Tutorial (week 2)

The following exercises are taken from last semester's final exam from course Algorithm and Computing I.

1. (a) Assume that you are given a variable x that refers to an object of type float. Write a selection statement that prints an error message if x is not a positive integer.

   Solution:
   ```
   if x <= 0 or int(x)!=x:  print('ERROR')
   ```

   (b) What is the result of the following logical expression ?

   ```
   5 > 6 or 8 > 4 and True
   ```

   Solution:
   True

   (c) What is printed on the screen when you execute the following commands.

   ```
   Sa = 'abcdefghij'
   Sb = Sa[:4]*2 + Sa[6:]*2 + Sa[5]
   print(Sb)
   ```

   Solution:
   'abcdabcdghijghijf'

   (d) What is the average asymptotic complexity of these two algorithms, relative to n ? Which one is asymptotically faster ?

   Algorithm 1:
   ```
   my_value = 1
   for i in range(n):
       for j in range(n):
           my_value += j
       my_value *= 2
   ```

   Algorithm 2:
   ```
   my_value = 1
   for i in range(n):
       my_value *= 2
   for j in range(n):
       my_value += j
   ```

   Solution:
   The first algorithm has two nested *for* loops, each ranging through n elements. Thus, its asymptotic complexity is $O(n^2)$. The second algorithm has two independent *for* loops, each ranging through n elements. Thus, its asymptotic complexity is $O(n)$.

   (e) Consider the following piece of code:

   ```
   for i in range(n):
       for j in range(i,n):
           print('*',end='')
       print('\n',end='')
   ```

Write what is printed on the screen when we execute these lines, assuming that $n = 4$.

Solution:

```
****
***
**
*
```

(f) Sort these five operators according the order precedence rule (from highest to lowest priority):

`*`   ()   `**`   `and`   +

Solution:

()   `**`   `*`   +   `and`

2. (i) We have a function $F(x, y)$ defined as $F(x, y) = y$ for any $y \neq 0$, and as

$$F(x,0) = \begin{cases} 4x^5 + x - 2 & \text{when } x < -3 \\ -x & \text{when } x \geq -3 \text{ and } x \leq 2 \\ x/2 & \text{when } x > 2 \end{cases}$$

when $y = 0$.

Implement a Python function `computeF` that takes as input the values x and y, and that returns the corresponding value $F(x, y)$.

Solution:

```python
def computeF(x,y):
    if y!=0:
        return y
    else:
        if x<-3:
            return 4*x**5+x-2
        elif x<=2:
            return -x
        else:
            return x/2
```

(ii) Assuming that you have access to the function `computeF` from the previous question, implement a user menu that will keep asking the user to input some value x and that will print every time the result of the function $F(x, 0)$ (no need for error checking). The process stops only when the user enters the string "exit". Besides, when the user enters the string "hi", the script should print on the screen the string "hello" (without exiting the program).

Here is a possible execution example of the program:

```
Please enter a value x: 10
F(10.000000,0) = 5.000000

Please enter a value x: 1
F(1.000000,0) = -1.000000
```

```
Please enter a value x: hi
hello

Please enter a value x: exit
exiting the program ...
```

Solution:

```python
while True:
    user_input = input('Please enter a value x: ')
    if user_input=='hi':
        print('hello')
    elif user_input=='exit':
        print('exiting the program ...')
        break
    else:
        x = float(user_input)
        print('F(%f,0) = %f' % (x,computeF(x,0)))
```

3. The Pascal triangle is a special triangular array named after the French mathematician Blaise Pascal. It can be built with this simple recursive method: in the first row, only the number 1 appears. Then, in order to deduce the elements of the following rows, just add the number above and to the left with the number above and to the right to find the new value. If either the number to the right or left is not present, substitute a zero in its place. For example, here is the triangle for the first 6 rows:

$$\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1
\end{array}$$

Write a Python recursive function `pascal` that will take as input an integer n and that will generate a list representing the n-th row of the Pascal triangle. For example, `pascal(5)` should return the list $[1, 4, 6, 4, 1]$.

Solution:

```python
def pascal(n):
    if n == 1:
        return [1]
    else:
        current_line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            current_line.append(previous_line[i] + previous_line[i+1])
        current_line.append(1)
    return current_line
```

4. For this question, you can assume that the NumPy module has already been imported using `import numpy as np`. Besides, you are not allowed to use built-in Python or NumPy functions such as `sum` or `np.average`.

   (i) Write a Python function `my_average` that will take as input a matrix `mat` of unknown size, represented as a NumPy two-dimensional array. The function will output the average of the elements of that matrix.

   Solution:

```python
def my_average(mat):
    (rows,cols) = mat.shape
    my_sum = 0
    for r in range(rows):
        for c in range(cols):
            my_sum += mat[r,c]

    return my_sum/(rows*cols)
```

   (ii) Write a Python function `is_average` that will take as input a matrix `mat` of unknown size, represented as a NumPy two-dimensional array. The function will output the boolean value *True* if any of the matrix elements is equal to the matrix average, *False* otherwise. You can assume that you have access to `my_average`, the function implemented in the previous question.

   Solution:

```python
def is_average(mat):
    (rows,cols) = mat.shape
    mat_average = my_average(mat)
    for r in range(rows):
        for c in range(cols):
            if mat[r,c]==mat_average:
                return True

    return False
```

   (iii) Write a function `sort_average` that will take as input a matrix `mat` of unknown size, represented as a NumPy two-dimensional array. The function will output a list containing all the elements of the matrix, sorted (in increasing order) according to their distance to the matrix average. For an element x and an average value a, the distance is defined as $|x - a|$. You can assume that you have access to `my_average`, the function implemented in the previous question.
   For example, with the matrix

$$\begin{bmatrix} 0 & 2 & 0 \\ 2 & 4 & 1 \\ 1 & 5 & 3 \end{bmatrix}$$

   the matrix average is 2, and the output of the function `sort_average` should be $[2, 2, 3, 1, 1, 4, 0, 0, 5]$ (some elements could be at a different position as they have the same distance to the average, for example 1's and 3's positions could be permuted).

Solution:

```
def sort_average(mat):
    (rows,cols) = mat.shape
    mat_average = my_average(mat)

    my_input_list = []
    for r in range(rows):
        for c in range(cols):
            my_input_list.append(mat[r,c])

    my_output_list = [my_input_list[0]]
    for j in range(1,len(my_input_list)):
        i=0
        while abs(my_output_list[i]-mat_average) < abs(my_input_list[j]-
            mat_average):
            i += 1
            if i==len(my_output_list):
                break
        my_output_list.insert(i,my_input_list[j])

    return my_output_list
```