# PYTHON 3

## Trabajo final obligatorio

2019 – 2020

| Dra. Carolina Mañoso Hierro — Dr. Ángel Pérez de Madrid y Pablo — Dr. Miguel Romero Hortelano

Como se indica en la Guía, está prevista la realización de un trabajo final obligatorio como parte del sistema de evaluación del curso. A continuación se proponen tres ejercicios. El estudiante deberá realizar y remitir en el plazo previsto sólo uno de ellos. Los dos primeros consisten en la elaboración de un programa en Python 3 en el que el estudiante demuestre un dominio básico de ese lenguaje; en el tercero el estudiante podrá realizar un programa de su elección.

> ***Importante:***
>
> • *Si no ha programado nunca en algún otro lenguaje y le ha costado seguir el curso, le recomendamos que realice el primer ejercicio.*
>
> • *Si ya ha programado en otros lenguajes o el curso le ha parecido de una dificultad razonable, entonces es mejor que realice el segundo o el tercer ejercicio.*

Al inicio del código del programa éste deberá incluir, como comentarios, el nombre del estudiante, la modalidad del ejercicio y la relación de las bibliotecas "extra" utilizadas (por ejemplo, *Pygame*) y su versión, si las hubiera. El código deberá documentarse, es decir, deberá incluir, en su caso, comentarios explicativos en la declaración de funciones, de clases… así como una explicación básica de la estructura del programa.

Para que pueda darse por bueno el programa deberá realizar correctamente la función prevista y deberá poder ejecutarse sin que se produzcan mensajes de error.

> ***Un programa que no funciona o que no hace correctamente lo que se espera de él es un programa que está mal.***

El estudiante deberá además redactar un pequeño informe en el que indicará el trabajo escogido, las ideas principales sobre cómo ha resuelto el problema planteado y todo aquello que considere necesario para poder ejecutar el programa. En particular se detallarán las bibliotecas "extra" utilizadas (si estuvieran permitidas), su versión e instrucciones detalladas para su instalación.

> *En el caso de utilizar bibliotecas u otros elementos adicionales, más allá de los que acompañan por defecto a Python 3, es muy importante que en el informe se den instrucciones detalladas de qué hace falta y cómo se instala.* ***Un programa que no se puede instalar correctamente es un programa que no va a funcionar y por tanto está mal.***

Todo ello, programa e informe, se subirá a la plataforma como un único archivo comprimido, preferentemente en formato RAR o ZIP.

Los trabajos deberán ser completamente originales y presentarse de manera individual.

> ***Si se comprueba que un trabajo no es original (por ejemplo, se ha descargado de Internet y se ha "maquillado" un poco) el estudiante obtendrá la mínima calificación (que es cero), no superando el curso.***

Para todo lo no indicado expresamente en estos enunciados, el estudiante puede hacer las suposiciones razonables que considere convenientes, debiendo en este caso indicarlas en el informe.

*Para solucionar cualquier problema que pueda surgirle a la hora de realizar este trabajo no dude en consultar al equipo docente.*

**Propuesta 1: "A la carta más alta"**

La composición de la baraja española es bien conocida: cuatro palos (oros, copas, espadas y bastos) y diez cartas en cada palo (de menor a mayor valor dos, tres, cuatro, cinco, seis, siete, sota, caballo, rey y as), lo que hace un total de cuarenta cartas.

En el juego un jugador "humano" jugará contra el ordenador cinco rondas, de manera que ganará el juego aquél que gane más rondas.

En cada ronda se generan dos cartas aleatorias, una para el humano y otra para el ordenador (en ese orden). Gana la ronda la carta de valor más alto, sin importar el palo; si ambas cartas tienen el mismo valor (ejemplo: Siete de Copas y Siete de Espadas) será un empate.

Aunque sea poco probable que se dé la siguiente situación, en cada ronda hay que comprobar que las dos cartas obtenidas sean distintas; si ambas son iguales antes de mostrar la segunda se obtendrá una nueva que la sustituirá, y así tantas veces como sea necesario. (Ejemplo: humano y ordenador no pueden sacar a la vez el Siete de Copas.)

En cada ronda se imprimirá en pantalla algo así como "Ronda 3: Tu carta es el As de Bastos y mi carta es el Dos de Oros; ganas tú." Entre ronda y ronda se nos pedirá que pulsemos una tecla antes de continuar, lo que se entenderá como que "se está barajando nuevamente".

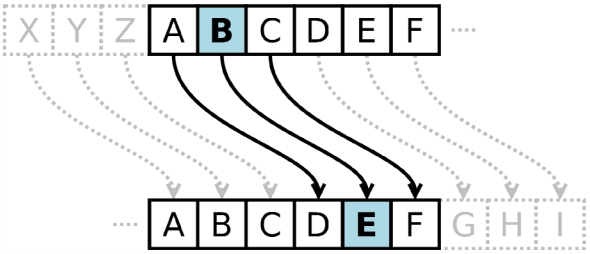
Al finalizar la quinta ronda se mostrará en pantalla quién ha ganado (junto con la puntuación obtenida) o si ha habido un empate.

*Nota:* El "núcleo" de este programa hace poco más que sacar cuatro números aleatorios (dos para los palos y dos para el valor de las cartas). El resto son unas sencillas comparaciones y algún bucle. Realmente no hace falta "barajar" ni "gestionar" el mazo de cartas. Es un ejercicio sencillo que se puede hacer en una tarde.

**Propuesta 2: "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.




Ejemplo de cifrado de Julio César: cada letra del texto original se sustituye por otra que se encuentra tres posiciones delante en el alfabeto. (Fuente: [1])

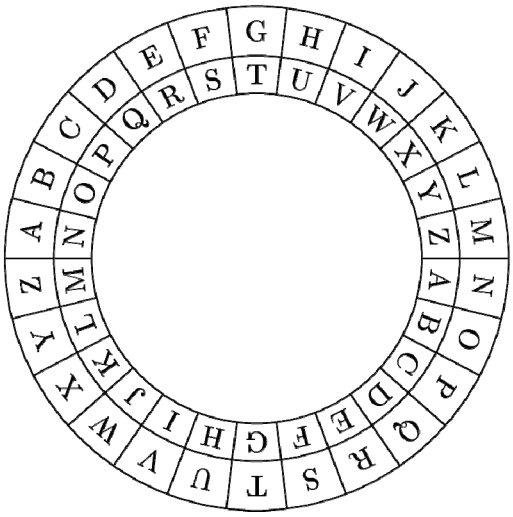
Vamos a introducir un poco de jerga técnica:

- 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.



Ruedas de cifrado. (Fuente [2])

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 saber que 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. (Para saber más puede consultar [3].)

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).

*El ejercicio propuesto:*

Le vamos a pedir que realice un programa que descifre el cifrado de Julio César, tal como lo hemos descrito, por fuerza bruta, es decir, probando todas las claves posibles. Hacerlo a mano es pesado y propenso a errores, pero en Python es coser y cantar. Recuerde:

- Sólo los caracteres de la A a la Z.
- La Ñ la hemos sustituido por NN.
- Nada de tildes ni signos de puntuación.
- Sólo mayúsculas.
- El texto cifrado, agrupado de cinco en cinco caracteres con un blanco entre cada grupo.

El texto cifrado se lo daremos en un fichero de texto, de nombre `fichero_cifrado.txt` (en realidad, lo único cifrado es su contenido), que deberá copiar en la misma carpeta en la que se encuentre su programa descifrador.

Abra dicho fichero y pruebe todas las claves posibles, de 1 a 25 (o de –1 a –25, si lo prefiere). Por ejemplo, para la cadena cifrada KXXYN OXSOF OCKXX YNOLS OXOC su programa deberá presentar en consola una salida parecida a esta:

```
Bucle de crackeo:
 0 KXXYNOXSOFOCKXXYNOLSOXOC
 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
26 KXXYNOXSOFOCKXXYNOLSOXOC
```

A la vista –a ojo– de esta salida, en su informe deberá decirnos, entre otras explicaciones, "El texto en claro es *año de nieves año de bienes* y la clave utilizada ha sido *10*". 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 las pruebe, pero si lo hace le servirá para comprobar si su programa funciona bien.

*Pistas:*

- 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…

Si se anima, hacer también el programa de cifrado (no se lo pedimos) puede ayudarle a comprender cómo se debe descifrar el texto. Por ejemplo, puede comprobar si es capaz de cifrar correctamente los ejemplos que hemos ido describiendo, y de aquí darle la vuelta al programa de cifrado para que en su lugar descifre.

*Cosas que debe entregarnos:*

Simplemente su programa descifrador y el informe. Nosotros probaremos el programa para comprobar que funciona. Éste debe mostrar en la consola, sin errores de funcionamiento, el resultado de probar las claves 1 a 25 (ó 0 a 26). Estos mismos resultados se deberán grabar en el fichero de texto `fichero_plano.txt`, que su programa debe crear. Debe tratarse de un fichero de texto "normal", que cualquier editor como "Notepad" o similar pueda abrir y mostrar correctamente. (O sea, no cree el archivo en modo binario.) ¡No olvide los saltos de línea!

El programa deberá leer necesariamente el fichero `fichero_cifrado.txt` para obtener el texto cifrado; nada de copiarlo a mano y meterlo en el programa.

Como ya hemos indicado, en el informe deberá decirnos cuáles cree que han sido el texto en claro y la clave utilizados.

*Posibles mejoras opcionales:*

Al final, la decisión de cuál han sido texto en claro y clave la debe hacer Vd. "a ojo". Sólo hay 25 posibilidades, luego es fácil hacerlo. (A día de hoy éste es un cifrado de juguete, pero como primer programa "serio" en Python tampoco está mal.)

Le proponemos, como mejora opcional, automatizar también esta tarea. Hágalo sólo si la versión "simple" de este ejercicio le ha resultado de una complejidad razonable.

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`. Lo hemos obtenido en [4] y contiene 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, 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 notablemente el número de palabras a comprobar.

Le dejamos que piense –opcionalmente– cómo hacerlo…

*Referencias:*

[1] *Cifrado César*. Disponible: https://es.wikipedia.org/wiki/Cifrado_C%C3%A9sar

[2] *How to create a Caesar's encryption disk using LaTeX*. Disponible: https://tex.stackexchange.com/questions/103364/how-to-create-a-caesars-encryption-disk-using-latex

[3] *ASCII*. Disponible: https://es.wikipedia.org/wiki/ASCII

[4] *Listado general de palabras en castellano*. Disponible: https://github.com/javierarce/palabras/blob/master/listado-general.txt

**Propuesta 3: Trabajo libre**

El estudiante realizará, en las mismas condiciones que para las propuestas anteriores, un programa en Python 3 sobre un tema que le interese o que pueda serle de utilidad en su vida personal o profesional. Dicho programa debe tener una complejidad igual o superior a la de la *Propuesta 2*. El estudiante deberá pedir autorización previa al equipo docente mediante correo electrónico a angel@scc.uned.es, en el que describirá detalladamente su propuesta de trabajo.