Lab Questions: Lab Session 6

Deadline: 27.09.2017 11:59pm SGT

Complete all assignments below. For those questions that are marked with an asterisk *, i.e. **Questions 7 and 11**, create the script files as requested. Once you are done with it, submit the file(s) via `iNTU`. Remember to put plenty of comments for all assignments, as this helps us to better understand your program (which might give you higher marks).

Important!!! Make sure your scripts work properly, as we give 0 marks otherwise. Please name the scripts 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. Write a `for` loop that prints all characters of a string `my_str` vertically. Do two versions: one that will iterate through the elements of `my_str` themselves, and one that will iterate through the indexes of `my_str`.

   Solution:

   exercice1.py

```
my_str = 'Hello !'

# iterate through the elements of my_str directly
for i in my_str:
    print(i)

# iterate through the indexes of my_str
for i in range(len(my_str)):
    print(my_str[i])
```

2. Write a script `prodby2.py` that generates a random positive integer $n \in [1, 100]$ and will calculate and print the product of the odd integers from 1 to n (or from 1 to *n-1* if n is even).

   Solution:

   prodby2.py

```
# Generate a random positive integer n in [1,100]
# Calculates and prints 1*3*5*..*n
import random

n = random.randint(1,100)

# initialize the factor to 1 and loop through all odd integers until n
out = 1
for i in range(1,n+1,2):
    out = out * i

print('For n=%d out=%d' % (n,out))
```

3. Write a script `belong.py` that will pick a list L of 10 random integers $\in [1, 100]$. Then, using a `for` loop, iterate through the integers from 32 to 64 included and display the current integer if it belongs to the list L.

Solution:

belong.py

```
import random

# this block of instruction can be replaced by
# L = [random.randint(1,100) for i in range(10)] (see next lecture)

# alternatively, using Numpy, we could have used
# np.random.randint(1,10, size=10).tolist()

L=[0]*10  # preinitialise the list to 10 elements
for i in range(10):
    L[i] = random.randint(1,100)

print(L)

# for all integer in [32,64], check if it belongs to L
for i in range(32,65):
    if i in L:
        print(i)
```

4. Create a Numpy array of 5 random integers, each in the range from -10 to 10 included. Perform each of the following tasks with two versions: using loop(s) (with `if` statements if necessary) and without loop.

Solution:

```
import numpy as np
my_array = np.random.randint(-10,11, size=5)
```

(a) subtract 3 from each element of the array

Solution:

```
# with a loop
for i in range(len(my_array)):
    my_array[i] -= 3


# without a loop
my_array -= 3
```

(b) count how many elements of the array are positive

Solution:

```
# with a loop
```

```
counter = 0
for i in range(len(my_array)):
    if my_array[i] > 0:
        counter += 1


# without a loop
# my_array>0 will replace positive elements by True, and the others by False
# since True is counted as 1 and False is counted as 0,
# we can sum the elements to count the number of positive numbers
counter = sum(my_array>0)
```

(c) replace each elements of the array by its absolute value

Solution:

```
# with a loop
for i in range(len(my_array)):
    my_array[i]=abs(my_array[i])

# without a loop
my_array=abs(my_array)
```

(d) find the maximum value of the array

Solution:

```
# with a loop
my_max = my_array[0]
for i in range(1,len(my_array)):
    if my_array[i] > my_max:
      my_max = my_array[i]


# without a loop
my_max = max(my_array)
```

5. Assume that a Numpy matrix `my_mat` is already defined. Write a script `matrixAverage.py` that will calculate and print the overall average of all numbers in the matrix. Use loops, not built-in functions, to calculate the average. The size of the matrix should be assumed to be unknown. What is the build in Numpy built-in function that performs exactly this task ?

Solution:

matrixAverage.py

```
# Calculates the overall average of numbers in a matrix
import numpy as np

# initialise the running sum to 0
my_sum = 0
(r,c) = my_mat.shape

# for all rows  of the matrix
for i in range(r):
```

```
    # for all columns of the matrix
    for j in range(c):
        my_sum += my_mat[i,j] # add the element to the running sum

# divide the running sum by the number of elements to get the average
average = my_sum/(r*c)


# Numpy built-in function
average = np.average(my_mat)
```

6. Write a script `beautyofmath.py` that produces the following output. The script should iterate from 1 to 9 to produce the expressions on the left, perform the specified operation to get the results shown on the right, and print exactly in the format shown here.

```
>> beautyofmath
1 x 8 + 1 = 9
12 x 8 + 2 = 98
123 x 8 + 3 = 987
1234 x 8 + 4 = 9876
12345 x 8 + 5 = 98765
123456 x 8 + 6 = 987654
1234567 x 8 + 7 = 9876543
12345678 x 8 + 8 = 98765432
123456789 x 8 + 9 = 987654321
```

Solution:

beautyofmath.py

```
# Shows the beauty of math!
leftnum = 0
for i in range(1,10):
    leftnum = leftnum * 10 + i
    result = leftnum * 8 + i
    print('%d x 8 + %d = %d' % (leftnum, i, result))
```

7. * Write a script <YourMatricNo>_Lab6_AboveandleftSumMatrix.py that asks the user for a number of rows r and a number of columns c. Then, it generates a $r \times c$ matrix with the following values:

   - the value of each element in the first row is the number of the column
   - the value of each element in the first column is the number of the row
   - the rest of the elements each has a value equal to the sum of the element above it and the element to the left

   Here is an example of such matrix if the user inputs $r = 4$ and $c = 5$:

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 7 & 11 & 16 \\ 3 & 7 & 14 & 25 & 41 \\ 4 & 11 & 25 & 50 & 91 \end{pmatrix}$$

Solution:

AboveandleftSumMatrix.py

```
# Prompt for a matrix size and then generates a matrix s.t.:
# - the values in the first row is the number of the column
# - the values in the first column is the number of the row
# - other values are the sum of the value above and on the left
import numpy as np

nr = int(input('Enter the number of rows: '))
nc = int(input('Enter the number of columns: '))
A = np.zeros([nr,nc])

# for every row of the matrix
for row in range(nr):
    # for every column of the matrix
    for col in range(nc):
        # the values in the first row is the number of the column
        if row==0:
            A[row,col] = col + 1
        # the values in the first column is the number of the row
        elif col==0:
            A[row,col] = row + 1
        # other values are the sum of the value above and on the left
        else:
            A[row,col] = A[row,col-1] + A[row-1,col]

print(A)
```

8. An inefficient sorting. Write a script `sorting.py` that first asks the user to enter a positive integer n (check if the integer entered is positive and if not continue asking the user). Then, it generates a vector of n random integers between 1 and 100 and finally sorts that vector (and prints it after being sorted). More precisely, you can use the following simple and inefficient algorithm to sort the vector (do not use a built-in Python function to sort the vector): check all pairs of consecutive elements of the vector and swap them if they are not sorted; repeat until all pairs of the vector are sorted.

Solution:

sorting.py

```
# Prompt the user for a number of elements n, then generates
# an array of n random intergers in [1,100] and sorts it
import numpy as np

# prompt the user for an integer n and repeat until it is a positive integer
n = int(input('Enter the number of elements (positive integer): '))
while n<=0:
    n = int(input('Enter the number of elements (positive integer): '))

# generate the array of n random integers in [1,100]
V = np.random.randint(1,101,size=n)

# this part sorts the array
# we continue sorting while the array is not sorted
sorted = False # "sorted" will indicate if V is sorted or not
while not sorted:
```

```
    sorted = True
    # check all the pairs of consecutive elements of V
    for i in range(0,n-1):
        # if one pair is not sorted, we switch the elements
        # and indicate that the array was still not sorted
        if V[i] > V[i+1]:
            tmp = V[i]
            V[i] = V[i+1]
            V[i+1] = tmp
            sorted = False

print(V)
```

9. Write a script `findmine.py` that will prompt the user for minimum and maximum integers, and then another integer which is the user's choice in the range from the minimum to the maximum. The script will then generate random integers in the range from the minimum to the maximum, until a match for the user's choice is generated. The script will print how many random integers had to be generated until a match for the user's choice was found. For example, running this script might result in this output:

```
>>> findmine
Please enter your minimum value:  -2

Please enter your maximum value:  3

Now enter your choice in this range:  0

It took 3 tries to generate your number
```

Solution:

findmine.py

```
# Prompts the user for a range of integers and then an integer in
# this range. Generates random integer until user's integer is
# generated, counting how many tries it took.
import random

my_min = int(input('Please enter your minimum value: '))
my_max = int(input('Please enter your maximum value: '))
my_chc = int(input('Now enter your choice in this range: '))

# generate a random integer in the selected range
my_ran = random.randint(my_min, my_max)
counter = 1

# while the user's integer is not reached, repeat
while my_ran != my_chc:
    my_ran = random.randint(my_min, my_max)
    counter += 1

print('\nIt took %d tries to generate your number' % (counter))
```

10. The inverse of the mathematical constant e can be approximated as follows:

$$\frac{1}{e} \approx \left(1 - \frac{1}{n}\right)^n$$

Write a script `approxe.py` that will loop through values of n until the difference between the approximation and the actual value is less than 0.0001 (the actual value of e can be obtained in the `math` module). The script should then print out the built-in value of e^{-1} and the approximation to 4 decimal places, and also print the value of n required for such accuracy.

Solution:

approxe.py

```
# Approximates 1/e as (1-1/n)^n, and determines the value of n
# required for accuracy to 4 decimal places
import math

# computes the actual value
actual = 1 / math.e

# initializes the variables for the while loop
n = 0
difference = 1

# while the approximation is not good enough, continue
while difference >= 0.0001:
    n += 1
    approx = (1 - 1/n)**n
    difference = actual - approx

print('The built-in value of 1/e is %.4f' % (actual))
print('The approximation is %.4f' % (approx))
print('The value of n is %d' % (n))
```

11. * Write a script <YourMatricNo>_Lab6_thresholdMatrix.py that asks the user to input a threshold value `thres` with `thres` $\in (0, 1)$. Then, using `while` and `for` loops, the script should generate a (5×5) matrix filled with random values in $(0, 1)$ and such that the average value for each row is greater or equal than `thres`. When a solution matrix has been found, print it as well as its average for each row, and the number of tries that were required to find it.

Solution:

thresholdMatrix.py

```
# Prompts for a threshold in (0,1) and then generates a random
# 5x5 matrix in (0,1) s.t. each row has average >= threshold
import numpy as np

thres = float(input('Enter the threshold (between 0 and 1): '))
while thres>=1 or thres<=0:
    thres = float(input('Enter the threshold (between 0 and 1): '))

# initializes the counter and the matrix at 0
average = np.zeros(5)
```

```
counter = 0

# while no proper matrix has been found, continue
while np.min(average) < thres:
    # updates the counter and generates a new random matrix
    counter += 1
    mat = np.random.random(size=(5,5))

    #compute the average for every row
    for row in range(5):
        average[row] = 0
        for col in range(5):
            average[row] += mat[row,col]
        average[row] = average[row]/5

# print the matrix and the averages
print('The matrix found is:')
print(mat)
print('The row averages are:')
print(average)
print('Tries required to find the matrix: %d' % (counter))
```

12. A blizzard is a massive snowstorm. Definitions vary, but for our purposes we will assume that a blizzard is characterized by both winds of 30 mph or higher and blowing snow that leads to visibility of 0.5 miles or less, sustained for at least four hours. Data from a storm one day has been stored in a file `stormtrack.dat`. There are 24 lines in the file, one for each hour of the day. Each line in the file has the wind speed and visibility at a location, e.g.
29.2 1.1
15.1 1.5
...
29.4 0.78
Create a sample data file `stormtrack.txt` or download a example version on iNTU, and place it in your working folder. Create a script `stormtrack.py` that reads this data from the file (you can do so by typing the command `my_array = np.loadtxt('stormtrack.txt')` that will put the data in the Numpy array `my_array`) and determines whether blizzard conditions were met during this day or not.

Solution:

stormtrack.py

```
# Reads wind and visibility data hourly from a file and
# determines whether or not blizzard conditions were met
import numpy as np

# loads the data
my_array = np.loadtxt('stormtrack.txt')
winds = my_array[:,0]
visibs = my_array[:,1]
length = len(winds)

# initializes the counters
count = 0
i = 0

# Loop until blizzard condition found or all data has been read
while count<4 and i<length:
```

```
    if winds[i]>=30 and visibs[i]<=0.5:
        count += 1
    else:
        count = 0
    i += 1

# checks if the blizzard condition was met
if count == 4:
    print('Blizzard conditions met')
else:
    print('No blizzard this time!')
```