



SUPINFO
International University

INSTITUTE OF INFORMATION TECHNOLOGY

Arithmetic and Cryptography

Mini - Project

Jefferson Disk

Version 1.0

Last update: 02/02/2017

Use: Students/Staff

Author: Laurent GODEFROY

CONTENTS

0 INTRODUCTION.....	3
1 GENERALITIES ON JEFFERSON DISK.....	4
2 A SHELL VERSION OF JEFFERSON DISK.....	4
3 A GUI VERSION OF JEFFERSON DISK.....	6
4 BONUS.....	13
5 INDICATIVE SCALE.....	14

1 INTRODUCTION

This examination is to be carried out in groups of two students. In the only case where the number of students per class is odd, one and only one group of three is allowed

Any form of plagiarism or use of codes available on the internet or any other format, even partially, is strictly prohibited and will be penalized with a 0 and will be referenced as "cheater" and will be summoned to a disciplinary committee.

This mini-project will be defended by a viva. Your passage schedule will be communicated to you by your campus.

The viva is also made up of groups of two. This will take **20 minutes** in which you will show to your examiner the proper functioning of your program with a demo. If you have not implemented the entire project, you will expose the functioning parts.

To support your presentation, you must prepare a PowerPoint file in which you will explain the most important and significant points of the code. It will not be necessary to send this file to your examiner, he will discover it the day of the defense. A statement specifying all of this will be sent in February.

Another note, **do not hesitate to use the forum to exchange about this project; I opened a thread for this purpose.**

If desired, you can add functions and procedures to those requested.

2 GENERALITIES ON JEFFERSON DISK

Important note: no code is requested in this part which is only explanatory.

The goal of this mini-project is to implement in Python a program simulating the famous **Jefferson Disk**.

Remind that this device incorporates many disks, 36 for the original one, rotating around an axis and where are registered unordered alphabets. The two correspondents have the same disks.

Each disk is identified by a number. The key is the order where the disks are inserted on the axis, it is therefore a sequence of numbers.

To encrypt a message, the sender arranges the disks according to the key, then makes them rotating in such way that the message appears on the same line of the cylinder. The encrypt message will be the content of the next sixth line (This last choice is only conventional, we could have done another).

To decipher a message, the receiver arranges also the disks according to the key, then makes them rotating in such way that the encrypt message appears on the same line of the cylinder. The original message will be the content of the previous sixth line.

See https://en.wikipedia.org/wiki/Jefferson_disk for complements and a photo of the original cylinder.

3 A SHELL VERSION OF JEFFERSON DISK

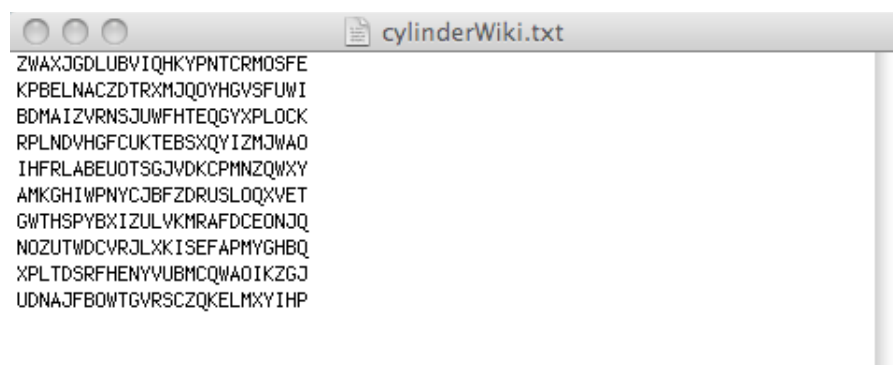
It is strongly recommended to read all the content of this part before coding. The requested work is noticeable with its blue colour.

In a file named « JeffersonShell.py », implement the following sub-routines:

- A « convertLetters(text) » function where **text** is a string. This function will delete all the spaces, punctuation marks and accented letters before returning the string.

- A « mix() » function which return a string composed by 26 characters of the alphabet in uppercase in a random order.
- A « createCylinder(file,n) » procedure where **file** is a name of text file and **n** a strictly positive integer. It will open the file in write mode, then will write **n** lines, each line is generated by the previous function.
- A « loadCylinder(file) » function where **file** is the name of a text file which each lines contains a permutation of the 26 letters of the alphabet in uppercase. This function will return a dictionary which the keys are integers between **1** and the number of lines of the file, the corresponding value of a **i** key is the line number **i** of the file.

Example: with the file



```
cylinderWiki.txt
ZWAXJGDLUBVIQHKYPNTCRMOSFE
KPBELNACZDTRXMJQOYHGVSFUWI
BDMAIZVRNSJUWFHTEQGYXPLOCK
RPLNDVHGFCUKTEBSXQYIZMJWAO
IHFRLABEUOTSGJVDKCPMNZQWXY
AMKGHIWPNYCJBFZDRUSLOQXVET
GWITHSPYBXIZULVKMRAFDCEONJQ
NOZUTWDCVRJLXKISEFAPMYGHBQ
XPLTDSRFHENYVUBMCQWAOIKZGJ
UDNAJFBOWTGVRSCZQKELMXYIHP
```

this function will return this dictionary:

```
{1: 'ZWAXJGDLUBVIQHKYPNTCRMOSFE', 2:
'KPBELNACZDTRXMJQOYHGVSFUWI', 3:
'BDMAIZVRNSJUWFHTEQGYXPLOCK', 4:
'RPLNDVHGFCUKTEBSXQYIZMJWAO', 5:
'IHFRLABEUOTSGJVDKCPMNZQWXY', 6:
'AMKGHIWPNYCJBFZDRUSLOQXVET', 7:
'GWITHSPYBXIZULVKMRAFDCEONJQ', 8:
'NOZUTWDCVRJLXKISEFAPMYGHBQ', 9:
'XPLTDSRFHENYVUBMCQWAOIKZGJ', 10:
'UDNAJFBOWTGVRSCZQKELMXYIHP'}
```

- A « keyOK(key,n) » function where **key** is a list of integers and **n** is a strictly positive integer. It will return **True** if the list is a permutation of all the integers (at large) between **1** and **n**.
- A « createKey(n) » function where **n** is a strictly positive integer. It will return a list constituted by the integers (at large) between **1** and **n** in a random order.
- A « find(letter,alphabet) » function where **letter** is a letter in uppercase of the alphabet and **alphabet** is a string composed by 26 letters of the alphabet in uppercase in a certain order. It will return the index of the letter in the string. You will code yourself this search algorithm without using a function provided by the language.

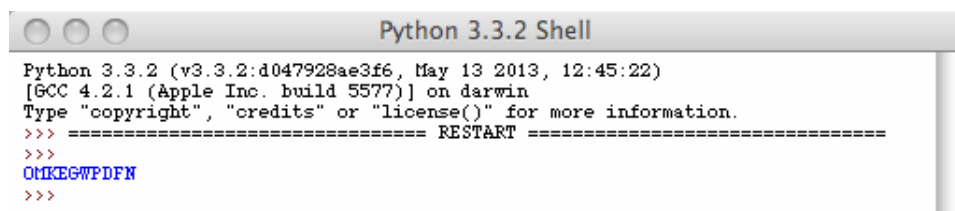
- A « shift(*i*) » function where **i** is an integer between 0 and 25 (at large). It will return **i + 6** with a congruence modulo 26.
- A « cipherLetter(letter,alphabet) » function where **letter** is a letter in uppercase of the alphabet and **alphabet** is a string composed by 26 letters of the alphabet in uppercase in a certain order. It will return the letter located 6 positions after the letter passed in parameter in the string also passed in parameter. It is assumed, of course, that the alphabet in question is circular.
- A « cipherText(cylinder,key,text) » function where **cylinder** is a dictionary which the keys are the first integers from 1 and the values are strings composed by 26 letters of the alphabet in uppercase in a certain order, **key** a list of integers and **text** string. Firstable It will verify if the key is valid according to the cylinder. If it's the case, it will convert the string in letters, then it will return the string encrypt according to the the Jefferson Method.
- See how to use/edit the previous function to decrypt a text.

Example of running (from wikipedia) of some of our subroutines:

- This code :

```
cylinder = loadCylinder("cylinderWiki.txt")
key = [7,9,5,10,1,6,3,8,2,4]
print(jeffersonCipher(cylinder,key,"Retreat Now"))
```

- Give that result :



```
Python 3.3.2 Shell
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 12:45:22)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
OMKEGWPDFN
>>>
```

Use of these procedures and functions :

- Decipher the text 'GRMYSGBOAAMQGDPEYVWLDFDQQQZXXVMSZFS' with the cylinder 'MP-1ARI.txt' and the key [12, 16, 29, 6, 33, 9, 22, 15, 20, 3, 1, 30, 32, 36, 19, 10, 35, 27, 25, 26, 2, 18, 31, 14, 34, 17, 23, 7, 8, 21, 4, 13, 11, 24, 28, 5].
- Encrypt a text of your choice with a cylinder and a key of your choice. Attach them to your project.

Three little questions that await a justified answer:

- What do you think about the security of this algorithm?

- What are its main qualities and downsides?
- How much there is keys in a cylinder of **n** disks?

4 A GUI VERSION OF JEFFERSON DISK

It is strongly recommended to read all the content of this part before coding. In particular, the final example. The requested work is noticeable with its blue colour.

We will necessary use the the graphic library Pygame. Using another library is not allowed.

Note: you are free to design your cylinder as you want while the functionality are available. The screenshots of this part are only here to give you an idea.

In a file named « **JeffersonGUI.py** », implements the followings subroutines:

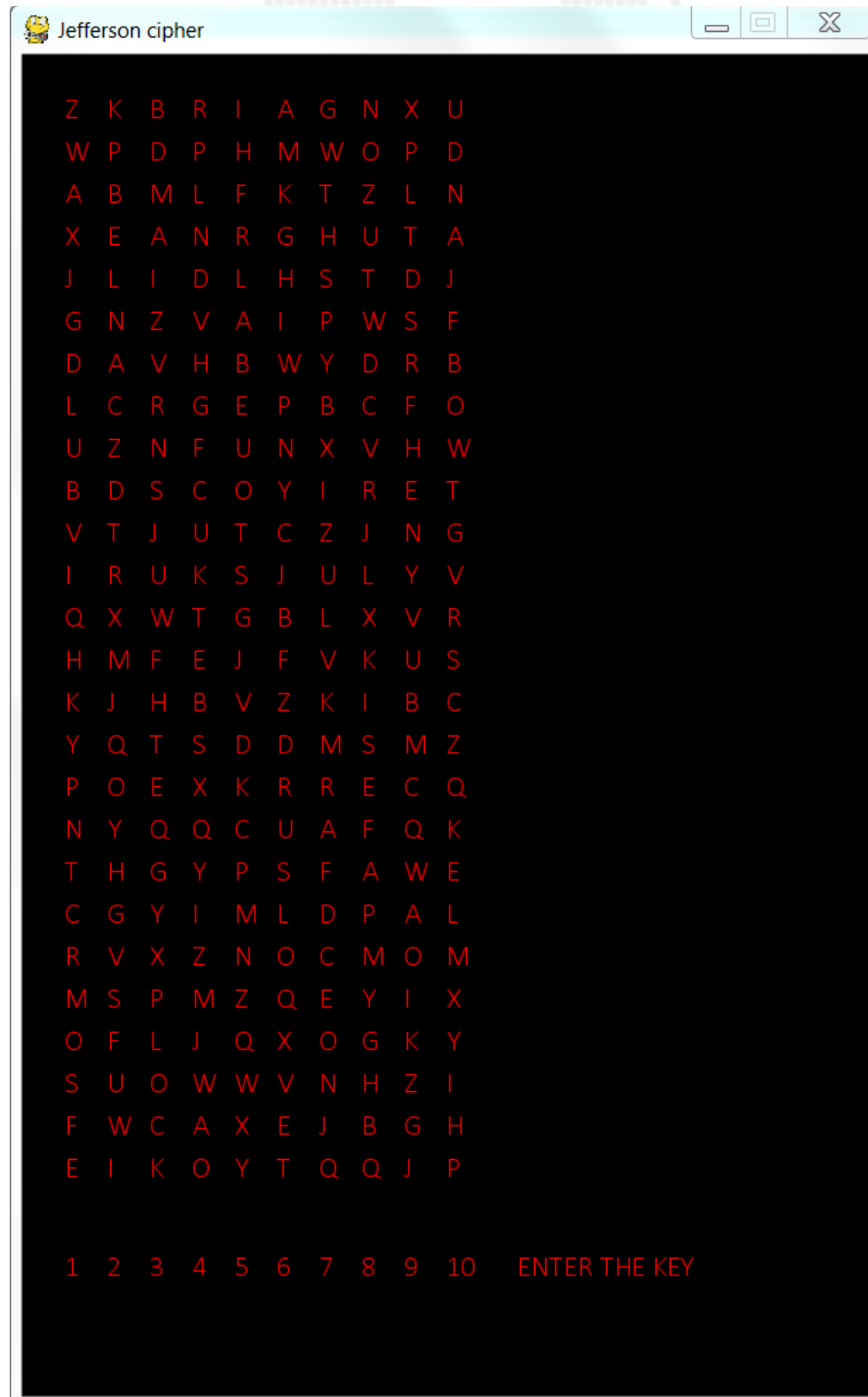
- A « **displayCylinder(mySurface,cylinder,i)** » procedure where **mySurface** is the current surface, **cylinder** is a dictionary representing a cylinder of Jefferson and **i** a number of disks. It will display at the “right place “on the surface the disk number **i** of the cylinder.
- A « **displayCylinders(mySurface,cylinder)** » procedure where **mySurface** is the current surface and **cylinder** a dictionary that represents a cylinder of Jefferson. It will display on the surface all the disks of the cylinder.
- A « **enterKey(mySurface,n)** » function where **mySurface** is the current surface and **n** is the number of disks of the cylinder. It will display under the cylinder the number of each disks. It will wait that the user clicks in the order of his choice on each of these numbers in order to constitute the key. Once a number has been selected, his colour change cannot be selected again. On a second line, it displays gradually the key. Finally, the function will return the key (a key that is a list of numbers).
- A « **rotateCylinder(cylinder,i,up=True)** » procedure where **cylinder** is a dictionary that represents cylinder of Jefferson, **i** a number of disk and **up** a boolean. If **up** is **True** it will do an offset of the letters of the disk number **i** “to the top “and if **up** is **False**, this offset will be “to the bottom “. This procedure ain’t got no single visual effect on the graphic window, the current surface is not passed into parameters. It only edits the dictionary that represents the cylinder.
- A « **rotateCylinders(mySurface,cylinder)** » procedure where **mySurface** is the current surface and **cylinder** a dictionary that represent a cylinder of Jefferson. The user can click on the arrows under each disks in order to

make them spin in one direction or another. Those rotations edit the current surface and the dictionary that represents the cylinder. When the text that the user wants to encrypt appears on the line "CLEAR "he clicks on "FINISH "to stop the process. The encrypted message appears therefore on the line "CIPHER ".

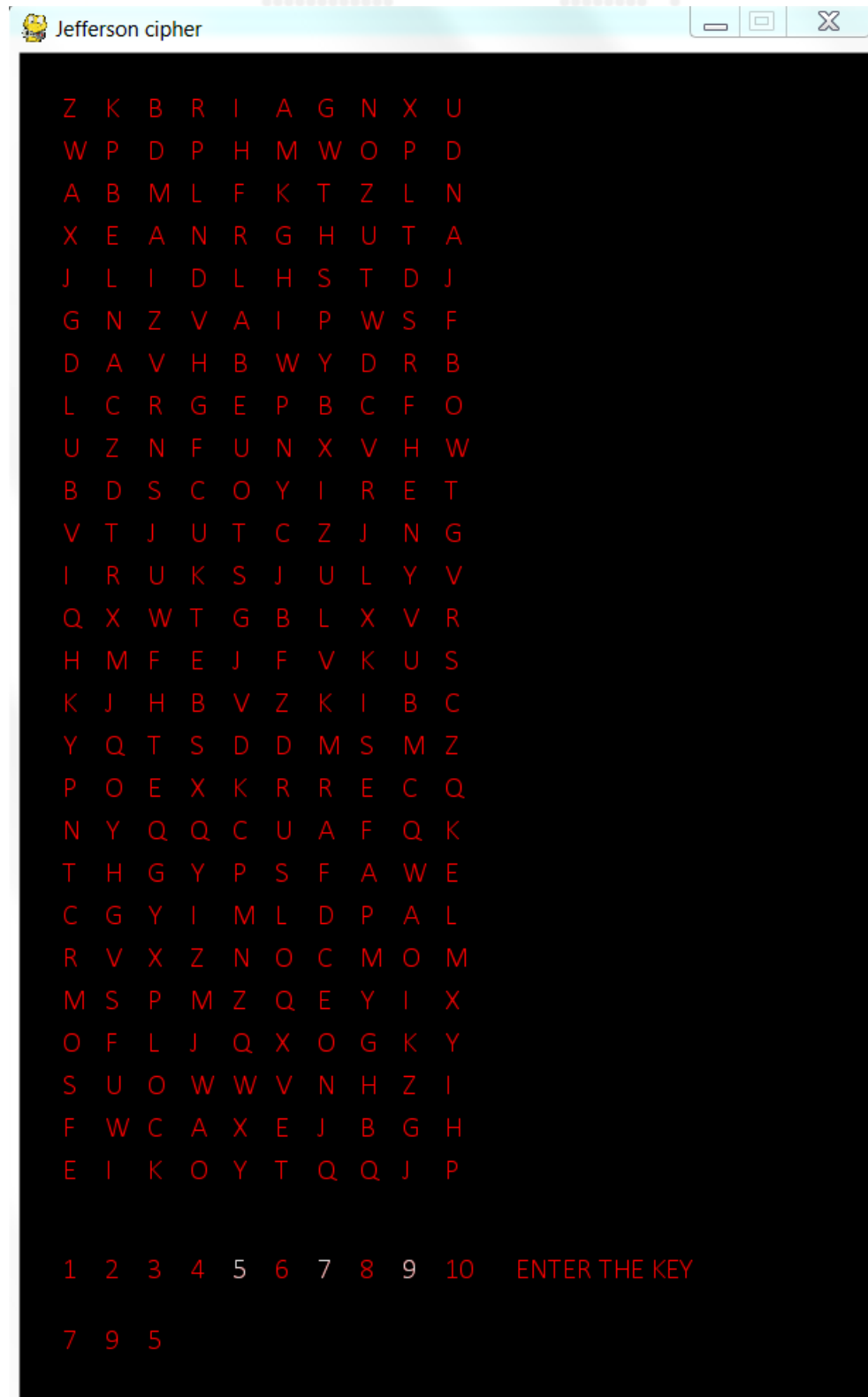
- A main procedure that gets a cylinder in a text file (Watch the previous part), creating a surface of size adapted to the number of disks, then uses the previous subroutines to enable the user to put a key and a message to encrypt. Once encrypted the message is then write on a text file.

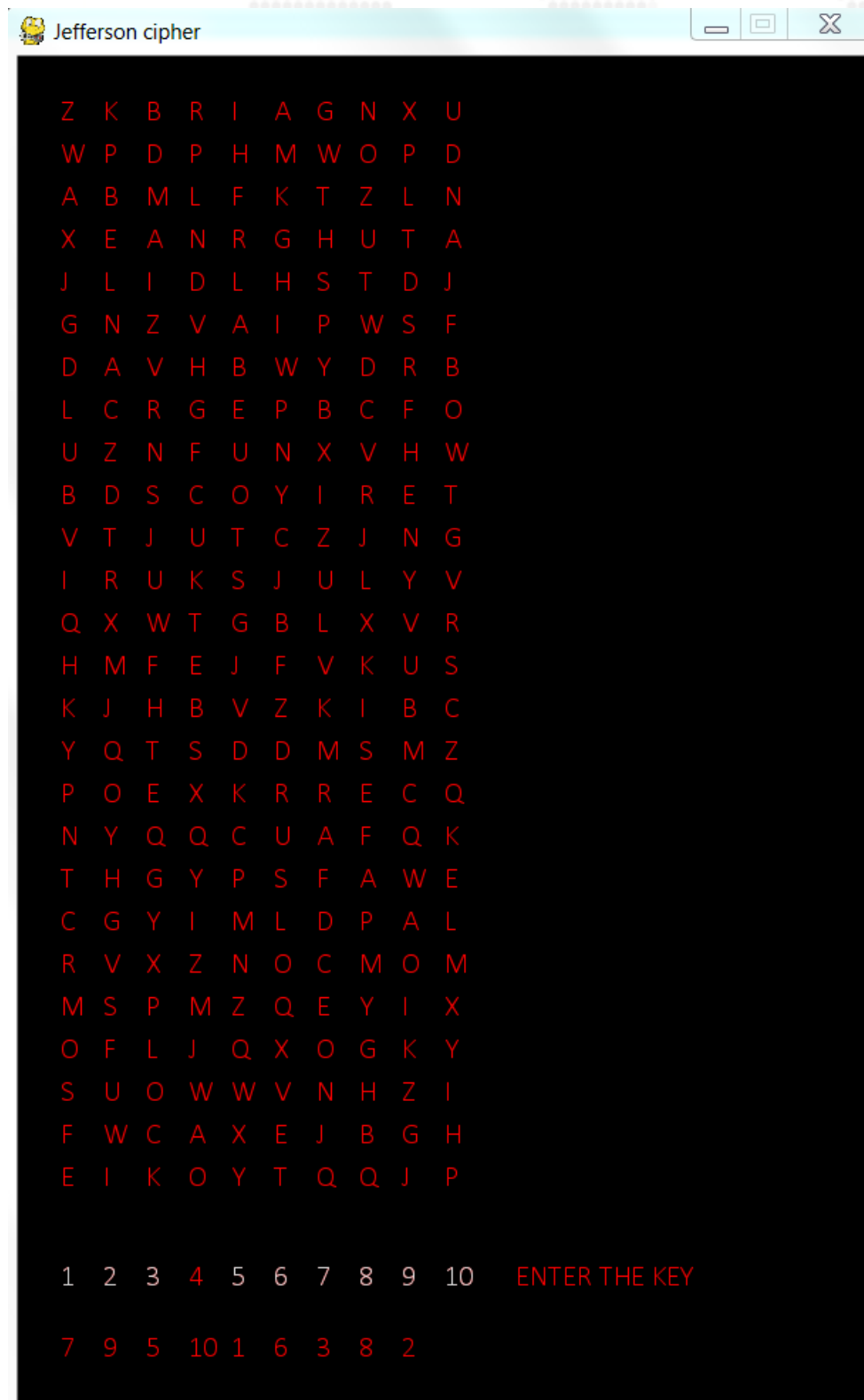
Example of running :

- Display of the disks from a text file (The same file of the previous part):

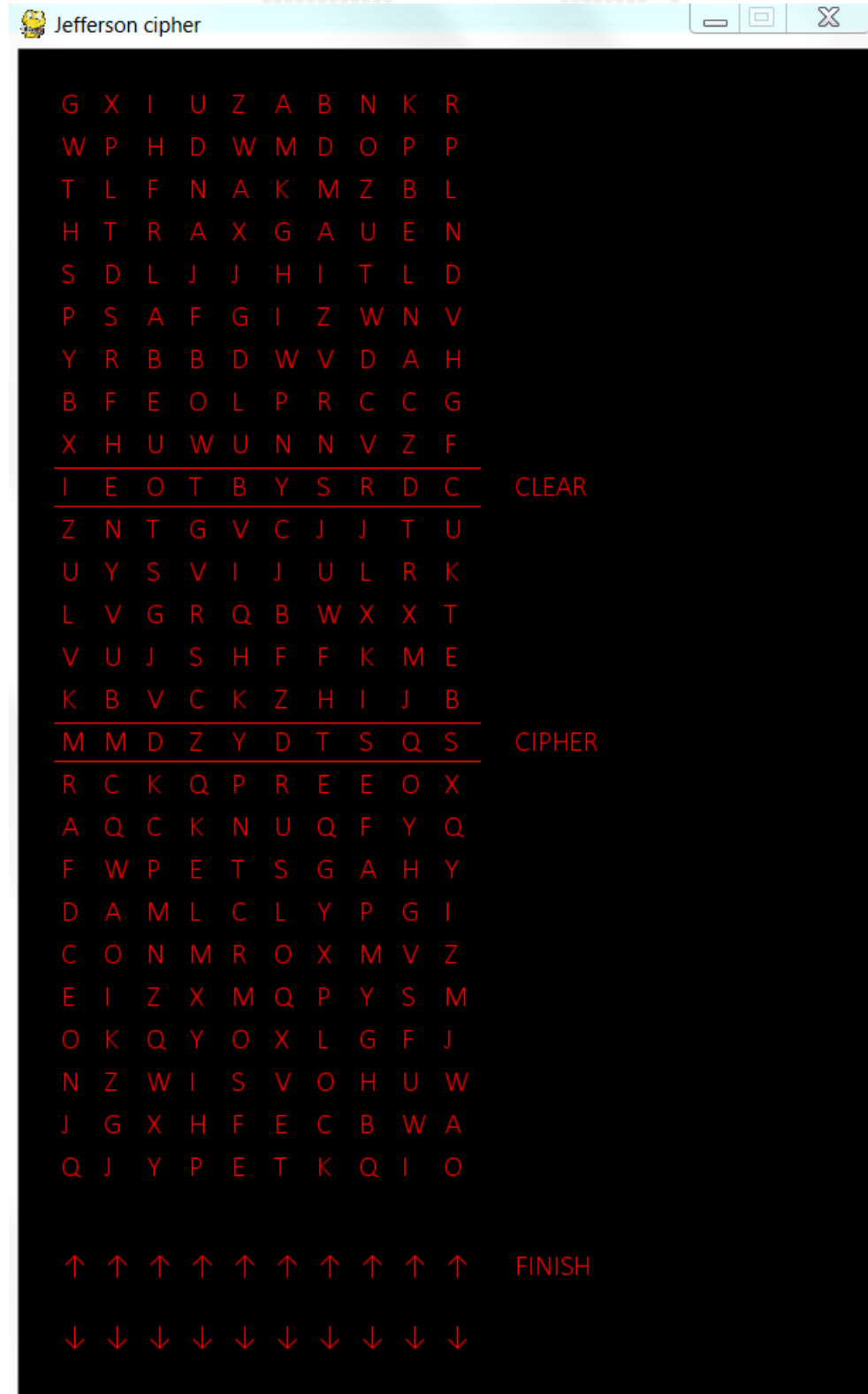


- Entering a key by clicking on the numbers of the disks :

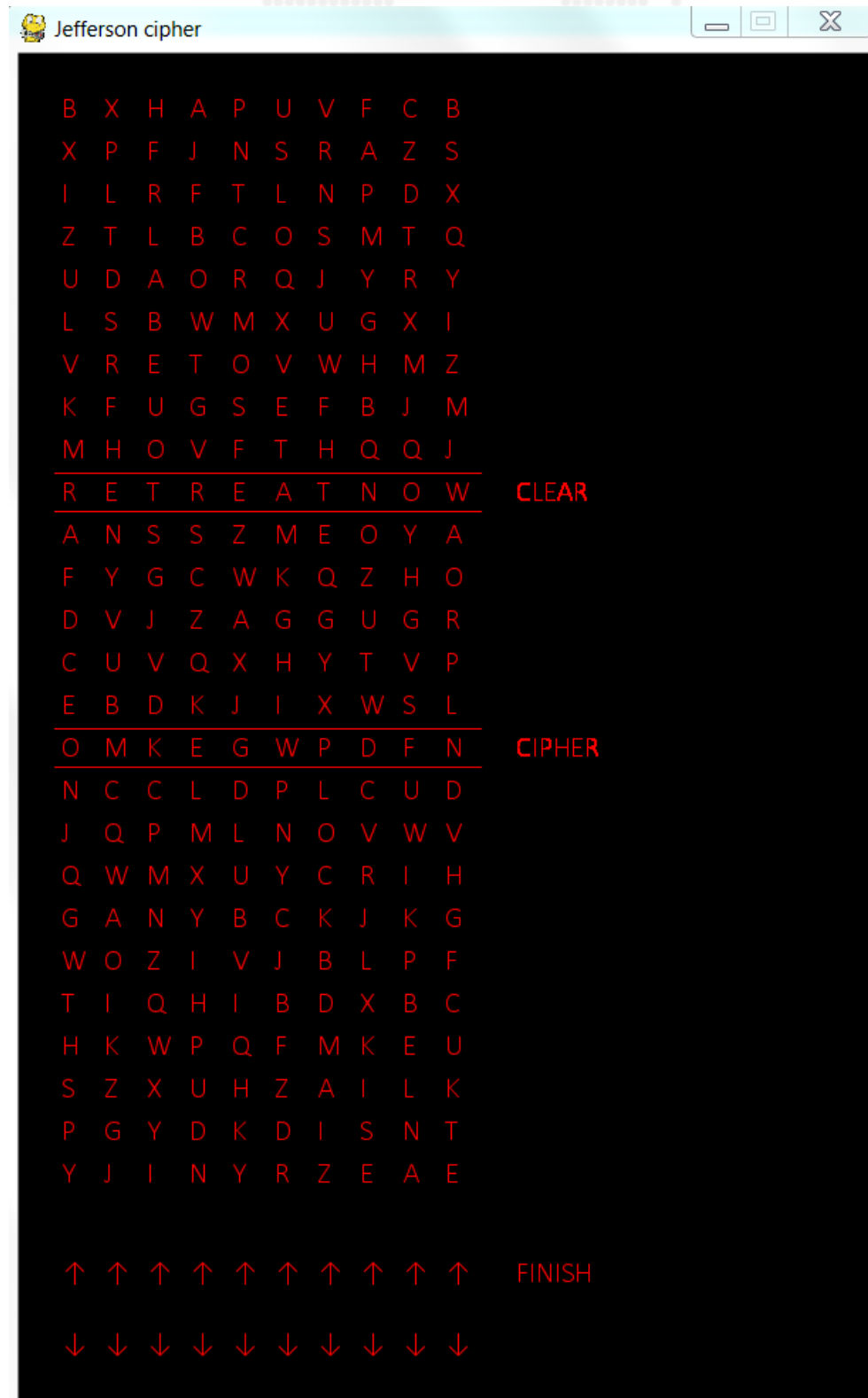




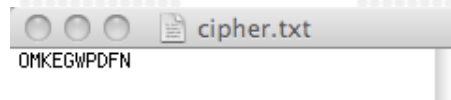
- Display of the disks in the order of the key:



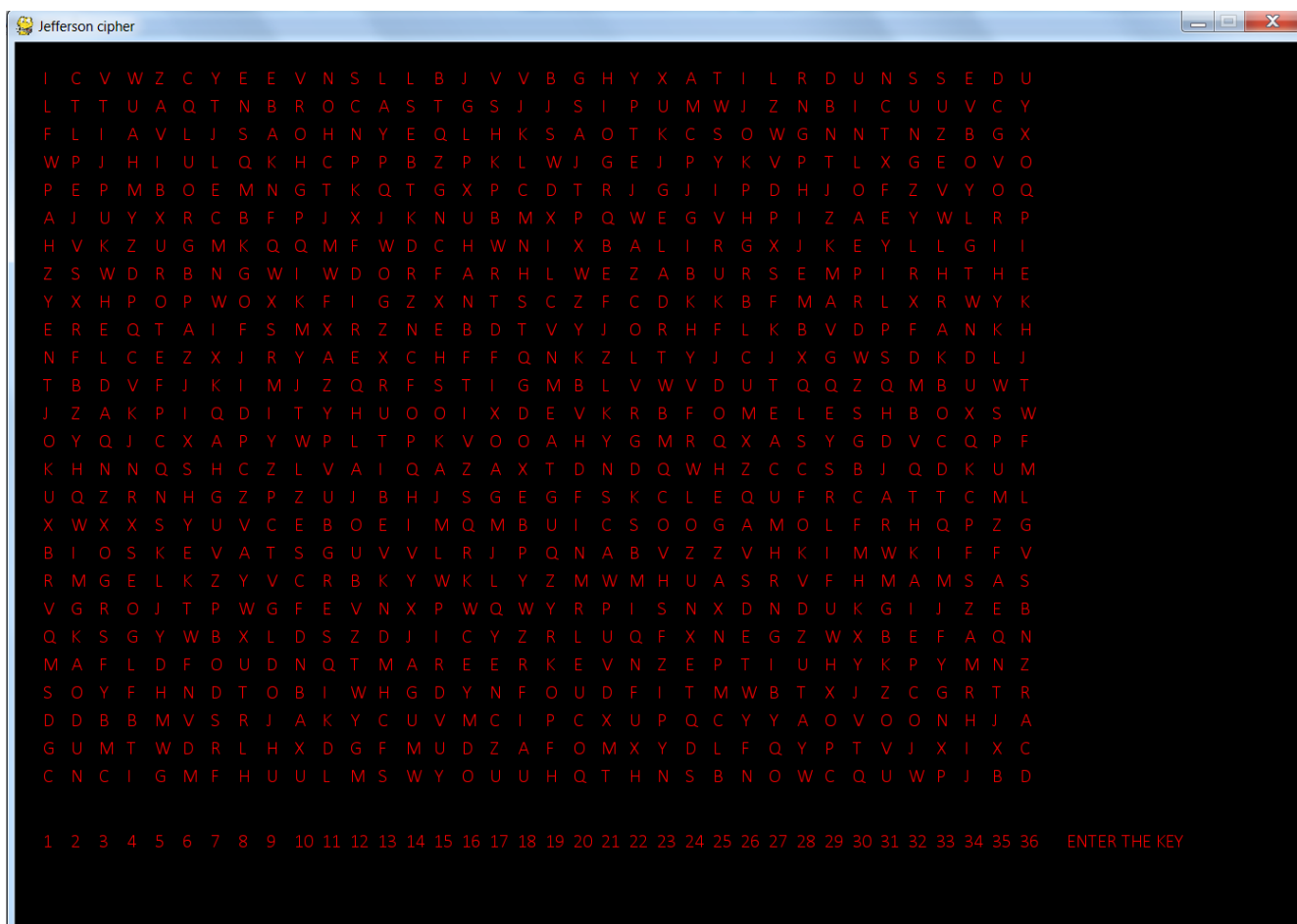
- Rotation of the disks to make appears the message to be encrypted, and therefore the encrypted message:



- The writing of the encrypted message on a text file:



An example of display with a greater number of disks. This shows the automatic sizing of the surface:



5 BONUS

Build physically, *i.e.* with disks and axis, a cylinder of Jefferson. The quality is expected for this bonus to be valued.

6 INDICATIVE SCALE

This scale can be set to change; it is therefore only indicative.

- Part 2 : 15 points
- Part 3 : 25 points
- Bonus : 10 points

This adds up to a total of 40 points, your viva being evaluated on 20 points.
The total grade will then be brought proportionally to a grade based on 20 points.