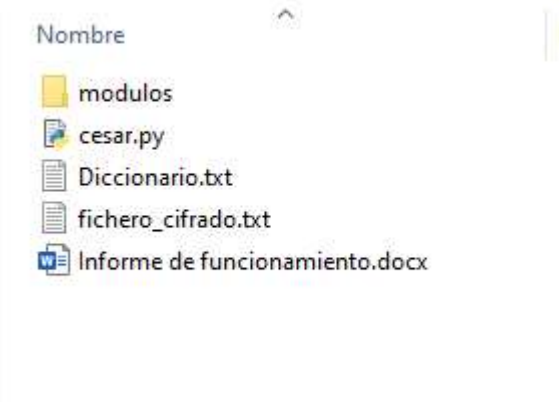
Python 3 – Informe de funcionamiento "Hackeando, a lo bruto, a Julio César"

Autor: Antonio Ramírez Martín

1. Preparación y funcionamiento.

En el archivo *TF_cesar.rar* se encuentra todo lo necesario para ejecutar el programa. Una vez descomprimido nos quedaría lo siguiente:



- Carpeta *modulos.* Contiene los archivos *decodificador.py* y *diccionario.py.*

  El módulo diccionario.py lo utiliza el programa principal para generar un diccionario llamado *Diccionario_personalizado.txt*, con palabras mayores de dos letras, sin tildes, diéresis y con las palabras en mayúsculas.

  El módulo decodificador.py se utiliza para generar todas las claves de crackeo que se guardarán en *fichero_plano.txt* y para la comparación entre el *Diccionario_personalizado.txt* y el *fichero_plano.txt* y así obtener la clave correcta.

- *cesar.py.* Archivo principal. Es el archivo que usaremos para ejecutar el programa y el que usará los módulos.
- *Diccionario.txt.* Diccionario proporcionado por el equipo docente a partir del cual generaremos nuestro diccionario.
- *fichero_cifrado.* Archivo proporcionado por el equipo docente que debemos descifrar.

Importante: El programa está realizado y probado para que funcione en Windows.

Para la puesta en marcha del programa ejecutaremos el fichero "*cesar.py*". Una vez iniciado, el programa realizará las siguientes acciones:

- Lectura de *Diccionario.txt* y creación de *Diccionario_personalizado.txt*
- Lectura de *fichero_cifrado*, obtención de todas las claves Cesar posibles y posterior guardado en *fichero_plano.txt*.
- Búsqueda de la clave Cesar correcta comparando las claves generadas de *fichero_plano.txt* con las palabras de *Diccionario_personalizado.txt* y posterior adicción a *fichero_plano.txt* de la clave con más coincidencias.
- Impresión en pantalla de todas las claves creadas y la identificación de la clave que creemos correcta.

Captura de pantalla del resultado final

Como puede verse, la clave válida es la número 13 que corresponde con la frase:

ALICIA EMPEZABA YA A CANSARSE DE ESTAR SENTADA CON SU HERMANA A LA ORILLA DEL RIO SIN TENER NADA QUE HACER HABIA ECHADO UN PAR DE OJEADAS AL LIBRO QUE SU HERMANA ESTABA LEYENDO PERO NO TENIA DIBUJOS NI DIALOGOS Y DE QUE SIRVE UN LIBRO SIN DIBUJOS NI DIALOGOS SE PREGUNTABA ALICIA

2. Notas respecto al código no incluidas en los comentarios.

Para la implementación de las ruedas de cifrado he usado la pista que se nos da para hacer el trabajo final (*carácter_cifrado = (carácter_en_claro + clave) % 26*) como puede apreciarse en la línea 25 de la siguiente imagen:

```
18  #Decodifica el cifrado usando las 27 claves y las añade al archivo "fichero_plano.txt"
19  def decodificar_claves(cadena_cifrada):
20      archivo_fichero_plano = open('fichero_plano.txt', 'w')
21
22      for clave in range(0,26):
23          archivo_fichero_plano.write(str(clave) + " - ")
24          for caracter_cifrado in cadena_cifrada:
25              archivo_fichero_plano.write(chr(65 + ((ord(caracter_cifrado) - clave + 13) % 26)))
26          archivo_fichero_plano.write("\n")
27
28      archivo_fichero_plano.close()
29
```

La corrección *+13* la hago para que a la hora de calcular el módulo de 65 este sea 0 y hacer que coincida con la letra A.

Como se nos advirtió, tanto para la creación del *Diccionario_personalizado.txt* como a la hora de identificar palabras he tenido en cuenta el carácter de "salto de página" que se da en Windows, por lo que es probable que en otros sistemas operativos el programa no funcione correctamente.

Eso también provoca que la última palabra del diccionario la contemos como si tuviera un carácter menos y el programa termina por no agregarla al *Diccionario_personalizado.txt* dadas las condiciones aplicadas, palabras mayores de dos caracteres.

Para evitar la creación del *Diccionario_personalizado.txt* y el consecuente gasto de recursos tras cada uso he importado la función isfile, que te informa si existe un archivo en un directorio dado.

```
9
10  #Condición para evitar la creación del diccionario personalizado tras cada uso.
11  if isfile("Diccionario_personalizado.txt") == False:
12      diccionario.crear_diccionario_personalizado()
13
```

3. Posibles/Futuras mejoras.

- Elección entre introducción de la cadena a descifrar por teclado o por fichero de texto.

```
13
14  decodificador.decodificar_claves(decodificador.leer_archivo_cifrado())
15  decodificador.buscar_clave_usada()
16  decodificador.mostrar_claves_consola()
17
```

Debe ser fácil, ya que la función *decodificador.leer_archivo_cifrado()* puede sustituirse por una cadena de texto o una variable que contenga una cadena de texto y el programa funcionaría correctamente.

- He abusado de leer y escribir archivos por facilidad cuando lo óptimo habría sido el tenerlos cargados en memoria durante la ejecución del programa.
- Conseguir que la aplicación separe las palabras de la frase.