Assignment 1

General Instructions:

1. Your need to submit ONLY ONE file: called: "A1_solution.py". This should be strictly observed for the testing file to work. The simplest way is to download the "A1_solution.py" provided to you and make some edits.
2. At the top of the file: include (or edit) the following header (Any submission without this header will be rejected):

```
#-------------------------
# Your Name and ID
# CP460 (Fall 2019)
# Assignment 1
#-------------------------
```

3. Deadline: Sunday, September 29 at 11:59 pm.
4. Assignment is worth 10 pts of your course grade.
5. You may not include any packages/libraries other than the ones already included in the file.
6. You may create your own utility functions, as long you write them in the file: solution_A1.py.
7. Every new function you create should have a header outlining input parameters, return values and a description. similar to the following:

```
#-------------------------------------------------
# Parameters:   ciphertext(string)
#               key (none)
# Return:       plaintext (string)
# Description:  Decryption using Polybius Square
#-------------------------------------------------
```

8. Make sure to test your functions using the given test file: "test_A1.py" and using the given text files. You may edit your local version of the test file. However, since you will not be submitting the testing file, such changes will be discarded.

Q1: Spartan Scytale Decryption (2 pts)

The encryption scheme for the Spartan Scytale cipher `e_scytale(plaintext, key)` was completed in class and is included in the *`solution_A1.py`* file.

Write a decryption function `d_scytale(ciphertext, key)` that would decrypt any ciphertext produced through the scytale scheme using a given key.

Remember, a ***key*** represent the diameter of the rod, which corresponds to the number of characters that can be written in one of the rounds around the rod. This can be mapped to number of rows in a table. Also, remember that it is assumed that the rod can have an infinite length, i.e. there is no limit on the number of columns.

The function description is as follows:

```
#-------------------------------------------------
# Parameters:   ciphertext(string)
#               key (string)
# Return:       plaintext (string)
# Description:  Decryption using Scytale Cipher
#               Assumes key is a valid integer in string format
#-------------------------------------------------
def d_scytale(ciphertext, key):
    # your code here
    return plaintext
```

The decryption function should return the supposedly deciphered plaintext. When writing the function watch for the scenario when the last row has missing characters.

There are scenarios when the decryption process will fail. In that scenario, the function should return an empty string.

For instance: “MOVETHETROOPSSOUTH” when encrypted using key = 5 will give:

M	O	V	E	Note here how the empty slots appear on the last row.
T	H	E	T	
R	O	O	P	
S	S	O	U	
T	H			

Which produces a ciphertext of: “MTRSTOHOSHVEOOETPU”.

However, If you attempt to decrypt it using a key of 8, you will get:

M	S	P	This can not be true because empty slots should only appear on the last row.
T	H	U	
R	V		
S	E		
T	O		
O	O		
H	E		
O	T		

Therefore, in the above scenario, the function should return an empty string.

The testing module tests against the following files:

- “plaintext1.txt” (key = 4)
- “plaintext2.txt” (key = 5)
- “plaintext3.txt”(key = 8)
- “ciphertext1.txt” (key = 10)

Below is the result of executing the testing module:

```
>>> test_q1()
-----------------------------------------
Testing Q1: Decryption of Scytale Cipher

Plaintext1: MOVETHETROOPSSOUTH
Encryption using (k = 4 ): MHOUOEPTVTSHERSTOO
Decryption using (k = 4 ): MOVETHETROOPSSOUTH
Decryption using (k = 8 ):

Plaintext2: A Tale of Two Cities by "Charles Dickens".
Encryption using (k = 5 ): A eak TsreTw lnaobesl ys"eC  . i"DotCifihc
Decryption using (k = 5 ): A Tale of Two Cities by "Charles Dickens".

Plaintext3:
 Python features a dynamic type system and automatic memory management.
It supports multiple programming paradigms, including object-oriented, imperative, functional and
 procedural, and has a large and comprehensive standard library [27].
Encryption using (k = 8 ):
 P aedv sysn ieaityapn,nvhsgrg deoteo f  nemgouhs merbnatf najcsaeatmet nan.mciadtd
ito au In-nlrratgoaadeu  rlr stspi gl ouaeaeiamprnn b apatdardtode nayiridpdrnctg,r ya sm oc mm s
ico[iem,mem2cmu pdp7 olieur]trtnrre.yyicaahp pltleemlui,n
Decryption using (k = 8 ):
 Python features a dynamic type system and automatic memory management.
It supports multiple programming paradigms, including object-oriented, imperative, functional and
 procedural, and has a large and comprehensive standard library [27].
```

```
Decrypting ciphertext1 using (key = 10 ):

Python is an interpreted high-level programming language for general-purpose programming.
Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphas
izes code readability, notably using significant whitespace.
It provides constructs that enable clear programming on both small and large scales. [26]
Python features a dynamic type system and automatic memory management.
It supports multiple programming paradigms, including object-oriented, imperative, functional and
procedural, and has a large and comprehensive standard library. [27]
Python interpreters are available for many operating systems.
CPython, the reference implementation of Python, is open source software[28] and has a community-b
ased development model, as do nearly all of its variant implementations.
CPython is managed by the non-profit Python Software Foundation.
Python was conceived in the late 1980s, [29] and its implementation began in December 1989 [30] by
Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the
ABC language (itself inspired by SETL) [31].
It is capable of exception handling and interfacing with the Amoeba operating system. [7]
Van Rossum remains Python's principal author. His continuing central role in Python's development
is reflected in the title given to him by the Python community: Benevolent Dictator For Life (BDFL
).
Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-dete
cting garbage collector and support for Unicode.
With this release, the development process became more transparent and community-backed. [33]
Python 3.0 (initially called Python 3000 or py3k) was released on 3 December 2008 after a long tes
ting period.
It is a major revision of the language that is not completely backward-compatible with previous ve
rsions. [34]
However, many of its major features have been backported to the Python 2.6.x [35] and 2.7.x versio
n series, and releases of Python 3 include the 2to3 utility, which automates the translation of Py
thon 2 code to Python 3. [36]

source: wikipedia.com
-----------------------------------------
```

Q2: Plaintext Detection
(3.5 pts)

As human beings we have the capacity to recognize if a given file is a plaintext or a ciphertext file. Through sight inspection, we can recognize whether the words in the file are English words or gibberish (cipher). However, this intuitive process is not obvious to computer machines, as the machine interprets both as stream of characters.

A simple method to automate the process is to read through a file word by word and check if the words appear in an English dictionary. If the majority of words matched some entries in the English dictionary, then it is more likely a plaintext file. The reverse is also true.

In this task you will write some functions to detect if a file is a plaintext or ciphertext. An English dictionary file: "`engmix.txt`" is provided to you.

First, write a function called: `load_dictionary(dictFile)`. The function reads a given dictionary file and return a list in which every element corresponds to a dictionary word. Assume the dictionary is formatted such that every word appear on a separate line. Below is the function definition:

```
#-----------------------------------------------------------
# Parameters:   dictFile (string): filename
# Return:       list of words (list)
# Description:  Reads a given dictionary file
#               dictionary file is assumed to be formatted:
#               each word in a separate line
#               Returns a list of strings, each pertaining to a dictionary word
#-----------------------------------------------------------
def load_dictionary(dictFile):
    # your code
    return dictList
```

This word list will be used later to sequentially search a dictionary. Yes, from the efficiency perspective this is horrible, but we will work on improving it later in the course.

Running the testing module will yield the following:

```
>>> test_q2()
-----------------------------------------
Testing Q2: Plaintext Detection

Testing load_dictionary:
Word #111:       aberrational
Word #2222:      altruism
Word #2700:      ancestry
Word #35000:     highjacker
Word #64000:     rubbernecks
```

Second, write a function called: `text_to_words(text)`. The function reads any given string and returns a list of words each pertaining to a word in that string.

There are three distinctions between this function and the `load_dictionary` one. First, the text is passed as a string. Therefore, there is no requirement to read from file. Second, there is no assumption that each word appear in a separate line. In such scenario, spaces and end of line would mark the end of a given word. Third, since some words might be preceded or proceeded by a punctuation symbol, the function should "clean" each word from punctuations appearing at the start or at the end of the word.

Below is the function definition:

```
#---------------------------------------------------------------------
# Parameters:   text (string)
# Return:       list of words (list)
# Description:  Reads a given text
#               Each word is saved as an element in a list.
#               Returns a list of strings, each pertaining to a word in file
#               Gets rid of all punctuation at the start and at the end
#---------------------------------------------------------------------
def text_to_words(text):
    # your code
    return wordlist
```

Running the testing module will yield the following:

```
Testing text_to_words:
plaintext1: ['MOVETHETROOPSSOUTH']
plaintext2:
['A', 'Tale', 'of', 'Two', 'Cities', 'by', 'Charles', 'Dickens']
plaintext3:
['Python', 'features', 'a', 'dynamic', 'type', 'system', 'and', 'automatic', 'memory',
'management', 'It', 'supports', 'multiple', 'programming', 'paradigms', 'including', '
object-oriented', 'imperative', 'functional', 'and', 'procedural', 'and', 'has', 'a',
'large', 'and', 'comprehensive', 'standard', 'library', '27']
```

Third, write a function called: `analyze_text(text, dictFile)`. The function calls the above two functions (`load_dictionary` and `text_to_words`) and return a tuple, in which the first element represents the number of dictionary matches and the second represents the number of dictionary mismatches.

Note that words are to be compared as lowercase letters. Below is the function definition:

```
#-------------------------------------------------------------
# Parameters:   text (string)
#               dictFile (string): dictionary file
# Return:       (#matches, #mismatches)
# Description:  Reads a given text, checks if each word appears in dictionary
#               Returns a tuple of number of matches and number of mismatches.
#               Words are compared in lowercase.
#-------------------------------------------------------------
def analyze_text(text, dictFile):
    # your code
    return(matches,mismatches)
```

Testing the function will give:

```
Testing analyze_text:
Analyzing plaintext2: (8, 0)
Analyzing plaintext3: (28, 2)
Analyzing plaintext4: (260, 55)
Analyzing ciphertext1: (53, 217)
```

Finally, write a function called `is_plaintext(text, dictFile, threshold)`. The function calls `analyze_text` and calculates the percentage of matches to the total number of words. If the percentage is greater than or equal to the threshold, then the given file is considered a plaintext. Otherwise, the file is not considered a plaintext. Below is the function definition:

```
#-------------------------------------------------------------
# Parameters:   text (string)
#               dictFile (string): dictionary file
#               threshold (float): number between 0 to 1
# Return:       True/False
# Description:  Check if a given file is a plaintext
#               If #matches/#words >= threshold --> True
#                   otherwise --> False
#               If invalid threshold given, default is 0.9
#               An empty string is assumed to be non-plaintext.
#-------------------------------------------------------------
def is_plaintext(text, dictFile, threshold):
     # your code
    return False
```

Executing the testing module should yield the following:

```
Testing is_plaintext:
plaintext2 (0.85): True
plaintext3 (1.10): True
plaintext3: (0.96) False
plaintext4: (0.91) False
plaintext4: (0.82) True
ciphertext1: (0.7) False
-----------------------------------------
```

Q3: Cryptanalysis of Spartan Scytale Cipher (1 pt)

In Q2, you have created several functions which will be useful to cryptanalysis of spartan scytale cipher.

Write a function called: `cryptanlaysis_scytale(p1,p2,p3,p4,p5)`. The function will use a brute-force attack, for a given key range.

The function takes the following parameters

1- The filename containing the ciphertext
2- The dictionary filename
3- The brute force starting key
4- The brute force end key
5- Threshold (to be used in text_analyze)

Below is the function definition:

```
#-----------------------------------------------------
# Parameters:   cipherFile (string)
#               dictFile (string)
#               startKey (int)
#               endKey (int)
#               threshold (float)
# Return:       key (string)
# Description:  Apply brute-force to break scytale cipher
#               Valid key range: 2-100
#                     (if invalid --> print error msg and return '')
#               Valid threshold: 0-1 (if invalid --> print error msg and return '')
#               If decryption is successful --> print plaintext and return key
#               If decrytpoin fails: print error msg and return ''
#-----------------------------------------------------
def cryptanalysis_scytale(cipherFile, dictFile, startKey, endKey, threshold):
    # your code
```

The function should loop through the given key range. If decryption is successful, the function should stop and print the found key and the recovered plaintext.

If the function fails, it should print an error message as displayed by the following screenshots.

```
>>> test_q3()
-----------------------------------------------
Testing Q3: Scytale Cipher Cryptanalysis

Case 1: ciphertext1.txt engmix.txt 4 - 300 0.8 :
Invalid key range. Operation aborted!
Returned Key =

Case 2: ciphertext1.txt engmix.txt 3 - 50 1.2 :
Invalid threshold value. Operation aborted!
Returned Key =
```

```
Case 3: ciphertext1.txt engmix.txt 7 - 70 0.8 :
key  7 failed
key  8 failed
key  9 failed

Key found: 10
Python is an interpreted high-level programming language for general-purpose programming.
Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability,
notably using significant whitespace.
It provides constructs that enable clear programming on both small and large scales. [26]
Python features a dynamic type system and automatic memory management.
It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a
large and comprehensive standard library. [27]
Python interpreters are available for many operating systems.
CPython, the reference implementation of Python, is open source software[28] and has a community-based development model
, as do nearly all of its variant implementations.
CPython is managed by the non-profit Python Software Foundation.
Python was conceived in the late 1980s, [29] and its implementation began in December 1989 [30] by Guido van Rossum at C
entrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) [31]
.
It is capable of exception handling and interfacing with the Amoeba operating system. [7]
Van Rossum remains Python's principal author. His continuing central role in Python's development is reflected in the ti
tle given to him by the Python community: Benevolent Dictator For Life (BDFL).
Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collecto
r and support for Unicode.
With this release, the development process became more transparent and community-backed. [33]
Python 3.0 (initially called Python 3000 or py3k) was released on 3 December 2008 after a long testing period.
It is a major revision of the language that is not completely backward-compatible with previous versions. [34]
However, many of its major features have been backported to the Python 2.6.x [35] and 2.7.x version series, and releases
of Python 3 include the 2to3 utility, which automates the translation of Python 2 code to Python 3. [36]

source: wikipedia.com
Returned Key =  10
```

```
Case 4: ciphertext1.txt engmix.txt 20 - 28 0.8 :
key  20 failed
key  21 failed
key  22 failed
key  23 failed
key  24 failed
key  25 failed
key  26 failed
key  27 failed
key  28 failed
No key was found
Returned Key =

Case 5: ciphertext1.txt engmix.txt 7 - 17 0.9 :
key  7 failed
key  8 failed
key  9 failed
key  10 failed
key  11 failed
key  12 failed
key  13 failed
key  14 failed
key  15 failed
key  16 failed
key  17 failed
No key was found
Returned Key =
```

```
Case 6: ciphertext2.txt engmix.txt 30 - 50 0.9 :
key  30 failed
key  31 failed
key  32 failed
key  33 failed
key  34 failed
key  35 failed
key  36 failed

Key found: 37
With drooping heads and tremulous tails, they mashed their way through the thick mud,
floundering and stumbling between whiles,
as if they were falling to pieces at the larger joints.
As often as the driver rested them and brought them to a stand,
with a wary "Wo-ho! so-ho-then!" the near leader violently shook his head and everything
upon it-like an unusually emphatic horse, denying that the coach could be got up the hill.
Whenever the leader made this rattle, the passenger started, as a nervous passenger might,
and was disturbed in mind.
Returned Key =  37

-------------------------------------------
```

Q4: Polybius Cipher Encryption (2.5 pts)

A customized version of Polybius square will be introduced that would cover the English alphabet and basic punctuations. The original English adapted square is 5x5, however, this customized version is 8x8.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
[1]		!	“	#	$	%	&	‘
[2]	(	)	*	+	,	-	.	/
[3]	0	1	2	3	4	5	6	7
[4]	8	9	:	;	<	=	>	?
[5]	@	A	B	C	D	E	F	G
[6]	H	I	J	K	L	M	N	O
[7]	P	Q	R	S	T	U	V	W
[8]	X	Y	Z	[	\	]	^	_

First, write a function called: `get_polybius_square()`. The function should return the above square as one sequential string (Yes, it is no more a square!). If you know ASCII, you would have recognized that the above table represent a sequential character set in ASCII, from 32 to 95 (inclusive). Your implementation should not hard-code of the string, instead get use of the above observation about the ASCII order.

Second, write a function called: `e_polybius()`.The function reads through a given stream of characters and encrypt each character as a pair of two numbers, the first is row# and the second is column#. For instance, ‘A’ would be encrypted as: 52 and ‘IN’ would be encrypted as: “6267”.

Your function should call first the `get_polybius_square` function. Remember, this is a string, not a 2D list, so you need to find an equation that would get you the row# and column#, relative to the index of that character in the square string.

If a ‘\n’ is available in the plaintext, then it should not be encrypted. Instead, a ‘\n’ is inserted in the ciphertext.

You may assume that the plaintext will only have characters defined in the 8x8 square, or the '\n', but no other characters, other than the lowercase alphabet.

Note that this encryption scheme will always convert a character (upper or lower case) to a pair of numbers. There is no way to enable detection of character case through the given 8x8 square. However, this is possible if we expand the square to include lower case characters, something you can explore by your own.

Running the testing module will give the following:

```
>>> test_q4()
-----------------------------------------
Testing Q4: Polybius Square Encryption

Testing get_polybius_square
Polybius Square is:
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_

plaintext1:  MOVETHETROOPSSOUTH
ciphertext:  66687756756156757368687174746876756 1

plaintext2:
A Tale of Two Cities by "Charles Dickens".
ciphertext:
52117552655611685711757868115462756256741153821113546152736556741155625464566774 1327

plaintext3:
Python features a dynamic type system and automatic memory management.
It supports multiple programming paradigms, including object-oriented, imperative, functional
and procedural, and has a large and comprehensive standard library [27].
ciphertext:
71827561686711575652757673567411521155826752666254117582715611748274755666115267551152767568 6
65275625411665666687382116652675258566656677527
62751174767171687375741166766575627165561171736858735266666267581171527352556258667425116267 5
46576556267581168536356547526687362566775565525116266715673527562775625115776675475626867526 5
11526755117173685456557673526525115267551161527411521165527358561152675511546866717356615667 7
46277561174755267555273551165625373527382118433388627

-----------------------------------------
```

Q5: Decryption Using Polybius Cipher (2 pts)

Write a function called: `d_polybius()`. The function applies the decryption scheme for the polybius cipher scheme defined in Q4. At the start of the function call get_polybius_square function to get the square as a string.

Below is the function definition:

```
#-------------------------------------------------------
# Parameters:   ciphertext(string)
#               key (none)
# Return:       plaintext (string)
# Description:  Decryption using Polybius Square Cipher
#               Detects invalid ciphertext --> print error msg and return ''
#               Case 1: #of chars (other than \n) is not even
#             Case 2: the ciphertext contains non-numerical chars (except \n')
#-------------------------------------------------------
def d_polybius(ciphertext, key):
    #your code
    return plaintext
```

The main challenge is to translate each two numbers in the ciphertext to the proper character position in square string.

Note that you might encounter a '\n' in the ciphertext. This should translate into a '\n' in the plaintext at that position of the text.

The function should detect scenarios in which an invalid ciphertext is provided. In that case, the function should print an error message and return an empty string as plaintext.

The first invalid scenario occurs when the ciphertext contains non-numerical characters. An exception is '\n' which is allowed to occur in the ciphertext. An example, "71A55" is an invalid cipher because there is no way it could have been generated through a Polybius cipher encryption scheme.

The second scenario when the number of characters is not even, excluding '\n'. For instance: "71\n5" is an invalid cipher because there is no way that the last number correspond to a character (we need to numbers).

Running the testing module would give:

```
>>> test_q5()
-----------------------------------------
Testing Q5: Polybius Square Decryption

Decrypting: 7561627411627411716865825362767412
THIS IS POLYBIUS!

Decrypting:
75616274
6274
716865825362767412

THIS
IS
POLYBIUS!

Decrypting: 7561ABC5825362767412
Invalid ciphertext! Decryption Failed!

Decrypting:
75616274
6274
71686582536277412

Invalid ciphertext! Decryption Failed!
```

```
Decrypting file ciphertext3.txt
PYTHON IS AN INTERPRETED HIGH-LEVEL PROGRAMMING LANGUAGE FOR GENERAL-PURPOSE PROGRAMMING.
CREATED BY GUIDO VAN ROSSUM AND FIRST RELEASED IN 1991, PYTHON HAS A DESIGN PHILOSOPHY THAT EMPHASIZES CODE READABILITY, NOTABLY USING
SIGNIFICANT WHITESPACE.
IT PROVIDES CONSTRUCTS THAT ENABLE CLEAR PROGRAMMING ON BOTH SMALL AND LARGE SCALES. [26]
PYTHON FEATURES A DYNAMIC TYPE SYSTEM AND AUTOMATIC MEMORY MANAGEMENT.
IT SUPPORTS MULTIPLE PROGRAMMING PARADIGMS, INCLUDING OBJECT-ORIENTED, IMPERATIVE, FUNCTIONAL AND PROCEDURAL, AND HAS A LARGE AND COMP
REHENSIVE STANDARD LIBRARY. [27]
PYTHON INTERPRETERS ARE AVAILABLE FOR MANY OPERATING SYSTEMS.
CPYTHON, THE REFERENCE IMPLEMENTATION OF PYTHON, IS OPEN SOURCE SOFTWARE[28] AND HAS A COMMUNITY-BASED DEVELOPMENT MODEL, AS DO NEARLY
ALL OF ITS VARIANT IMPLEMENTATIONS.
CPYTHON IS MANAGED BY THE NON-PROFIT PYTHON SOFTWARE FOUNDATION.
PYTHON WAS CONCEIVED IN THE LATE 1980S, [29] AND ITS IMPLEMENTATION BEGAN IN DECEMBER 1989 [30] BY GUIDO VAN ROSSUM AT CENTRUM WISKUND
E & INFORMATICA (CWI) IN THE NETHERLANDS AS A SUCCESSOR TO THE ABC LANGUAGE (ITSELF INSPIRED BY SETL) [31].
IT IS CAPABLE OF EXCEPTION HANDLING AND INTERFACING WITH THE AMOEBA OPERATING SYSTEM. [7]
VAN ROSSUM REMAINS PYTHON'S PRINCIPAL AUTHOR. HIS CONTINUING CENTRAL ROLE IN PYTHON'S DEVELOPMENT IS REFLECTED IN THE TITLE GIVEN TO H
IM BY THE PYTHON COMMUNITY: BENEVOLENT DICTATOR FOR LIFE (BDFL).
PYTHON 2.0 WAS RELEASED ON 16 OCTOBER 2000 AND HAD MANY MAJOR NEW FEATURES, INCLUDING A CYCLE-DETECTING GARBAGE COLLECTOR AND SUPPORT
FOR UNICODE.
WITH THIS RELEASE, THE DEVELOPMENT PROCESS BECAME MORE TRANSPARENT AND COMMUNITY-BACKED. [33]
PYTHON 3.0 (INITIALLY CALLED PYTHON 3000 OR PY3K) WAS RELEASED ON 3 DECEMBER 2008 AFTER A LONG TESTING PERIOD.
IT IS A MAJOR REVISION OF THE LANGUAGE THAT IS NOT COMPLETELY BACKWARD-COMPATIBLE WITH PREVIOUS VERSIONS. [34]
HOWEVER, MANY OF ITS MAJOR FEATURES HAVE BEEN BACKPORTED TO THE PYTHON 2.6.X [35] AND 2.7.X VERSION SERIES, AND RELEASES OF PYTHON 3 I
NCLUDE THE 2TO3 UTILITY, WHICH AUTOMATES THE TRANSLATION OF PYTHON 2 CODE TO PYTHON 3. [36]

SOURCE: WIKIPEDIA.COM

-----------------------------------------
```