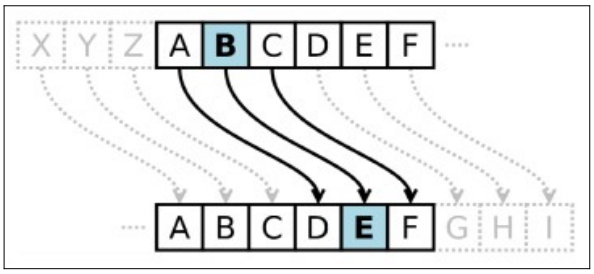
Hackeando, a lo bruto, a Julio César

Introducción:

Para comunicarse con sus generales, Julio César ideó una técnica sencilla de cifrado que consiste en sustituir cada letra del texto original por otra que se encuentra un número fijo determinado de posiciones delante en el alfabeto. Si se llega al final del alfabeto se sigue contando desde el principio. Así, en la siguiente figura podemos ver cuáles son las sustituciones cuando fijamos tres posiciones por delante.

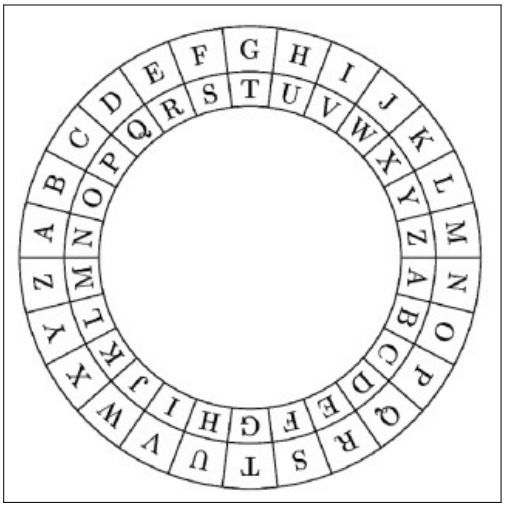


Vamos a introducir un poco de jerga técnica:

- La clave es el número de posiciones hacia delante que vamos a utilizar para cifrar cada carácter. Es un número fijo que hemos de determinar antes de realizar el cifrado.
- El texto sin cifrar se denomina texto en claro o texto plano.
- El texto cifrado se denomina simplemente así, texto cifrado.

Por motivos obvios, Julio no solía enviar la clave con el mismo mensajero a la vez que enviaba el texto cifrado... Si conocemos la clave, el texto en claro se recupera desplazando cada carácter del texto cifrado ese mismo número de posiciones, pero esta vez hacia atrás.

Todo esto es más fácil de ver si imaginamos dos ruedas concéntricas que pueden girar alrededor de su centro común. Inicialmente se colocan de manera que coincidan los caracteres. A continuación, la rueda interior se desplaza hacia delante tantas posiciones como indique la clave, y ya estamos en condiciones de cifrar nuestro texto en claro, carácter a carácter, sin volver a mover las ruedas. Para descifrar hacemos lo mismo, pero movemos la rueda interior hacia atrás. Si tiene impresora quizás se anime a imprimir las ruedas para trabajar y preparar este ejercicio con ellas.



Observemos que la rueda no contiene el carácter Ñ. Volveremos sobre ello en un ratito.

Por simplicidad, vamos a considerar que el texto en claro tiene sólo letras mayúsculas (nada de signos de puntuación), y tampoco vamos a utilizar tildes. Supongamos que queremos cifrar “Mi mamá me mima mucho”. Con lo que acabamos de decir, más bien sería “MI MAMA ME MIMA MUCHO”. Veamos qué sucede si ciframos con las claves 1, 2, 3 y 20; naturalmente, cada cifrado es independiente del anterior, es decir, en cada caso partimos del texto en claro:

```
      MI MAMA ME MIMA MUCHO
1     NJ NBNB NF NJNB NVDIP
2     OK OCOC OG OKOC OWEJQ
3     PL PDPD PH PLPD PXFKR
20    GC GUGU GY GCGU GOWBI
```

Y si tenemos un texto cifrado y sabemos que la clave utilizada ha sido 7, fácilmente podemos recuperar el texto en claro, simplemente desplazando hacia atrás (compruébelo, es sencillo):

```
7     FV TPTV H TP THTH
      YO MIMO A MI MAMA
```

Volviendo a lo de antes, ¿por qué nos olvidamos de la Ñ? La culpa la tiene el cómo representan los computadores las letras: el código ASCII (que podemos considerar como un “subconjunto” de Unicode). Sin entrar en muchos detalles, nuestro computador representa internamente los caracteres como números enteros. Para nuestros fines, nos bastará saberque el carácter A tiene el código 65, B tiene 66, y así hasta Z con 90; las minúsculas tienen otros códigos superiores a éstos y la Ñ queda fuera, relegada al código ASCII extendido (que además depende de la configuración regional que tengamos escogida en nuestro sistema). El espacio en blanco también lo vamos a dejar fuera por lo que vamos a ver enseguida, así que de la A a la Z tenemos sólo 26 caracteres.

En nuestra codificación, para evitar confusiones con palabras como “caña” y “cana” (u otras que suenan peor...) sustituiremos el carácter Ñ por dos N’s consecutivas: Ñ → NN. Por ejemplo:

```
      Año de nieves, año de bienes
      ANNO DE NIEVES ANNO DE BIENES
      ANNODENIEVESANNODEBIENES
10    KXXYNOXSOFOCKXXYNOLSOXOC
```

Y para despistar todavía un poco más (pero poco...), agruparemos los caracteres del texto cifrado de cinco en cinco, dejando un espacio en blanco entre ellos:

```
KXXYNOXSOFOCKXXYNOLSOXOC
KXXYN OXSOF OCKXX YNOLS OXOC
```

Así que finalmente la cadena “Año de nieves, año de bienes” quedará cifrada como “KXXYN OXSOF OCKXX YNOLS OXOC” si utilizamos 10 como clave (las comillas “” no forman parte del mensaje en sí).

DISEÑA UN MÓDULO CON LAS SIGUIENTES FUNCIONES:

Estas funciones pueden utilizar otras funciones que le faciliten el trabajo y que aumenten la legibilidad del programa.

- def texto_plano(fichero_texto_plano: str) → None

  Le pide al usuario que escriba un texto que se guardará en un fichero cuyo nombre se pasa como parámetro.

- def cifrar(fichero_texto_plano: str, fichero_cifrado_cesar: str) → None

  Debe leer un fichero ASCII con el texto plano, que se pasa como parámetro, y lo debe cifrar según una clave aleatoria de 1 a 25. De esta forma no se sabe la clave. El texto cifrado se guarda en el fichero indicado como parámetro.

  Recuerda que se debe preprocesar el fichero de texto plano:

  - Solo caracteres de la A a la Z.

  - La Ñ → NN

  - Nada de tildes (Á → A) , diéresis (Ü → U) ni signos de puntuación.

  - Sólo mayúsculas

  Recuerda que el texto cifrado se agrupa de cinco en cinco caracteres con un blanco entre cada grupo.

- def descifrar_fuerza_bruta(fichero_cifrado_cesar: str , fichero_resultados: str) → None

  Abre el fichero con el texto cifrado y saca los textos correspondientes a intentos de descifrar con las claves de 0 a 26 (en realidad solo hace falta de 1 a 25). Los distintos intentos se guardan en distintos ficheros. Por ejemplo, si fichero_resultados 'test_resultado.txt' se generan para cada clave un fichero: test_resultado1.txt, test_resultado2.txt, ...

- def descifrar_fuerza_bruta_dic(fichero_cifrado: str , fichero_resultado:str, diccionario: list, porcentaje=0.25, rapido=False, minPalabra=None, maxPalabra=None) → None

  Abre el fichero con el texto cifrado y saca los textos correspondientes a intentos de descifrar con las claves de 0 a 26 (en realidad solo hace falta de 1 a 25). Por cada texto, indica el número de subcadenas que coinciden con entradas del diccionario.
  fichero_cifrado_cesar: str Fichero con el texto cifrado.
  fichero_resultado: str Fichero con los resultados (texto y coincidencias) de aplicar todas las claves.
  diccionario: list lista de palabras en castellano.
  porcentaje: int Aquellas claves que generen un porcentaje (en tanto por 1) de coincidencias cercano a la mejor (porcentaje 1) tal que 1 - porcentaje <= coincidencias <= 1 serán consideradas como solución alternativa.
  rapido: boolean Si es False mira todas las posibles subcadenas. Si es True pide la longitud mínima y máxima de las cadenas a considerar. De esta forma, si se hace con criterio, se acelera bastante la búsqueda en el diccionario.
  minPalabra: int Tamaño mínimo de palabra a considerar.

maxPalabra: int Tamaño máximo de palabra a considerar.
Retorna por la salida estándar todos los textos precedidos de la clave aplicada en cada caso. Como side-effect crea un fichero con los resultados. Si hay algún tipo de error informa de ello.

PISTAS para descubrir la clave

- Abrir el fichero cifrado y probar todas las claves posibles desde 1 a 25 (o desde -1 a -25 si lo prefiere). Por ejemplo, para la cadena cifrada KXXYN OXSOFOCKXX YNOLS OXOC su programa deberá presentar en consola una salida parecida a esta:

  Bucle de crackeo:

  ```
  1  JWWXMNWRNENBJWWXMNKRNWNB
  2  IVVWLMVQMDMAIVVWLMJQMVMA
  3  HUUVKLUPLCLZHUUVKLIPLULZ
  4  GTTUJKTOKBKYGTTUJKHOKTKY
  5  FSSTIJSNJAJXFSSTIJGNJSJX
  6  ERRSHIRMIZIWERRSHIFMIRIW
  7  DQQRGHQLHYHVDQQRGHELHQHV
  8  CPPQFGPKGXGUCPPQFGDKGPGU
  9  BOOPEFOJFWFTBOOPEFCJFOFT
  10 AÑODENIEVESAÑODEBIENES
  11 ZMMNCDMHDUDRZMMNCDAHDMDR
  12 YLLMBCLGCTCQYLLMBCZGCLCQ
  13 XKKLABKFBSBPXKKLABYFBKBP
  14 WJJKZAJEARAOWJJKZAXEAJAO
  15 VIIJYZIDZQZNVIIJYZWDZIZN
  16 UHHIXYHCYPYMUHHIXYVCYHYM
  17 TGGHWXGBXOXLTGGHWXUBXGXL
  18 SFFGVWFAWNWKSFFGVWTAWFWK
  19 REEFUVEZVMVJREEFUVSZVEVJ
  20 QDDETUDYULUIQDDETURYUDUI
  21 PCCDSTCXTKTHPCCDSTQXTCTH
  22 OBBCRSBWSJSGOBBCRSPWSBSG
  23 NAABQRAVRIRFNAABQROVRARF
  24 MZZAPQZUQHQEMZZAPQNUQZQE
  25 LYYZOPYTPGPDLYYZOPMTPYPD
  ```

NOTA: las claves 0 y 26 dan lo mismo que el texto cifrado

En este caso particular, observe que la cadena que contiene el texto en claro es más corta que los demás, pero esto no debe confundirle, ya que ello es debido simplemente a que hemos vuelto a sustituir NN por Ñ; en general éste no será el caso y todas las cadenas serán de la misma longitud.

Probar las claves 0 (esto es, “no hago nada”) y 26 (“doy la vuelta completa”) deja el texto cifrado tal y como estaba, simplemente le quita los espacios en blanco. Realmente no hace falta que pruebe estas claves, pero si lo hace le servirá para comprobar si su programa funciona bien.

- Tenga en cuenta todas las explicaciones que hemos dado sobre cómo se ha cifrado el texto. En su programa deberá hacer básicamente lo contrario…
- Repase los métodos específicos para el tratamiento de cadenas de caracteres. Hay mucho trabajo ya hecho por adelantado.
- Para saber el código ASCII de un carácter puede utilizar la función ord(). Así, ord('A') devuelve 65.
- Para obtener el carácter a partir del código ASCII puede utilizar chr(). Así, chr(65) devolverá 'A'.
- ¿Existe una forma fácil de implementar las ruedas de cifrado? Sí. Observe la siguiente figura. Supongamos que hemos escogido la clave 3 y tenemos que cifrar el carácter ‘Y’, cuyo código ASCII es 89 (en rojo). Debe convertirse en el carácter ‘B’ (recuerde, al llegar al final comenzamos nuevamente por el principio), de código 66 (en verde). Para hacer la conversión en Python podríamos utilizar una serie de condicionales, lo que no es práctico ni elegante. Pero supongamos que utilizamos la numeración alternativa que se propone en la fila inferior de la imagen. Entonces carácter_cifrado = (carácter_en_claro + clave) % 26, o sea, (24 + 3) % 26 = 1, donde % es el operador módulo que hemos estudiado. Convertir todo esto nuevamente a códigos ASCII es sencillo.

Carácter	**A**	**B**	**C**	**D**	…	**X**	**Y**	**Z**
C. ASCII	65	66	67	68	…	88	89	90
C. altern.	0	1	2	3	…	23	24	25

Relación de caracteres (fila superior) y sus respectivos códigos ASCII (fila central). En la fila inferior se propone una numeración alternativa, de 0 a 25. La explicación de los colores se da en el texto.

Naturalmente, para este ejercicio no se le permite el uso de bibliotecas de terceros que no sean las que estrictamente acompañan a una instalación “normal” por defecto de Python 3.x. Están especialmente prohibidas las bibliotecas de criptografía…

En Internet puede obtener ficheros con la relación de las palabras del castellano. En particular, junto a este enunciado en el curso virtual le hemos dejado el fichero Diccionario.txt. quecontiene una relación de más de ochenta mil palabras en castellano, compilada por Javier Arce (puede haber derechos reservados).

Con ayuda de otro programa en Python, que puede realizar fácilmente, quite tildes y diéresis, pase a mayúsculas y convierta Ñ en NN. Guarde el resultado, pues lo va a necesitar.

En su programa principal de “crackeo”, para cada una de las posibles frases que va obteniendo, intente contabilizar cuántas subcadenas se encuentran en este “diccionario”. La clave que dé un mayor número de coincidencias será, con mucha probabilidad, la clave que hemos utilizado.

Es poco probable que con dos claves distintas obtenga resultados parecidos, pero si se diera el caso, indíquelo en la salida del programa. Puede ser interesante que en el “diccionario” castellano que le hemos facilitado elimine las palabras de uno, dos e incluso tres caracteres de

longitud. Le ayudará a descartar “falsos” positivos, a la vez que reducirá (sólo un poco) el número de palabras a comprobar. Le dejamos que piense –opcionalmente– cómo hacerlo…

Sobre el salto de línea en los ficheros de texto:

La representación del salto de línea en los ficheros de texto puro difiere de un sistema operativo a otro:

- Windows (y MS DOS) añade al final de cada línea los caracteres ASCII no imprimibles 0D 0A (hexadecimales) o, de manera equivalente, 13 10 (decimales). (Es posible, como excepción, que al final del fichero no los haya si no se termina con un salto de línea.) Ello viene de la época de los teletipos, en que había que señalizar de manera expresa e independiente el salto de línea y el retorno del carro.
- Los sistemas derivados de Unix, como Linux o macOS, utilizan en su lugar un único carácter, 0A.

El fichero Diccionario.txt que les hemos facilitado utiliza la representación de Windows.

Ello no reviste mucha importancia salvo que, por ejemplo, quiera contar la longitud de cada palabra que ha leído desde ese fichero con su programa en Python. En ese caso, es posible que Python le diga que la palabra "sol" tiene cuatro caracteres de longitud. ¿Cuatro? Bueno, si ese es su caso es que Python está contando el carácter OD como parte de la palabra. Si eso le causara problemas las posibles soluciones son varias:

- A la hora de contar, descuente siempre uno.
- Abra el fichero en modo binario, lea carácter a carácter y copie todo en un nuevo fichero, salvo el carácter 0D, que directamente lo tira a la basura.
- ...