```
"""
This module contains all functions with which the user's text can be treated.
"""


# ******************************** begin CAESAR ******************************** #
# *** ENCRYPTION *** #
def encrypt_caesar(text, key):
    """
    encrypts and returns the given text by using the principle of Caesar
    :param text: user's text (string)
    :param key: user's key (number)
    :return: text: encrypted text
    """
    encrypted_text = ''
    for i in text:
        # if by adding the key to the number of the letter,we go beyond the numbers between A and Z
        # we must go back to the beginning of the alphabet by using a modulo (26)
        # ord('A') because each letter got a number starting 65=ord('A')
        i = chr((ord(i) + key) % 26 + ord('A'))
        encrypted_text += i
    return encrypted_text


# *** DECRYPTION *** #
def decrypt_caesar(text, key):
    """
    decrypts and returns the given text by using the principle of Caesar
    :param text: user's text (string)
    :param key: user's key (number)
    :return: text: decrypted text
    """
    decrypted_text = ''
    for i in text:
        # if by removing the key to the number of the letter, we go beyond the numbers between A and Z
        # we must go back to the beginning of the alphabet by using a modulo (26)
```

```
37          # ord('A') because each letter got a number
   starting 65=ord('A')
38          i = chr((ord(i) - key) % 26 + ord('A'))
39          decrypted_text += i
40      return decrypted_text
41
42
43  # ********************************* end CAESAR
     ******************************** #
44
45
46  # ********************************* begin VIGENERE
     ******************************** #
47  # *** HELP FUNCTIONS *** #
48  def create_vigenere_table():
49      """
50      creates a table 26x26 with the alphabet that recreates
    the Vigenere table
51      :return: the table
52      """
53      alphabet = [chr(x) for x in range(ord('A'), ord('Z') +
    1)]
54      table = []
55      for k in range(len(alphabet)):  # fill table
56          # alphabet is shifted -1 in each row and column
57          table += [alphabet[k:] + alphabet[:k]]
58      return table
59
60
61  def create_table_text_key(text, key):
62      """
63      creates a table with the user's text in the first line
    and the repeated key in the second
64      :param text: string
65      :param key: string
66      :return: table
67      """
68      key = key.upper()
69      repeated_key = ""
70      # we repeat the key until its length is bigger than
   the length of the text
71      while len(repeated_key) < len(text):
72          repeated_key += key
73      list_key = []
74      # in a table of two lines, we add in the first line
```

```
 74 the text and in the second line
 75     # the repeated key(of the same length as the text)
 76     for i in range(len(text)):
 77         list_temp = [text[i], repeated_key[i]]
 78         list_key.append(list_temp)
 79     return list_key
 80
 81
 82 # *** ENCRYPTION *** #
 83 def encrypt_vigenere(text, key):
 84     """
 85     encrypts and returns the given text by using the
    principle of Vigenere
 86     :param text: user's text (string)
 87     :param key: user's key (string)
 88     :return: text: encrypted text
 89     """
 90     table_of_vigenere = create_vigenere_table()
 91     repeated_key_table = create_table_text_key(text, key)
 92     encrypted_text = ""
 93     alphabet = [chr(x) for x in range(ord('A'), ord('Z')
    + 1)]
 94     for i in range(len(text)):
 95         # column corresponding to the letter of the text
 96         text_letter = repeated_key_table[i][0]
 97         # line corresponding to the letter of the key
 98         key_letter = repeated_key_table[i][1]
 99         encrypted_text += table_of_vigenere[alphabet.
    index(text_letter)][alphabet.index(key_letter)]
100     return encrypted_text
101
102
103 # *** DECRYPTION *** #
104 def decrypt_vigenere(text, key):
105     """
106     decrypts and returns the given text by using the
    principle of Vigenere
107     :param text: user's text (string)
108     :param key: user's key (string)
109     :return: text: decrypted text
110     """
111     vigenere_table = create_vigenere_table()
112     repeated_key_table = create_table_text_key(text, key)
113     decrypted_text = ""
114     alphabet = [chr(x) for x in range(ord('A'), ord('Z')
```

```
114 + 1)]
115     for i in range(len(text)):
116         # column corresponding to the letter of the text
117         letter_text = repeated_key_table[i][0]
118         # lign corresponding to the letter of the key
119         letter_key = repeated_key_table[i][1]
120         list_temp = vigenere_table[alphabet.index(
    letter_key)]
121         decrypted_text += alphabet[list_temp.index(
    letter_text)]
122     return decrypted_text
123
124
125 # ******************************** end VIGENERE
     ******************************** #
126
127
128 # ******************************** begin ENIGMA
     ******************************** #
129 # *** HELP FUNCTIONS *** #
130 # * Help functions * #
131 def create_initial_list():
132     """
133     creates a list filled with the alphabet (capital
    letters)
134     :return: list
135     """
136     initial_list = [chr(x) for x in range(65, 91)]  #
    fill list thanks to Ascii code
137     return initial_list
138
139
140 def search_index(initial_list, letter):
141     """
142     searches the index of a given letter in the given
    list
143     :param initial_list: list for searching
144     :param letter: index of this letter is wanted
145     :return: index of the letter
146     """
147     i = 0
148     # index initialized with 999 because
149     # if something goes wrong in the loop the program
    stops after the execution of this function
150     index = 999
```

```
151
152     while i < len(initial_list):
153         if initial_list[i] == letter:
154             # if the given letter is found in the list,
    set the index to the actual position
155             index = i
156         i += 1
157     else:
158         # check if the index is out of range
159         if (index < 0) or (index > len(initial_list)):
160             index %= len(initial_list)
161         # check if something went wrong before
162         if index > len(initial_list):
163             return "This should never happen"
164         else:
165             return index
166
167
168 def plugboard(letter):
169     """
170     represents the plugboard, i. e. checks if a given
    letter is plugged to another.
171     If yes returns this letter, if no returns the given
    letter.
172     :param letter: the given letter (char)
173     :return: letter: either the plugged or the given
    letter (char)
174     """
175     # In this case the plugboard's setting is the one of
    the 31th of a month,
176     # for more information see 'https://en.wikipedia.org/
    wiki/Enigma_rotor_details'
177     # the plug combination is:
178     # A-H, B-L, C-X, D-I, E-R, F-K, G-U, N-P, O-Q, T-Y
179     # (J, M, S, V, W and Z not plugged)
180     if letter in "JMSVWZ":
181         return letter
182     elif letter in "A":
183         return "H"
184     elif letter in "B":
185         return "L"
186     elif letter in "C":
187         return "X"
188     elif letter in "D":
189         return "I"
```

```
190     elif letter in "E":
191         return "R"
192     elif letter in "F":
193         return "K"
194     elif letter in "G":
195         return "U"
196     elif letter in "N":
197         return "P"
198     elif letter in "O":
199         return "Q"
200     elif letter in "T":
201         return "Y"
202     elif letter in "H":
203         return "A"
204     elif letter in "L":
205         return "B"
206     elif letter in "X":
207         return "C"
208     elif letter in "I":
209         return "D"
210     elif letter in "R":
211         return "E"
212     elif letter in "K":
213         return "F"
214     elif letter in "U":
215         return "G"
216     elif letter in "P":
217         return "N"
218     elif letter in "Q":
219         return "O"
220     elif letter in "Y":
221         return "T"
222     else:
223         return "This should never happen"
224
225
226 def permutation_reflector(letter):
227     """
228     represents the reflector, i. e. returns the
    corresponding letter of the given letter
229     :param letter: char
230     :return: letter: char
231     """
232     # In this case the reflector's setting is the one of
    roller 'A',
```

```
233     # for more information see 'https://en.wikipedia.org/
    wiki/Enigma_rotor_details'
234     # the permutation is the following:
235     # A-E, B-J, C-M, D-Z, F-L, G-Y, H-X, I-V, K-W, N-R, O
    -Q, P-U, S-T (and in the other direction)
236     if letter in "A":
237         return "E"
238     elif letter in "B":
239         return "J"
240     elif letter in "C":
241         return "M"
242     elif letter in "D":
243         return "Z"
244     elif letter in "E":
245         return "A"
246     elif letter in "F":
247         return "L"
248     elif letter in "G":
249         return "Y"
250     elif letter in "H":
251         return "X"
252     elif letter in "I":
253         return "V"
254     elif letter in "J":
255         return "B"
256     elif letter in "K":
257         return "W"
258     elif letter in "L":
259         return "F"
260     elif letter in "M":
261         return "C"
262     elif letter in "N":
263         return "R"
264     elif letter in "O":
265         return "Q"
266     elif letter in "P":
267         return "U"
268     elif letter in "Q":
269         return "O"
270     elif letter in "R":
271         return "N"
272     elif letter in "S":
273         return "T"
274     elif letter in "T":
275         return "S"
```

```
276     elif letter in "U":
277         return "P"
278     elif letter in "V":
279         return "I"
280     elif letter in "W":
281         return "K"
282     elif letter in "X":
283         return "H"
284     elif letter in "Y":
285         return "G"
286     elif letter in "Z":
287         return "D"
288     else:
289         return "This should never happen"
290 
291 
292 # * functions representing the rotors * #
293 def shift_first_rotor(return_path, index, offset):
294     """
295     represents the letter shift of the first rotor and returns the corresponding letter
296     :param offset: int: represents the physical rotation of the rotor
297     :param return_path: boolean: True if it's the return path (after the reflector) or False if it's the forward path
298     :param index: the index of the actual letter
299     :return letter: letter
300     """
301     # In this case the rotor's shift is the one of roller 'I',
302     # for more information see 'https://en.wikipedia.org/wiki/Enigma_rotor_details'
303     alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
304     rotor1 = 'EKMFLGDQVZNTOWYHXUSPAIBRCJ'
305     # shift_list calculates the shift of each letter between the alphabet and rotor1
306     shift_list = [(ord(rotor1[i]) - ord(alphabet[i])) % len(alphabet) for i in range(len(alphabet))]
307     letter = alphabet[index]
308     if return_path is False:
309         # assigns a new letter given the shift and the offset
310         letter = alphabet[(index + shift_list[(index + offset) % len(alphabet)]) % len(alphabet)]
```

```
311         if letter.isalpha():
312             return letter
313         else:
314             print("This should never happen")
315             return -1
316     elif return_path:
317         # sweeps the alphabet and tests for each letter
318         for x in range(len(alphabet)):
319             if letter == alphabet[(x + shift_list[(x +
    offset) % len(alphabet)]) % len(alphabet)]:
320                 letter = alphabet[x]
321                 return letter
322             else:
323                 pass
324         else:
325             "This should never happen"
326 
327 
328 def shift_second_rotor(return_path, index, offset):
329     """
330     represents the letter shift of the second rotor and
    returns the corresponding letter
331     :param offset: int: represents the physical rotation
    of the rotor
332     :param return_path: boolean: True if it's the return
    path (after the reflector) or False if it's the forward
    path
333     :param index: the index of the actual letter
334     :return letter: letter
335     """
336     # In this case the rotor's shift is the one of roller
     'II',
337     # for more information see 'https://de.wikipedia.org/
    wiki/Enigma_(Maschine)'
338     alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
339     rotor2 = 'AJDKSIRUXBLHWTMCQGZNPYFVOE'
340     # shift_list calculates the shift of each letter
    between the alphabet and rotor2
341     shift_list = [(ord(rotor2[i]) - ord(alphabet[i])) %
    len(alphabet) for i in range(len(alphabet))]
342     letter = alphabet[index]
343     if return_path is False:
344         # assigns a new letter given the shift and the
    offset
345         letter = alphabet[(index + shift_list[(index +
```

```
345 offset) % len(alphabet)]) % len(alphabet)]
346         if letter.isalpha():
347             return letter
348         else:
349             print("This should never happen")
350             return -1
351     elif return_path:
352         # sweeps the alphabet and tests for each letter
353         for x in range(len(alphabet)):
354             if letter == alphabet[(x + shift_list[(x +
    offset) % len(alphabet)]) % len(alphabet)]:
355                 letter = alphabet[x]
356                 return letter
357             else:
358                 pass
359         else:
360             "This should never happen"
361
362
363 def shift_third_rotor(return_path, index, offset):
364     """
365     represents the letter shift of the third rotor and
    returns the corresponding letter
366     :param offset: int: represents the physical rotation
    of the rotor
367     :param return_path: boolean: True if it's the return
    path (after the reflector) or False if it's the forward
    path
368     :param index: the index of the actual letter
369     :return letter: letter
370     """
371     # In this case the rotor's shift is the one of roller
     'III',
372     # for more information see 'https://en.wikipedia.org/
    wiki/Enigma_rotor_details'
373     alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
374     rotor3 = 'BDFHJLCPRTXVZNYEIWGAKMUSQO'
375     # shift_list calculates the shift of each letter
    between the alphabet and rotor3
376     shift_list = [(ord(rotor3[i]) - ord(alphabet[i])) %
    len(alphabet) for i in range(len(alphabet))]
377     letter = alphabet[index]
378     if return_path is False:
379         # assigns a new letter given the shift and the
    offset
```

```
380         letter = alphabet[(index + shift_list[(index +
    offset) % len(alphabet)]) % len(alphabet)]
381         if letter.isalpha():
382             return letter
383         else:
384             print("This should never happen")
385             return -1
386     elif return_path:
387         # sweeps the alphabet and tests for each letter
388         for x in range(len(alphabet)):
389             if letter == alphabet[(x + shift_list[(x +
    offset) % len(alphabet)]) % len(alphabet)]:
390                 letter = alphabet[x]
391                 return letter
392             else:
393                 pass
394     else:
395         "This should never happen"
396
397
398 # *** EN/DECRYPTION *** #
399 def enigma(text, key):
400     """
401     encrypts or decrypts and returns the given text by
    using the principle of the Enigma machine
402     :param text: user's text (string)
403     :param key: user's key (string composed of three
    letter)
404     :return text: encrypted text
405     """
406     initial_list = create_initial_list()
407
408     # set the rotors according to the key
409     offset_first_rotor = search_index(initial_list, key[0
    ])
410     offset_second_rotor = search_index(initial_list, key[
    1])
411     offset_third_rotor = search_index(initial_list, key[2
    ])
412
413     encrypted_text = ""
414     index_of_letter = 0
415     return_path = False
416
417     for letter in text:
```

```
418         # ** forward path **
419         return_path = False
420         # plugboard
421         letter = plugboard(letter)
422 
423         # determine index
424         index_of_letter = search_index(initial_list,
    letter)
425         # first rotor
426         letter = shift_first_rotor(return_path,
    index_of_letter, offset_first_rotor)
427 
428         # determine index
429         index_of_letter = search_index(initial_list,
    letter)
430         # second rotor
431         letter = shift_second_rotor(return_path,
    index_of_letter, offset_second_rotor)
432 
433         # determine index
434         index_of_letter = search_index(initial_list,
    letter)
435         # third rotor
436         letter = shift_third_rotor(return_path,
    index_of_letter, offset_third_rotor)
437 
438         # permutation reflector
439         letter = permutation_reflector(letter)
440 
441         # ** return path **
442         return_path = True
443         # determine index
444         index_of_letter = search_index(initial_list,
    letter)
445         # third rotor
446         letter = shift_third_rotor(return_path,
    index_of_letter, offset_third_rotor)
447 
448         # determine index
449         index_of_letter = search_index(initial_list,
    letter)
450         # second rotor
451         letter = shift_second_rotor(return_path,
    index_of_letter, offset_second_rotor)
452 
```

```
453         # determine index
454         index_of_letter = search_index(initial_list,
    letter)
455         # first rotor
456         letter = shift_first_rotor(return_path,
    index_of_letter, offset_first_rotor)
457
458         # plugboard
459         letter = plugboard(letter)
460
461         # add encrypted letter to the encrypted text
462         encrypted_text = encrypted_text + letter
463
464         # change offset variables
465         offset_first_rotor += 1
466         if offset_first_rotor == len(initial_list):
467             offset_first_rotor = 0
468             offset_second_rotor += 1
469             if offset_second_rotor == len(initial_list):
470                 offset_second_rotor = 0
471                 offset_third_rotor += 1
472                 if offset_third_rotor == len(initial_list
    ):
473                     offset_third_rotor = 0
474                 else:
475                     pass
476             else:
477                 pass
478         else:
479             pass
480
481     return encrypted_text
    # ********************************* end ENIGMA
     ********************************* #
482
```