Lab Questions: Lab Session 3

Deadline: 06.09.2017 11:59pm SGT

Complete all assignments below. Create a script file <yourMatricNo_Lab3>.py and use this script to save your answers (write the commands, not what the console displays after executing your commands) to those questions below that are marked with an asterisk *. These questions are 6 and 10. Once you are done with it, submit the file via iNTU.

Important!!! Please name the files according to the requirements, and upload each file separately and not in a Zip file or similar. The submission system closes at the deadline. Hence after that, you will get no marks for your solution.

1. Using the `range` function, create the following lists:

    (a) [3, 4, 5, 6]

    Solution:
    ```
    >>> list(range(3,7))
    ```

    (b) [18, 16, 14, 12]

    Solution:
    ```
    >>> list(range(18,10,-2))
    ```

    (c) [50, 45, 40, 35, 30, 5, 10, 15, 20, 25, 30]

    Solution:
    ```
    >>> list(range(50,25,-5)) + list(range(5,35,5))
    ```

2. Using the `linspace` function from the NumPy module (consider it is already imported as `np`), create the following vectors:

    (a) [4, 6, 8]

    Solution:
    ```
    >>> np.linspace(4,8,3)
    ```

    (b) [-3, -6, -9, -12, -15]

    Solution:
    ```
    >>> np.linspace(-3,-15,5)
    ```

    (c) [9, 7, 5]

    Solution:
    ```
    >>> np.linspace(9,5,3)
    ```

3. Create the following vectors twice, using `linspace` from the NumPy module (consider it is already imported as `np`) and using the `range` function:

(a) [1, 2, 3, 4, 5, 5, 4, 3, 2, 1]

Solution:
```
>>> np.concatenate([np.linspace(1,5,5),np.linspace(5,1,5)])
>>> np.array(list(range(1,6)) + list(range(5,0,-1)))
```

(b) [4, 5, 4, 5, 4, 5, 4, 5, 4, 5]

Solution:
```
>>> np.tile(np.linspace(4,5,2),5)
>>> np.array(list(range(4,6))*5)
```

4. Create a variable `myEnd` which stores a random integer in the range from 40 to 60 included. Then, using the `range` function, create a list `myList` filled with increasing integers (with a step of 3), starting from 1 and not exceeding `myend`.

Solution:
```
>>> myEnd = random.randint(40,60)
>>> myList = list(range(1,myEnd,3))
```

5. Write an expression that refers to only the odd-numbered elements in a list (starting the counting from 0), regardless of the length of the list in other words, we would like to get elements located at index [1357 . . .]. Test your expression on lists that have both an odd and even number of elements.

Solution:
```
>>> myList = list(range(random.randint(1,50))) # generate an integer list with at most
50 elements
>>> myList[1::2]
```

6. * Assume you are given two lists `myList1` and `myList2`, both of unknown sizes.

(a) Using only methods from lists, remove the last element from `myList1`, remove the first element from `myList2`, concatenante these two new lists while adding a string element `'Hello'` in between.

(b) Same question, without using any list method.

7. Consider that you have access to an array `myArray` that can have any length.

(a) Write assignment statements that would store the first half of the array in one array variable `fhalf` and the second half in another array variable `shalf`. Make sure that your assignment statements are general, and work whether `myArray` has an even or odd number of elements (hint: use a rounding function).

Solution:
```
>>> myArray = np.array(list(range(random.randint(1,50)))) # generate an array filled
```

with integers with at most 50 elements
```
>>> fhalf = myArray[0:round(len(myArray)/2)]
>>> shalf = myArray[round(len(myArray)/2)+1:]
```

(b) Assign the first element of `fhalf` to 100. What happens to `myArray` ? How would you answer to question (a) if you wanted to avoid this effect ?

Solution:
```
>>> fhalf[0] = 100
>>> print(myArray)
```
One can observe by printing `myArray` that its first element has been changed to 100 as well. This is due to the fact that slicing of an array does not create a new object (in contrary to lists): `fhalf` and `shalf` refer to actual part of `myArray`. Thus, modifying `fhalf` or `shalf` will modify `myArray` as well. To avoid this effect, one needs instead to do copies of the data stored in `myArray`:
```
>>> fhalf = myArray[0:round(len(myArray)/2)].copy()
>>> shalf = myArray[round(len(myArray)/2)+1:].copy()
```

8. Create two variables `rows` and `cols` that are two random integers in the range from 1 to 5 included. Create a matrix (two dimension array) of all zeros with the dimensions given by the values of `rows` and `cols`.

Solution:
```
>>> rows = random.randint(1,5)
>>> cols = random.randint(1,5)
>>> np.zeros([rows,cols])
```

9. Create a 5×3 matrix `myMat` (a two dimension array) filled of random values in [0,1). Then, replace the second row of `myMat` with zeros using a single command. Finally, again using a single command, apply the function $f(x) = 2^x + 1$ to all elements of the matrix.

Solution:
```
>>> myMat = np.random.rand(5,3)
>>> myMat[1,:]  = np.zeros([1,3])
>>> 2**myMat+1
```

10. * Assume you are given a variable `myMat` that contains a square matrix of an unknown dimension. Create a new square matrix `myRandMat` that is of the same dimension as `myMat`, but filled with random values in [0,1). Compute the square matrix `myNewMat` defined by:

$$\texttt{myNewMat} = (2 \cdot \texttt{myMat} + (\texttt{myRandMat})^2)$$

11. Create a three-dimensional matrix `mat3d` consisting of all ones and get its dimension.

Solution:
```
>>> mat3d = np.ones([3,5,2])
>>> np.shape(mat3d)
```

12. Create the following vector C:

$$C = [0.7\ 1.9\ 3.1\ 4.3\ 5.5\ 6.7\ 7.9\ 9.1\ 10.3\ 11.5\ 12.7\ 13.9\ 15.1\ 16.3\ 17.5]$$

Then use `reshape` function from the NumPy module to create the following matrix D from the vector C (search for information on the reshape function by typing the command `help(np.reshape)` for example):

$$D = \begin{pmatrix} 0.7 & 1.9 & 3.1 & 4.3 & 5.5 \\ 6.7 & 7.9 & 9.1 & 10.3 & 11.5 \\ 12.7 & 13.9 & 15.1 & 16.3 & 17.5 \end{pmatrix}$$