 2]
 [1 1 1 1 1]]
```

a) Return the odd-numbered rows

```
M_odd = M[1::2]

print(M_odd)
```

```
[[2 1 2 1 2]
 [2 1 2 1 2]
 [1 1 1 1 1]]
```

b) Return the submatrix consisting of elements that are equal to 2.

```
M_2 = M[1:5,0::2]

print(M_2)
```

```
[[2 2 2]
 [2 2 2]
 [2 2 2]
 [2 2 2]]
```

c) Return the submatrix consisting of the rows $0, 1, 5$ and columns $1, 3, 4$.

```
i = [0,1,5]
j = [1,3,4]

print(M[np.ix_(i,j)])
```

```
[[1 1 1]
 [1 1 2]
 [1 1 1]]
```

Question 5

Consider the following array.

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

print(x)
```

```
[ 3  6  4  5  5  5  1  4  2  9  6  7 11 10]
```

a) Use a Boolean mask to access all element whose value is smaller or equal than 4.

```
mask = x <= 4
print(x[mask])
```

```
[3 4 1 4 2]
```

b) Use a Boolean mask to access all element whose value is in the interval $[5, 10]$.

```
mask = (x >= 5) & (x <= 10)

print(x[mask])
```

```
[ 6  5  5  5  9  6  7 10]
```

c) Use a Boolean mask to access all element whose value is in the interval $[2, 4]$ or $[6, 9]$.

```
mask = ((x >= 2) & (x <= 4)) | ((x >= 6) & (x <= 9))
print(x[mask])
```

```
[3 6 4 4 2 9 6 7]
```

Question 6

Take the array $x = [0, 1, 2, 3]$ and convert it to

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{bmatrix}$$

```
x = np.arange(0,4)

y = np.tile(x,(2,2))
print(y)
```

```
[[0 1 2 3 0 1 2 3]
 [0 1 2 3 0 1 2 3]]
```

Question 7

Construct the following matrix M using appropriate NumPy functions starting from the array $[1, 2, 4]$:

$$\begin{bmatrix} 1 & 1 & 2 & 2 & 4 & 4 \\ 1 & 1 & 2 & 2 & 4 & 4 \end{bmatrix}$$

```
x = np.array([1,2,4])

x_repeat = np.repeat(x,2)

M = np.tile(x_repeat,(2,1))
print(M)
```

```
[[1 1 2 2 4 4]
 [1 1 2 2 4 4]]
```

Question 8

Write a function `blocks(m,n)` that, for given inputs n and m, returns an $(m+n) \times (m+n)$ matrix that contains an $m \times m$ block of ones on the top left, and an $n \times n$ block of ones on the bottom right (and zeros elsewhere). Use `hstack()` and `vstack()` in your solution.

For $m = 2$ and $n = 3$, this results in the matrix

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

```
def blocks(m,n):
    X1 = np.ones((m,m),dtype='int')
    X2 = np.ones((n,n),dtype='int')
    Y1 = np.zeros((m,n),dtype='int')
    Y2 = np.zeros((n,m),dtype='int')
    return np.vstack((np.hstack((X1,Y1)),np.hstack((Y2,X2))))

#Testing
print(blocks(2,3))
```

```
[[1 1 0 0 0]
 [1 1 0 0 0]
 [0 0 1 1 1]
 [0 0 1 1 1]
 [0 0 1 1 1]]
```

Question 9

Write a function `checkerboard(n)` that returns a checkerboard pattern of zeros and ones of size $n \times n$ (see exmples below; the top-left element is always a 1).

For $n = 5$ and $n = 6$, the matrix should look like this

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

```
def checkerboard(n):
    x = np.zeros((n,n),dtype=int)
    x[0::2,0::2] = 1
    x[1::2,1::2] = 1
    return x

#Testing
n = 5
print(checkerboard(n))
```

```
n = 6
print(checkerboard(n))
```

```
[[1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 1]]
[[1 0 1 0 1 0]
 [0 1 0 1 0 1]
 [1 0 1 0 1 0]
 [0 1 0 1 0 1]
 [1 0 1 0 1 0]
 [0 1 0 1 0 1]]
```

Question 10

Compute the anti-diagonal of the matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & -7 \\ -9 & 10 & 11 \end{bmatrix}$$

using `rot90()`. The anti-diagnoal is obtained by going from the bottom-left to the top-right element, i.e., $[-9, 6, 3]$ in this case (and not $[3, 6, -9]$). Have a look at the documentation of `rot90()`for details of how this function works.

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

# We first rotate the matrix 3*90 = 270 degrees
# and then take the diagonal.
d = np.diag(np.rot90(A,k=3))
print(d)
```

```
[-9  6  3]
```