ELEC0021 - Programming
ASSIGNMENT 2

This assignment is worth 20% of the Programming 2 mark. You must work **individually** implementing 3 different algorithms (sorting, basic image processing and basic data processing). Remember that you can discuss ideas with your classmates, but **you must not share your code with others**.

The deadline for this coursework is **Monday 20st March 2023, 9am**. Oral examinations of your submitted code will take place on the last week of the term (from 20th to 24th March).

PART A: Descending Hybrid Sort (30 points, sorting algorithms)

Hybrid Sort is an algorithm that mixes the operation principles behind Bubble Sort and Selection Sort to speed up the sorting task.

In the descending version (i.e. sorting from largest to smallest) of Bubble and Selection sort, at the end of the first iteration of the outer loop of Bubble Sort the smallest value is stored in the last position of the array whilst at the end of the first iteration of the outer loop of Selection Sort (when sorting from largest to smallest) the largest element is stored in the first position of the array.

Hybrid sort will mix both sorting algorithms in such a way that **at the end of the first iteration** of the outer loop the largest element will be found in the first position of the array (due to Selection Sort) and the smallest element will be stored in the last position of the array (due to Bubble Sort). In the **second iteration** (of the outer loop), the second largest element will be stored in the second position of the array and (due to Selection Sort) and the second smallest element will be stored in the penultimate position of the array (due to Bubble Sort), and so on.

To achieve the behaviour described above, Hybrid Sorts uses the **code of Bubble Sort as the baseline code** and includes a part of Selection Sort in it as follows: whilst Bubble Sort compares pairs of values in the array to decide whether to swap them or not (in the inner loop), Hybrid Sort will use **this very same loop** to search for the maximum value in the array and store it in the right position at the end of this inner iteration. Once the maximum value is found, it is swapped with the number stored in the position where the maximum value should be stored (that is the Selection Sort part). Notice that Hybrid Sort keeps the structure of one loop inside another loop (avoid the implementation that uses two inner loops – one after another – inside the outer loop).

Table 1 shows how the content of an array made of 5 elements ([67, 12, 44, 24, 66]) changes **after each iteration of the outer loop (while loop)** of Hybrid Sort:

Table 1. Example of execution of descHybridSort

[25 55 75 38 5]	**Original array**
[55 25 75 38 5]	End of 1st iteration of inner loop. Highlighted values are analysed. Bubble sort has swapped them. Selection sort has recorded the position of the maximum value between 55 and 25 (position 0).
[55 75 25 38 5]	End of 2nd iteration of inner loop. Highlighted values are analysed. Bubble sort has swapped them. Selection sort has recorded the position of the maximum value between 55 (previous maximum) and 75. New position for maximum is 1.
[55 75 38 25 5]	End of 3rd iteration of inner loop. Highlighted values are analysed. Bubble sort has swapped them. Selection sort has recorded the position of the maximum value between 75 (previous maximum) and 38. No new maximum is found.
[55 75 38 25 5]	End of 4th iteration of inner loop. Highlighted values are analysed. Bubble sort has **not** swapped them. Selection sort has recorded the position of the maximum value between 75 (previous maximum) and 5. No new maximum is found.
[75 55 38 25 5]	Selection sort swaps maximum value (75) with element in position [0].
[75 55 38 25 5]	**Array at the end of 1st iteration of outer loop (while loop).** Notice that first and last position are already sorted, so the inner loop on the next outer iteration does not consider those positions.
[75 55 38 25 5]	End of 1st iteration of inner loop. Highlighted values are analysed. Bubble sort has **not** swapped them. Selection sort has recorded the position of the maximum value between 55 and 38 (position 1).
[75 55 38 25 5]	End of 2nd iteration of inner loop. Highlighted values are analysed. Bubble sort has **not** swapped them. Selection sort has recorded the position of the maximum value between 55 (previous maximum) and 25. No new maximum is found.
[75 55 38 25 5]	Selection sort swaps maximum value (55) with element in position [1].
[75 55 38 25 5]	**Array at the end of 2nd iteration of outer loop (while loop).** No swaps have been performed, algorithm ends.

Your task: Implement the method **descHybridSort** in the class below. The method descHybridSort implements the descending version of Hybird Sort (from largest to smallest) and **prints the content of the array immediately after the end of the outer loop**, as shown in the example of Figure 1.

```
import numpy as np
class sortArray:

  def __init__(self,size):
    self.size=2
    self.setSize(size)
    self.numbers=np.random.random([self.size])*100
    self.numbers=self.numbers.round(0)

  def __str__(self):
    return "Array of "+str(self.size)+" numbers"+"\n"+str(self.numbers)

  def setSize(self,size):
    if(size>=2):
      self.size=size

  def descHybridSort(self):
    #your code goes here
```

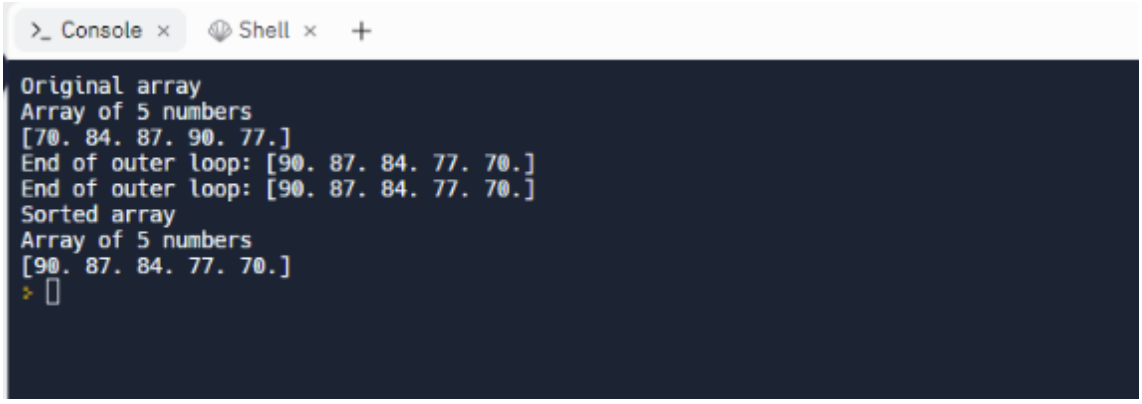

Figure 1. Example of data printed on screen when executing descHybridSort.

The information shown in Figure 1 was obtained by running the following code:

```
print("Original array")
A=sortArray(5)
print(A)
A.descHybridSort()
print("Sorted array")
print(A)
```

You must submit the file sortArray.py above, including your implementation of **descHybridSort**. You must not modify the code given in the class, except by adding your own code in the method descHybridSort.

Suggestion: If you feel stuck, you might want to use incremental developing to solve this problem in steps:

- Step 1: Code Bubble Sort in the standard way (from smallest to largest) and make sure you understand how it works. Next, change Bubble Sort to sort numbers from largest to smallest (i.e. largest element in position 0 and smallest element in the last position of the array).
- Step 2: Code Selection Sort (separately) and make sure you understand how it works. Next, change Selection Sort to sort numbers from largest to smallest.
- Step 3: Modify the code of Bubble Sort (inner loop) to find the maximum value in the unsorted part of the array in each iteration. Print this value, to make sure your code is working correctly.
- Step 4: Further modify Bubble Sort to apply the working principle of Selection Sort. That is, swap the maximum value found in Step 3 with the value that is occupying its final correct position. That is, in the first iteration the maximum must go into position 0, in the second it needs to go in position 1 and so on.
- Step 5: Test your code with normal, extreme (e.g. reversed sorted data) and random cases.

Just for fun (this will not be evaluated), you might want to measure the time it takes Bubble Sort and Hybrid Sort to sort a big array. You will realise that, even when both algorithms have the same worst-case computational complexity ($O(N^2)$), their execution times can be very different.
You can learn about measuring execution time here: https://pynative.com/python-get-execution-time-of-program/

PART B: Image Filter (40 points, Numpy and Matplotlib)

Grayscale images can be stored as 2D arrays. Every element in the array is a pixel with a value between 0 (black) and 255 (white). The Python code below uses a 2D array to create 2 images, as follows:

- **Lines 1-3:** Relevant modules are imported. The module matplotlib allows to read and write image files
- **Line 5:** A 2D array, called image, of 300x300 elements storing the number 0 is created.
- **Line 7:** The 2D array image is stored as an image in the file all_black.jpg, thanks to the method imsave available in matplotlib. The image will look like a black square of 300x300 pixels.
- **Lines 9-10:** The number 255 is stored in the diagonal of the array. Thus, we are adding a white diagonal line to the original image.
- **Line 12:** The black square with the white diagonal line is stored in file line.jpg

```
1.  import numpy as np
2.  import random
3.  import matplotlib.pyplot as plt

4.  #creating a black square of 300x300 pixels
5.  image=np.zeros([300,300])

6.  #saving black square to png file
7.  plt.imsave('all_black.jpg', image, cmap=plt.cm.gray)

8.  #drawing a white diagonal
9.  for i in range(0,300):
10.   image[i,i]=255

11. #saving png file
12. plt.imsave('line.jpg', image, cmap=plt.cm.gray)
```

You can execute this code and open the generated files to check their content.

Image filters are nothing more than operations performed in the data of the array storing the image. For example, the following code acts as a filter that mixes 3 images:

```
import numpy as np
import random
import matplotlib.pyplot as plt

#creating a black square with a white diagonal
image1=np.zeros([300,300])
```

```
for i in range(0,300):
  image1[i,i]=255
plt.imsave('diag.png', image1, cmap=plt.cm.gray)

#creating a black square with a smaller square in it
image2=np.zeros([300,300])
for i in range(100,200):
  for j in range(100,200):
    image2[i,j]=255
plt.imsave('square.png', image2, cmap=plt.cm.gray)

#creating a black square with a "reverse" white diagonal
image3=np.zeros([300,300])
for i in range(0,300):
  image3[i,299-i]=255
plt.imsave('rev_diag.png', image3, cmap=plt.cm.gray)

#extracting 1/3 of each image and adding it in image 4
image4=np.zeros([300,300])
for j in range(0,100):
  image4[:,j]=image1[:,j]
for j in range(100,200):
  image4[:,j]=image2[:,j]
for j in range(200,300):
  image4[:,j]=image3[:,j]
plt.imsave('merge.png', image4, cmap=plt.cm.gray)
```

You can execute the code above and open the generated files to check their content.

Your task: Implement the methods removeNoise() and invert() in the class noisyPattern shown below:

```
import numpy as np
import random
import matplotlib.pyplot as plt

class noisyPattern:

  def __init__(self):
    self.image=np.full([80,80],255)
    self.setFigure()
```

```
    def setFigure(self):
      for i in range(20,30):
        self.image[i,20:30]=0
        self.image[i, 49:59]=0
      for i in range(40,60):
        self.image[i,i:i+10]=0
        self.image[i,79-i-10:79-i]=0
      plt.imsave('figure.png', self.image, cmap=plt.cm.gray)

  def addNoise(self):
    noise=0.05
    for i in range (0,self.image.shape[0]):
      for j in range(0,self.image.shape[1]):
        if(noise > random.random()):
          self.image[i,j]=255
    plt.imsave('noisy_pattern.png', self.image, cmap=plt.cm.gray)

  def removeNoise(self):
    #your code here
    #if there is a white dot, replace its value by the value
    # of the majority of neighbouring pixels
    #save the resulting image in file noise_removed.png

  def invert(self):
     #your code here
     #produce an upside-down version of the image
```

Below, you can see the contents of the files (you will need to zoom in to be able to see the details):

- figure.png (Figure B1.a), created by the method setFigure()
- noisy_pattern.png (Figure B1.b), created by the method addNoise() for noise=0.05
- noise_removed.png (Figure B1.c), the noisy pattern after being processed to reduce noise by the method removeNoise()
- inverted.png (Figure B.1d), the original image (figure.png) after applying the method invert(). The image is now upside down.

a) figure.png b)noisy_pattern.png c)noise_removed.png d)inverted.png

Figure B1. Images used/generated by the code

The implementation of the methods removeNoise() and invert() must follow the description below:

- removeNoise: every white pixel (pixel containing value 255) that is surrounded by a majority of black pixels is replaced by 0. For example, in Figure B.2, the majority of the pixels surrounding the pixel being analysed (cell highlighted in light blue) are black pixels. Hence, the pixel is changed to black.

0	0	255
0	255	0
0	0	255

→

0	0	255
0	0	0
0	0	255

Figure B.2. Example of noise removal in a black and white noisy image.

If the majority of pixels surrounding a white pixel are also white, no change is needed. To keep the code easy to read, you **must create** an auxiliary private method to determine whether the majority of neighbouring pixels are white or black. You do not need to filter the noise in the borders of the image.

- invert(): a new image where the contents of the first row in the original figure are now the contents of the last row, the contents of the second row are now the contents of the second to last row and so on. Below, you can see how a 4x4 pixel image is inverted.

0	255	255	255
255	0	255	255
255	255	0	255
255	255	255	0

→

255	255	255	0
255	255	0	255
255	0	255	255
0	255	255	255

Figure B.3. Example of method invert applied to an image of 4x4 pixels

Please, notice that you must not modify the code given to you, except for the methods you need to build and the main programme you need to write to test your code.

PART C: Getting started with Data Science (30 points, Strings and Pandas)

Many times, people who need to analyse the data from a csv file are not programmers. For them, a programme that allows them to perform data processing with an easy interface is of paramount importance.

Your task: implement the functions filtering() and plotting() for the data available in the file **athlete_events.csv** available at https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results. The description of the functions is as follows:

- **filtering:** this function takes a dataframe as input argument. Next, it asks the users what features of the dataset will be used as filters, as shown in Figure C.1. Due to the limited time you have for the assignment, this function will only filter the features Sex, Age, Team, Year and Sport.

```
Please, enter the numbers of the filters you would like to
use (e.g. 234 if you want to filter by age, team and year):
  1.Sex
  2.Age
  3.Team
  4.Year
  5.Sport
```

Figure C.1. Initial information presented to the user

The user can then enter their choices. For example, if the user enters number 3, it means that they want to filter using the Team feature only. Instead, if the user enters 145, it means the user wants to filter the data based on the features of Sex, Year and Sport. If the user enters 12345, it means they want to filter the data based on the 5 features.

If the user enters an invalid number (i.e. more than 5 digits or any individual digit higher than 5), the execution stops and a message is displayed, as shown in Figures C.2 and C.3.

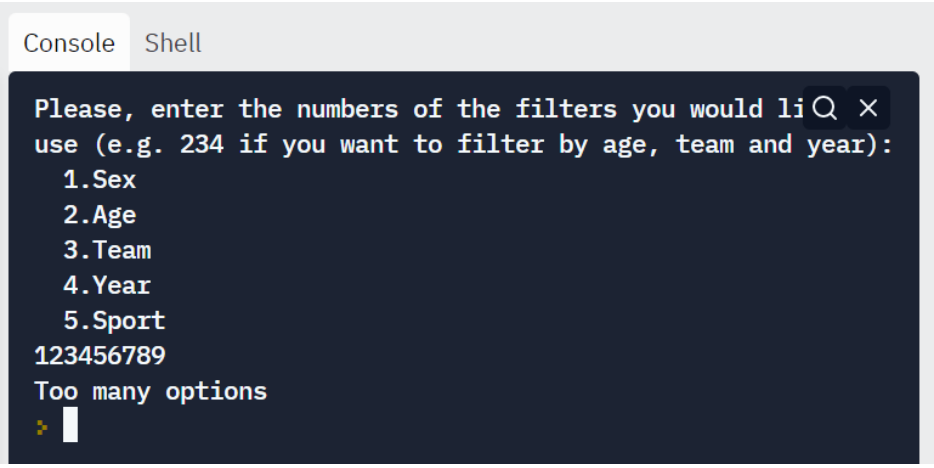

Figure C.2. Programme behaviour when the user enters more than 5 digits

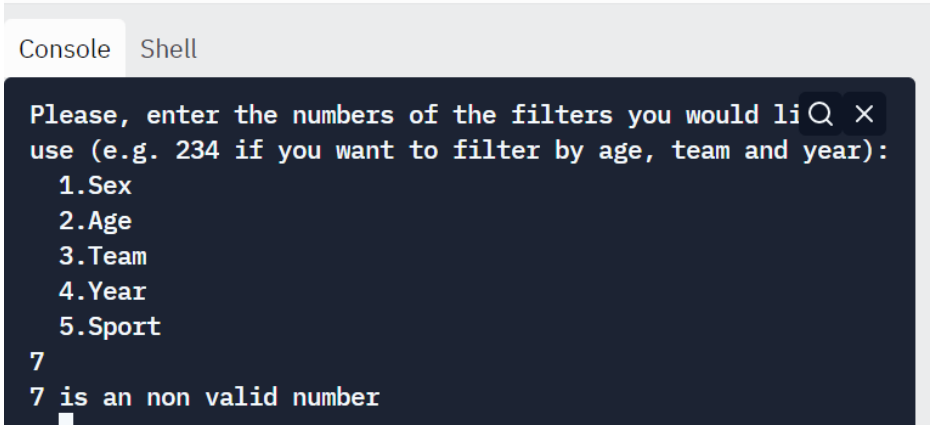

Figure C.3. Example of the programme behaviour when user enters a nonvalid number

If the user enters a valid option, the system asks about the value of the features, as shown in Figure C.4. In this case, the user has entered the number 125, meaning they want to filter using the features of Sex, Age and Sport. Thus, the programme asks for the values the user wants to use for each of those features.

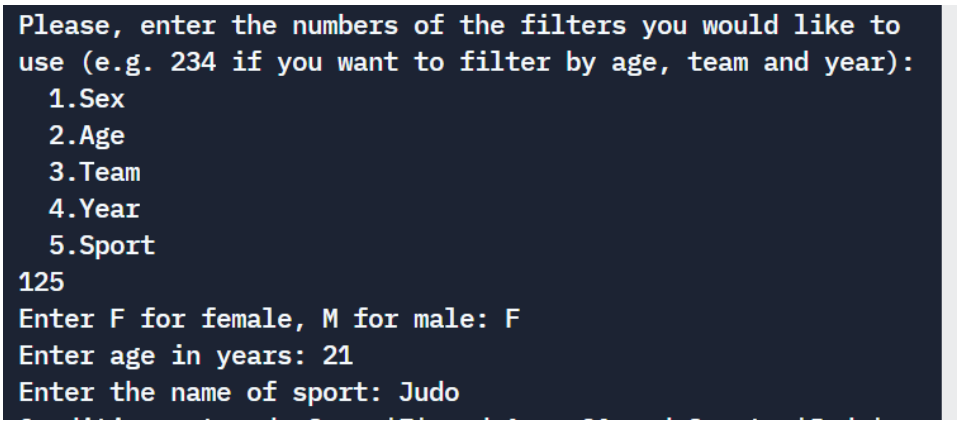

Figure C.4. Requesting additional information to the user

Next, the function must filter the data and store the filtered data in a new dataframe made only of the rows with the athletes data that match the request made by the user. In the example of Figure C.4. that means filtering the rows that comply with Sex being equal to F, Age being equal to 21 and Sport being equal to Judo.

Finally, the function filtering returns **the number of records of the new dataframe and the new dataframe itself** (the one with the filtered data). In the cases of error (exemplified in Figures C.2 and C.3) the function must return **zero (zero numbers of records) and the original dataframe**.

- **plotting:** this function takes two input arguments: the number of records of a dataframe and the dataframe. First, it prints the number of records received. Next, if

the number of rows of the dataframe is lower than 100 (and higher than 0), it generates a scatter plot of the **weight of the athletes** (horizontal axis is the ID of the athletes). This plot must be saved in a file called scatter.png and the message "File scatter.png saved" must be printed on screen. If the number of records is higher than or equal to 100, the function generates a **histogram of the weight using 12 bins**. This plot must be saved in a file called hist.png and the message "File hist.png saved" must be printed on screen. If the number of rows is equal to 0, no plot is generated. Please, see an example of the information printed on screen in Figure C.5.

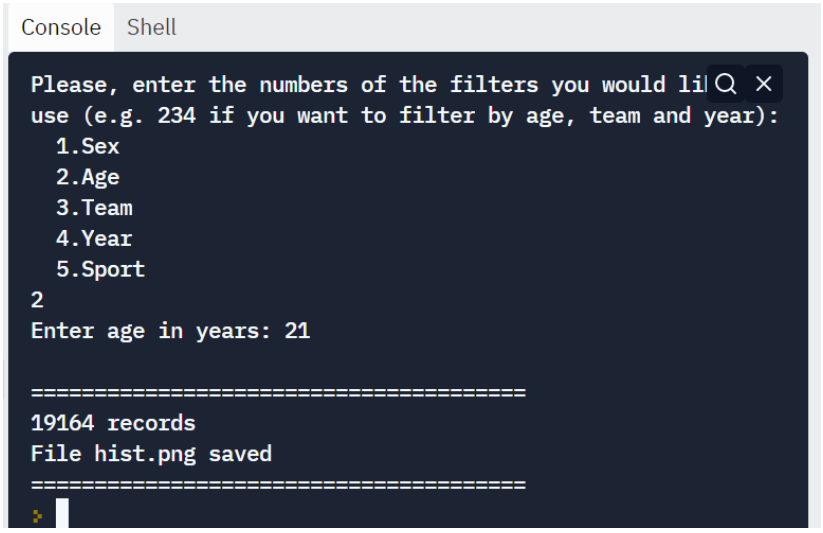

Figure C.5. Example of information printed on screen by the programme under normal operation