

TEMA 3

Elemento de control (Condicionales y Bucles)



ÍNDICE

Try - except	2
Condicionales	7
Bucle while	13
Bucle for	19
Instrucciones de código usadas	24
Reto	25

Try - except

Ya hemos realizado nuestros primeros programas con Python. Los mismos constaban de un algoritmo secuencial que no alteraba su orden de ejecución, se ejecutaban paso a paso y siempre de la misma forma.

Algoritmo es el nombre que recibe una secuencia ordenada de acciones que es, básicamente, un programa.

En este tema, vamos a ver **elementos de control** para nuestro script, mediante los cuales el algoritmo seguirá un camino u otro en función de comprobaciones y decisiones que realizará Python, siempre respondiendo a nuestra programación.

Antes de lanzarnos a ver cómo se usan esos elementos de control vamos a aprender una instrucción de código bastante interesante. En el tema anterior hemos visto ya que en ocasiones Python no sabe cómo realizar una instrucción y nos arroja un error. En muchas ocasiones el programa puede estar bien realizado pero el usuario no ha sabido introducir un input adecuado. Por ejemplo, si pedimos un número y contesta una letra el programa no va a saber tratar esa letra como número.

Vamos a realizar un pequeño script donde pidamos al usuario dos números y los vamos a sumar. Guarda en un documento .py el siguiente código y ejecútalo por terminal o con la IDLE. Si hay algo que no entiendas del mismo repasa el tema anterior:

```
print("Vamos a sumar dos números")
numero_1 = input("¿Cuál es el primer número?: ")
numero_2 = input("¿Cuál es el segundo número?: ")
numero_1 = int(numero_1)
numero_2 = int(numero_2)
suma = numero_1 + numero_2
suma = str(suma)
print("La suma de ambos números es " + suma)
```

Mucho cuidado si copiamos y pegamos texto para hacer un script, a veces ciertos caracteres se pegan de forma errónea en el archivo de destino, cosa que suele ocurrir con las comillas. Los editores de texto las escriben como comillas iniciales y finales y Python sólo entiende de comillas rectas (es así de especialito).

Al ejecutar el script anterior obtendremos algo como:

```
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 suma.py
Vamos a sumar dos números
¿Cuál es el primer número?: 5
¿Cuál es el segundo número?: 8
La suma de ambos números es 13
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

¿Qué ocurre si elegimos un caracter no numérico?:

```
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 suma.py
Vamos a sumar dos números
¿Cuál es el primer número?: 3
¿Cuál es el segundo número?: a
Traceback (most recent call last):
  File "suma.py", line 5, in <module>
    numero_2 = int(numero_2)
ValueError: invalid literal for int() with base 10: 'a'
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

Como puedes observar Python nos ha dado un error, no puede convertir una **a** en un **int**. Vamos a intentar evitar esos errores.

Python tiene una función que nos permite intentar realizar una instrucción y si su resultado es un error ejecutará otra instrucción que señalemos para tal fin. Hablamos de *try - except*. Le pediremos a Python que pruebe a ejecutar un código (*try*) y si el resultado del código es un error pasará a ejecutar el código contenido en el *except*.

Llegados a este punto conviene hablar de las indentaciones como base de jerarquía en Python. Si no has entendido esta última frase no te preocupes, a lo largo del curso vas a tener claro que son las jerarquías. En algunos lenguajes de programación, para marcar dónde empieza y dónde termina una función, se usan llaves (**{ }**). De esta forma el programa sabe perfectamente dónde termina una instrucción porque lo marca una llave final.

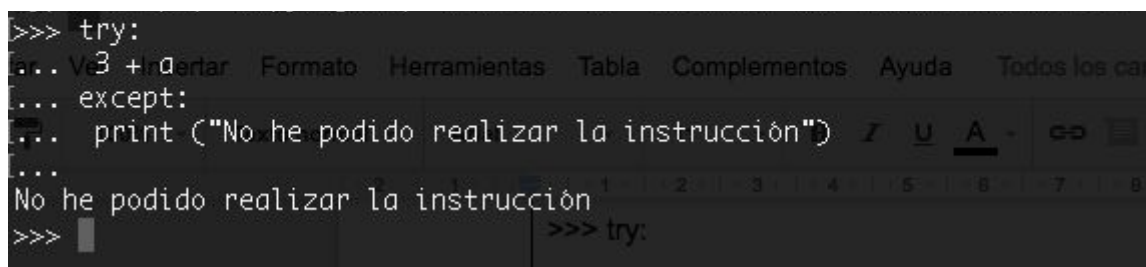
Python es bastante más simple y ello nos puede llevar a cometer errores si no sabemos cómo entenderle (o cómo nos entiende él a nosotros). Cuando una instrucción de código (como puede ser un *try*) contiene un código a ejecutar hay que indentar dicho código respecto de la instrucción.

Voy a escribir un código para que veáis a lo que me refiero:

```
try:
    3 + a
except:
    print("No he podido realizar la instrucción")
```

Si te fijas, el código que contiene el `try` está indentado, es decir, desplazado a la derecha respecto del `try`. Así mismo, el contenido del `except` está desplazado a la derecha. Todas las funciones de Python necesitan que su contenido esté indentado para que sepan qué contienen exactamente y dónde acaba su campo de acción.

Veamos esta instrucción ejecutada por terminal:



```
>>> try:
...     3 + a
... except:
...     print("No he podido realizar la instrucción")
...
No he podido realizar la instrucción
>>>
```

La cantidad de espacios que pongo en el contenido de una función de Python es indiferente, puedo poner un espacio, tres, cuatro... lo importante es que siempre ponga la misma cantidad de espacios. En scripts lo habitual es usar una tabulación. En terminal o por la IDLE es un poco complicado este proceso las primeras veces porque marca con puntos suspensivos (como se puede ver en la imagen superior) que nos encontramos dentro de una instrucción y debemos poner espacios, la tecla de tabulación no funciona. En el ejemplo he puesto un único espacio bajo el `try` y un único espacio bajo el `except`.

Prueba a repetir el código de la imagen anterior y, tras ello, repítelo cambiando la `a` por un `5` para que veas que el código que contiene el `try` si se ejecuta cuando no da error.

Una vez que hemos visto cómo funciona un `try - except` vamos a ver cómo podemos utilizar esta instrucción en nuestro script suma.

```
print("Vamos a sumar dos números")
numero_1 = input("¿Cuál es el primer número?: ")
numero_2 = input("¿Cuál es el segundo número?: ")
try:
    numero_1 = int(numero_1)
except:
    numero_1 = input("No has elegido un número, elige un número: ")
```

```
numero_1 = int(numero_1)
numero_2 = int(numero_2)
suma = numero_1 + numero_2
suma = str(suma)
print("La suma de ambos números es " + suma)
```

El programa está corregido para tratar de convertir el contenido de la variable *numero_1* en entero y, si no puede hacerlo, nos volverá a preguntar por un número y lo convertirá en número entero. Vamos a probarlo:

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 suma.py
Vamos a sumar dos números
¿Cuál es el primer número?: a
¿Cuál es el segundo número?: 3
No has elegido un número, elige un número: 5
La suma de ambos números es 8
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

Si observas la ejecución del código verás que inicialmente seleccioné una **a** y un **3** y el programa me preguntó después por un número ya que el *try* le suponía un error. Al escribir un 5 ya podía hacer la suma.

Antes de seguir, sitúa un *try - except* para la variable *numero_2* como hemos hecho para la variable *numero_1* y prueba el programa.

Si pensamos un poco en el recurso que hemos aprendido podríamos llegar a la conclusión que nos evita un error una única vez, pero en nuestro ejemplo el usuario podría seguir poniendo letras y volvería a ocurrir un error:

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 suma.py
Vamos a sumar dos números
¿Cuál es el primer número?: a
¿Cuál es el segundo número?: 4
No has elegido un número, elige un número: a
Traceback (most recent call last):
  File "suma.py", line 5, in <module>
    numero_1 = int(numero_1)
ValueError: invalid literal for int() with base 10: 'a'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "suma.py", line 8, in <module>
    numero_1 = int(numero_1)
ValueError: invalid literal for int() with base 10: 'a'
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

Como ves, al volver a elegir una **a** tras elegirla en primera opción ocurre un error y encima Python avisa que el error ocurrió durante la excepción, es decir, durante la ejecución de un *try - except*.

Más adelante aprenderemos a evitar un error reiterativo, pero de momento nos podemos conformar con dar una segunda oportunidad de introducir un número al usuario.

Llegados a este punto puedes ver el vídeo referente al *try - except*.

Ahora que ya hemos visto una instrucción que nos va a ser muy útil vamos a pasar a ver elementos de control.

Condicionales

El primer elemento de control que vamos a ver es el condicional *if* (en castellano si). El condicional *if* nos permite, al ejecutarse, realizar una acción sólo si se cumple una o varias condiciones. Por ejemplo: si el número es par haz tal cosa...

Es importante tener claro a partir de ahora la diferencia entre usar `=` y usar `==`. Usar un igual (`=`) significa asignar un valor. La expresión `numero = 3` asigna a la variable `numero` el valor 3. En cambio, usar dos iguales (`==`) significa comparar si dos elementos tienen el mismo valor. La expresión `numero == 3` comprueba si el contenido de la variable `numero` tiene el valor 3.

Una comprobación con `==` en Python nos dará el valor *True* de ser cierta y el valor *False* de no ser cierta, puedes observar la siguiente captura de un código en terminal para entenderlo:

```
[>>> numero = 3 numero = int(numero)
[>>> numero == 3
True
[>>> numero = 4
[>>> numero == 3
False
[>>> if numero%2 == 0:
print("Has elegido un número par")
```

En el primer caso asigno el valor 3 a la variable `numero` y luego compruebo si la variable `numero` coincide con el número 3 y, al ser cierto, el programa nos devuelve el valor *True*. En el segundo caso asigno a la variable `numero` el valor 4 y luego compruebo si `numero` tiene el valor 3, con un resultado de *False* dado que no es cierta esa igualdad.

Ahora que hemos conocido los valores *True* y *False* podemos conocer un nuevo tipo de variable, la **variable booleana**, que sólo tiene dos estados (*True* y *False*):

```
>>> estado = True
>>> type(estado)
<class 'bool'>
```

Este tipo de variables se usan para todo aquello que sólo puede tomar dos estados (que en programación es algo habitual).

Vamos directamente a ver y probar un código que contiene un *if* para entender cómo trabaja:


```
numero = input("Elige un número: ")
try:
    numero = int(numero)
except:
    numero = input("No has elegido un número, elige un número: ")
    numero = int(numero)
if numero % 2 == 0:
    print("Has elegido un número par")
```

Prueba el siguiente código teniendo cuidado con las indentaciones. Guárdalo en un script y ejecútalo para ver qué ocurre cuando eliges un número. No sigas leyendo hasta que no lo hayas probado.

Un *if* requiere comprobar una condición para ejecutarse. En nuestro ejemplo hemos introducido un número y el programa comprueba cual es el resto de la división de ese número entre 2 (*numero % 2*) y lo compara con el valor 0.

Al ser *numero* una variable de tipo entero el resultado de dividirla entre 2 no puede ser otro que 0 o 1. Si queremos que nos diga, así mismo, si el número elegido es impar en caso de ocurrir podemos añadir un segundo *if*:

```
numero = input("Elige un número: ")
try:
    numero = int(numero)
except:
    numero = input("No has elegido un número, elige un número: ")
    numero = int(numero)
if numero % 2 == 0:
    print("Has elegido un número par")
if numero % 2 == 1:
    print("Has elegido un número impar")
```

Aquí tienes un par de pruebas con el código anterior:

```
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 numero_par.py
Elige un número: 3
Has elegido un número impar
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 numero_par.py
Elige un número: 2
Has elegido un número par
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

Como ves, al ejecutar el programa puedo probar números pares e impares y el resultado es el esperado.

El problema es que a veces tenemos un montón de posibilidades a comprobar y no es buena práctica incluir cada una de ellas en un *if* diferente porque Python tendrá que comprobar todas ellas una por una aunque la primera ya haya sido la elegida. En nuestro ejemplo, sin ir más lejos, Python comprueba si el número es impar aunque ya haya certificado que es par.

Para evitar esto, existe una instrucción ampliada del condicional *if* que nos permite ir chequeando diferentes comparativas hasta que una se cumpla. Para cada nuevo valor que quiera chequear y esté vinculado al primer *if* añadiré el código *else if* o *elif*. Esta instrucción toma la siguiente forma:

```
if numero == 0:  
    print("Has elegido el 0")  
elif numero == 1:  
    print("Has elegido el 1")  
elif numero == 2:  
    print("Has elegido el 2")  
...
```

Podríamos seguir poniendo tantas comprobaciones como necesitáramos pero una vez que, al ejecutarse el programa, una sea cierta, el resto no se comprobarán. Esto hace que los programas sean bastante más ágiles y rápidos pues en ocasiones hay que comprobar muchas comparaciones.

Existe una tercera opción además del *if* y *elif*, sólo podemos usarla al final de nuestra lista de chequeos o comprobaciones y se aplica a cualquier otra condición no incluida en las comprobaciones hechas. Dicha opción será *else* a secas y, en caso de existir, su contenido siempre se ejecuta si no se ha cumplido ninguna de las anteriores comprobaciones.

Prueba el siguiente código guardándolo en un script *.py* y elige diferentes números para ver cómo funciona:

```
numero = input("Elige un número del 0 al 2: ")  
if numero == "0":  
    print("Has elegido el 0")  
elif numero == "1":  
    print("Has elegido el 1")  
elif numero == "2":  
    print("Has elegido el 2")  
else:  
    print("El número elegido no está entre el 0 y el 2 o no es un
```

```
número")
```

Como podrás ver, he hecho la comprobación con un número entre comillas. En esta ocasión en vez de cambiar el contenido de la variable *numero* a *int* directamente compruebo en los condicionales con un número como *string*, que es lo que almacena un input.

Aquí tienes el resultado de probarlo con varios números:

```
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 condicional.py
Elige un número del 0 al 2: 1
Has elegido el 1
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 condicional.py
Elige un número del 0 al 2: 2
Has elegido el 2
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 condicional.py
Elige un número del 0 al 2: 3
El número elegido no está entre el 0 y el 3 o no es un número
```

A los condicionales que incluyen varias condiciones se les llama *if - else*.

Ahora que ya conoces el condicional vamos a empezar a hacer un programa funcional. Vamos a crear un script donde se pregunte al usuario por dos números y se le dé varias opciones para seleccionar:

```
numero_1 = input("Elige el primer número: ")
try:
    numero_1 = int(numero_1)
except:
    numero_1 = input("No has elegido un número, elige el primer número: ")
    numero_1 = int(numero_1)
numero_2 = input("Elige el segundo número: ")
try:
    numero_2 = int(numero_2)
except:
    numero_2 = input("No has elegido un número, elige el segundo número: ")
    numero_2 = int(numero_2)
print("Puedes elegir los siguientes números:")
print("1 - Sumar ambos números")
print("2 - Restar ambos números")
seleccion = input("¿Cual es tu selección?: ")
```

Prueba a realizar tú mismo el siguiente ejercicio antes de seguir con el documento. Intenta incluir a continuación del código anterior un *if - else* que sume ambos números si la selección es 1, que reste ambos números si la selección es 2 y que informe de que la selección es incorrecta en caso de no ser 1 ni 2. Es importante que lo intentes antes de ver cómo se haría, si no lo consigues puedes seguir leyendo:

```
numero_1 = input("Elige el primer número: ")
try:
    numero_1 = int(numero_1)
except:
    numero_1 = input("No has elegido un número, elige el primer número: ")
numero_2 = input("Elige el segundo número: ")
try:
    numero_2 = int(numero_2)
except:
    numero_2 = input("No has elegido un número, elige el segundo número: ")
print("Puedes elegir los siguientes números:")
print("1 - Sumar ambos números")
print("2 - Restar ambos números")
seleccion = input("¿Cual es tu selección?: ")
if seleccion == "1":
    print ("Resultado de la suma:")
    print (numero_1 + numero_2)
elif seleccion == "2":
    print ("Resultado de la resta:")
    print (numero_1 - numero_2)
else:
    print ("No has elegido una opción válida")
```

Para que el contenido del *if - else* se muestre impreso en una sola línea (en nuestro ejemplo está en dos líneas) deberíamos concatenar el número resultado de la operación con el *string* "Resultado de la..." y para ello previamente deberíamos guardar el resultado de la operación en una variable y convertirlo a *string*. Sería importante, llegados a este punto, que entiendas de qué estamos hablando y sepas hacerlo (aunque te cueste revisar unas cuantas cosas del tema anterior).

Ahora que ya sabes hacer tu primer script con condicionales intenta añadir dos nuevas opciones para multiplicar los números y dividirlos. Debes incluirlas como opción dada al usuario (que sepa que puede elegir las) y añadir los *if - else* necesarios para que se ejecuten esas opciones si las elige el usuario.

Una vez lo hayas conseguido o si no lo consigues del todo puedes ver el vídeo de condicionales.

Bucle while

Una vez que sabemos cómo alterar la ejecución de un programa en función de una decisión puntual, vamos a ver otro tipo de elemento de control, en este caso uno que se repetirá mientras (*while*) o hasta que se cumpla una condición.

Volvamos a nuestro programa *try - except* inicial:

```
print ("Vamos a sumar dos números")
numero_1 = input("¿Cuál es el primer número?: ")
numero_2 = input("¿Cuál es el segundo número?: ")
try:
    numero_1 = int(numero_1)
except:
    numero_1 = input("No has elegido un número, elige un número: ")
    numero_1 = int(numero_1)
numero_2 = int(numero_2)
suma = numero_1 + numero_2
suma = str(suma)
print("La suma de ambos números es " + suma)
```

En dicho programa faltaba implementar la solución para el número 2, más adelante tendrás que completarlo si no lo hiciste en su momento.

¿Qué tal si forzamos al usuario a que la variable que introduzca en el input *numero_1* sea un número repitiendo una y otra vez la pregunta hasta que se pueda convertir en *int* la respuesta?

Para este tipo de acciones (y muchas otras) se usan los bucles. Vamos a aprender a usar el bucle *while* (mientras). Dicho bucle se repetirá mientras se cumpla una condición, y también vamos a aprender a utilizar las variables como elementos para provocar o dejar de provocar bucles.

Para ello, usaremos variables booleanas (recuerda, las que pueden tomar el valor *True* o *False*).

La forma de entender este proceso es sencilla:

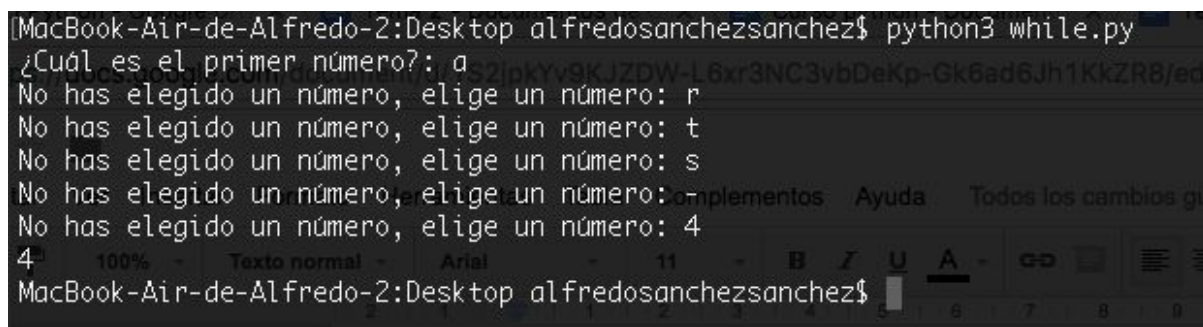
1. Declaramos una variable con el valor *True*.
2. Creamos un bucle *while* que se repita hasta que la variable declarada tenga el valor *False*.

3. Dentro del bucle situamos el código que necesitamos que se repita hasta la solución correcta.

Veámoslo con un código sencillo:

```
repetimos = True
numero_1 = input("¿Cuál es el primer número?: ")
while repetimos == True:
    try:
        numero_1 = int(numero_1)
        repetimos = False
    except:
        numero_1 = input("No has elegido un número, elige un número: ")
print(numero_1)
```

Aquí tienes una captura de la ejecución del mismo:



```
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 while.py
¿Cuál es el primer número?: a
No has elegido un número, elige un número: r
No has elegido un número, elige un número: t
No has elegido un número, elige un número: s
No has elegido un número, elige un número: 4
4
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

Como ves, el usuario no tiene más remedio que poner un número ya que se repite la pregunta **hasta** que así sea o, dicho de otra forma, **mientras** que no sea así.

Lo más importante es entender **cómo usamos una variable para entrar y salir de un bucle** y cómo usamos el bucle para forzar al usuario a elegir un número. El cambio de la variable *repetimos* de *True* a *False* produce que se termine el bucle *while*. Ese cambio no ocurre hasta después de que el programa pueda convertir *numero_1* en entero, y si este proceso da error pasará directamente al *except* y se saltará el cambio de la variable *repetimos* de *True* a *False*.

Intenta, ahora que has visto este programa, repetirlo para seleccionar un segundo número y terminar preguntando al usuario qué quiere hacer con esos números (sumarlos, restarlos, multiplicarlos o dividirlos...). En este punto has llegado a un paso importante que todo programador acaba dando: **reutilizar código**. Utiliza el final del programa del apartado Condicionales para agilizar el proceso y completar el reto lanzado.

Vamos a utilizar el bucle *while* para otra función, aprendiendo a la vez a usar bibliotecas.

Una librería o biblioteca (del inglés *library*) es un código o conjunto de códigos funcionales que podemos utilizar importándolos a nuestro programa. Dicho de otra forma, son programas que nos permiten ser utilizados simplemente cargándolos y utilizándolos enteros o algunas partes del mismo.

Las bibliotecas pueden tener muchos orígenes pero siempre hay información sobre su uso y opciones en internet. Veamos un ejemplo con la biblioteca *time*.

La biblioteca o librería *time* nos permite usar varias funciones, entre ellas establecer tiempos de espera en nuestro programa antes de ejecutar la siguiente línea de código. Vamos a hacer un cronómetro con las siguientes instrucciones:

- El usuario debe seleccionar el tiempo desde el que quiere cronometrar.
- Cuando lleguemos a cero pararemos la cuenta atrás.
- El tiempo disminuirá de segundo en segundo.

Para utilizar una librería debemos usar el siguiente código:

```
import time
```

En ocasiones usaremos una parte de una librería y no entera, lo veremos más adelante. La opción anterior importa a tu programa **toda** la librería *time*.

Ahora vamos a crear un programa sencillo y básico y luego propondré mejoras para que puedas practicar. Observa el siguiente código:

```
import time
tiempo = input("¿Cuánto tiempo tendrá la cuenta atrás?: ")
tiempo = int(tiempo)
while tiempo >= 0:
    print(tiempo)
    tiempo = tiempo - 1
    time.sleep(1)
```

Prueba a guardarlo como script y ejecutarlo, elige un tiempo prudencial (10, por ejemplo) y mira qué ocurre antes de seguir.

Si ya lo has probado deberías haber conseguido algo similar a esto:


```
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 cronometro.py
¿Cuánto tiempo tendrá la cuenta atrás?: 10
10
9
8
7
6
5
4
3
2
1
0
```

Si observas el código verás que hemos usado dos instrucciones nuevas. En primer lugar hemos usado una librería importándola y ejecutando una de sus funciones. Las funciones de las librerías se ejecutan con el nombre de la librería seguido de un punto y el nombre de la función que necesitamos. Algunas funciones de las librerías necesitan, además, parámetros, que irán entre paréntesis.

En nuestro caso la función `time.sleep()` necesita entre paréntesis el tiempo (en segundos) que va a esperar el programa antes de continuar.

Veremos más funciones de las librerías en otros temas, pero de momento podemos quedarnos con esta función tan interesante: ¡esperar una cantidad de tiempo!.

Además, hemos utilizado una nueva forma de comparación en el `while`. No hemos usado un `==` sino un `>=`. Esta opción, ubicada en el código `tiempo >= 0` indica al programa que la comparación será verdadera cuando `tiempo` tenga un valor mayor o igual a cero. Al estar dentro de un `while` ese bucle se repetirá hasta que `tiempo` tenga un valor inferior a 0, de ahí que la cuenta atrás se detenga al llegar a cero.

Veamos un pequeño cambio que puede suponer una gran diferencia para entender el bucle. Efectúa el cambio en el código que verás a continuación y pruébalo como script:

```
import time
tiempo = input("¿Cuánto tiempo tendrá la cuenta atrás?: ")
tiempo = int(tiempo)
while tiempo >= 0:
    tiempo = tiempo - 1
    print(tiempo)
    time.sleep(1)
```

Como puedes observar, el único cambio es situar primero el cambio de la variable *tiempo* y después imprimimos la variable *tiempo*.

El resultado al probarlo debería ser algo similar a ésto (he elegido 5 como *input* para reducir el tiempo de espera):

```
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 cronometro.py
¿Cuánto tiempo tendrá la cuenta atrás?: 5
4
3
2
1
0
-1
```

Ese pequeño cambio resulta en un programa erróneo. Es importante pensar en **cómo las fases del programa se suceden** y qué necesito hacer primero. Si yo quiero hacer una cuenta atrás de 5 segundos, primero tendré que imprimir el 5, después lo convierto en un 4 y tras una espera de un segundo lo imprimo. Podría primero esperar un segundo y después convertirlo en 4 e imprimirlo, pero si primero le quito una unidad y luego lo imprimo mi cinco inicial se convertirá primero en un 4 y luego lo imprimiré, lo cual no es lo que queremos.

Así mismo, el programa termina en un -1. Esto es así porque (y esto es importante) la condición que contiene el *while* se comprueba tras cada ejecución de su contenido **completo**, es decir, el programa realiza todo el contenido del *while* en cada ejecución del mismo antes de comprobar si la condición sigue siendo válida. Para entender esto mejor vamos a poner un ejemplo muy claro:

```
import time
tiempo = input("¿Cuánto tiempo tendrá la cuenta atrás?: ")
tiempo = int(tiempo)
while tiempo >= 0:
    print(tiempo)
    tiempo = tiempo - 1
    time.sleep(1)
print("La variable tiempo ha acabado con un valor de: ")
print(tiempo)
```

Si ejecutamos este código eligiendo como *input* 5 segundos obtendremos lo siguiente:

```
[MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 cronometro.py  
¿Cuánto tiempo tendrá la cuenta atrás?: 5  
5  
4  
3  
2  
1  
0  
La variable tiempo ha acabado con un valor de:  
-1
```

Es importante entender por qué la variable ha acabado con un valor de -1. El programa imprime la variable *tiempo* antes de realizar ninguna otra acción del *while*. En la última ejecución del *while* el programa imprime el 0, después le resta 1 y la variable *tiempo* pasa a tener el valor -1 y posteriormente espera 1 segundo. Tras ello, salimos del *while* al volver a comprobar si *tiempo* es mayor o igual que 0 y, ya fuera del *while*, imprimimos de nuevo el valor de la variable *tiempo*, que tiene el valor final de -1.

Seguramente en el futuro cambiarás muchas cosas de este programa, pero en este punto suficiente para nuestra cuenta atrás. Sólo añadiré que realmente no está pasando un segundo exacto en cada ejecución del *while*, pues el resto de acciones también lleva un tiempo, aunque sea muy pequeño, pero a efectos prácticos tenemos una cuenta atrás en segundos.

El reto, ahora, es que consigas hacer una cuenta atrás que pregunte el tiempo al usuario y lo repita hasta que introduzca un número correcto y que, tras llegar a cero, imprima el valor “Ya hemos terminado”.

Bucle for

Ya tenemos más o menos claro qué es un condicional y qué es un bucle. En el caso de estos últimos tenemos una evolución del ya conocido bucle *while* que nos permite realizar un número concreto de acciones. Dicha evolución se conoce como bucle *for*.

Este bucle es muy usado para repetir acciones cuando conocemos el número de veces que se va a repetir, es decir, para aquellos bucles en los cuales sabemos cuando va a comenzar y cuando va a terminar.

Antes de introducirnos en el bucle vamos a ver un ejemplo con un *while* y después lo trasladaremos a un *for*.

Imagina que tenemos que escribir todas las letras que componen una palabra. Para ello tenemos que ir recorriendo la palabra e imprimiendo la letra que ocupa cada posición de la misma.

Aprendamos algunas cosas nuevas sobre los *string* (en castellano, cadenas). ¿Cómo se nombra la posición que ocupa cada caracter de un *string*? Prueba el siguiente código en tu terminal o IDLE:

```
palabra = "Python"  
print(palabra[1])
```

Analizando el código podemos intuir que imprimiremos la posición 1 del *string* almacenado en la variable *palabra*. Ahora bien, si lo probáis veréis lo siguiente:

```
>>> palabra = "Python"  
>>> print (palabra[1])  
y  
>>> █
```

Resulta que la posición 1 de la palabra Python la ocupa la **y**, cuando lo lógico para un ser humano hubiese sido pensar que la posición 1 sería la **P**. Pues bien, en lenguaje máquina **siempre empezaremos a numerar y contar desde cero**. Esto es muy importante y al principio cuesta un poco pensar en numerar desde cero, pero con la práctica lo naturalizarás.

Si pensamos un poco en cómo contamos los humanos hay cosas que si empezamos a contar desde cero, por ejemplo los años de vida, nuestro primer año de vida es el cero. Hay

otras que empezamos en el uno, por ejemplo las cantidades de productos que tenemos, el primer limón que tenemos en casa es el uno.

Siguiendo con nuestras cadenas, es muy útil saber el número de caracteres que tiene una cadena. Veamos cómo se podría realizar esta acción:

```
palabra = "Python"
print(len(palabra))
```

Sencillamente debemos indicar que queremos conocer la longitud de un string con el comando `len` seguido del nombre de la variable donde está el string.

También se puede utilizar directamente para medir un string no almacenado:

```
print(len("Python"))
```

Una vez que ya sabemos cómo conocer la posición de un caracter en una cadena y cómo medir su longitud vamos a generar un código que deletree una palabra.

Analiza, guarda y prueba el siguiente código para intentar entender cómo funciona:

```
palabra = input("¿Qué palabra quieres deletrear?: ")
longitud = len(palabra)
posicion = 0
while posicion + 1 <= longitud:
    print(palabra[posicion])
    posicion = posicion + 1
```

Aquí tienes el resultado de su ejecución con la palabra "Python":

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 deletreo.py
¿Qué palabra quieres deletrear?: Python
P
y
t
h
o
n
```

Quizá no tengas muy claro por qué funciona el código, es algo habitual al ver código creados y es un momento estupendo para aprender a poner **comentarios**. Los comentarios son textos que ponemos para explicar parte del programa y que Python no interpretará porque sabrá que son aclaraciones para el usuario. Los comentarios en Python se indican con una

almohadilla *#comentario* si son de una línea o con triples comillas si quiero hacer un comentario más largo *""" comentario muy largo """*. Veamos si ahora entiendes bien el código con esta captura de pantalla de un editor:

```
1  """Preguntamos al usuario qué palabra quiere deletrear"""
2  palabra = input ("¿Qué palabra quieres deletrear?: ")
3  """Almacenamos la longitud de la palabra en una variable"""
4  longitud = len (palabra)
5  """Creamos una variable para recorrer la palabra"""
6  posicion = 0
7  """Este bucle while se repetirá mientras el valor que tenga la variable
8  posición + 1 sea menor o igual que longitud. Hay que sumarle uno a la
9  variable posición porque empezamos a contar en 0 y la longitud de un
10 string cuenta el número de caracteres que tiene, por lo que una palabra
11 con una sola letra tendra una longitud de 1 y la posición de su letra
12 será la 0."""
13 while posicion + 1 <= longitud:
14     """Imprimimos el caracter que está en la variable posición"""
15     print (palabra[posicion])
16     """Sumamos uno a la variable posición para ir recorriendo el
17     string"""
18     posicion = posicion + 1
```

Si ya lo tienes un poco más claro podemos seguir. No tengas problema en expresar toda dificultad con las comparaciones entre valores que tienen diferentes rangos o que empiezan en diferentes números (unas en cero y otras en uno).

Éste es el caso de las posiciones de un string respecto de su longitud. Las posiciones empiezan en 0, la longitud empieza en 1, si queremos compararlas hay que conseguir que ambas empiecen en el mismo número, o bien en el 1 y habrá que sumar una unidad a la posición, o bien en el 0 y habrá que restar una unidad a la longitud.

Pasemos ahora a ver cómo funciona un bucle *for*. Prueba el siguiente código guardándolo en un *script*:

```
palabra = input("¿Qué palabra quieres deletrear?: ")
for letra in range(len (palabra)):
    print(palabra[letra])
```

Como habrás podido comprobar, es un deletreador perfecto y en bastantes menos líneas que el programa elaborado en un *while*. Un *for* es un bucle que se repite las veces que

queramos, siguiendo un patrón que le indiquemos, por ejemplo, repite una acción contando hasta 10 de uno en uno o repite hasta 10 contando de dos en dos...

Los bucles *for* necesitan una **variable propia** para ir contando y almacenando el valor de la cuenta. Podemos declarar esa variable en el mismo *for*. En nuestro ejemplo, sin ir más lejos, la variable se llama *letra* y está **declarada directamente** en el bucle *for*. Habitualmente se usa la letra *i* para la variable propia del *for*.

Además, necesitan una indicación de cómo contar y hasta cuánto. En el ejemplo contaremos desde cero hasta la cantidad de caracteres que tenga el *string palabra*. Para ello hemos indicado que cuente dentro del rango (*range*) de la longitud de *palabra*.

Veamos un ejemplo de aplicación para explicarlo con más calma:

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 for.py
¿Qué palabra quieres deletrear?: Python
P
y
t
h
o
n
```

Hemos vuelto a deletrear la palabra *Python*, cuya longitud es 6, así que el bucle *for* viene a ser, traducido a nuestro lenguaje:

Para cada letra en el rango de 6 letras, repite:

Al ejecutarse el bucle *for* este va a ir contando hasta 6 y almacenando el valor de la cuenta en la variable *letra*, eso sí, empezando por 0 (como ya hemos dicho, en programación empezamos a contar en 0).

De esta forma, en la primera iteración o ejecución del bucle *for* la variable *letra* tiene el valor 0, por lo que si le pedimos que imprima la posición *letra* de la palabra *Python* imprimirá la P.

Pero, si le pido que cuente hasta 6 empezando en el 0, ¿no contará 7 veces?. Es importante acordarse que el bucle *for*, al indicarle el rango en el que queremos contar siempre empezará en 0 y llegará hasta **el valor anterior al indicado**.

Si le decimos que cuente en un rango de 6 contará de la siguiente forma: 0, 1, 2, 3, 4 y 5. El valor indicado no lo incluye. Digamos que cuenta hasta el 6 no incluido.

Como esta instrucción suele causar algún problema más en su comprensión vamos a desglosar el bucle tal como lo hace Python con la instrucción *for letra in range (len (palabra))* y *palabra = "Python"*

- Comienza el bucle for. Para cada letra en el rango 6:
 - Letra = 0
 - Imprime la posición 0 de la variable Python: P
 - Letra = 1
 - Imprime la posición 1 de la variable Python: y
 - Letra = 2
 - Imprime la posición 2 de la variable Python: t
 - Letra = 3
 - Imprime la posición 3 de la variable Python: h
 - Letra = 4
 - Imprime la posición 4 de la variable Python: o
 - Letra = 5
 - Imprime la posición 5 de la variable Python: n
- Fin del bucle

Que código más efectivo y breve, ¿verdad?. Pues aún se puede hacer más corto.

El bucle *for* admite muchas formas como instrucción de búsqueda. Pongamos algunos ejemplos:

```
for letra in range [0, 1, 2, 3, 4, 5]:  
    print (palabra [letra] )  
for letra in "Python":  
    print (letra)  
for letra in palabra:  
    print (letra)
```

Los tres bucles hacen exactamente lo mismo. En el primero están enumeradas las posiciones de cada letra de la palabra Python (cosa que no tiene mucho sentido a no ser que sepamos qué palabra vamos a deletrear). El segundo deletrea el *string* que situemos en la condición del *for*, cosa que nos obliga a ceñirnos a un único string, y el tercero recorre la variable *palabra* e imprime cada elemento que contenga (al cual llamamos *letra*).

No vamos a hablar mucho más por ahora de los bucles *for*, iremos utilizándolos a lo largo del curso en otras ocasiones. Ahora toca hacer una prueba: intenta reproducir la cuenta atrás que hicimos con un bucle *while* utilizando ahora un bucle *for* tras preguntar al usuario por la cantidad de la cuenta atrás.

Instrucciones de código usadas

```
#Intentar ejecutar un código y evitar un error
try:
    #código
except:
    #código
#Código que se ejecuta si se cumple una condición (condicional)
if comparacion:
    #código
elif comparacion:
    #código
else:
    #código
#Código que se repita mientras se cumpla una condición (while)
while comparacion:
    #código
#Código que se repita una serie de veces (for)
for repeticiones:
    #código
#Establecer un rango de datos
range(datos)
#Instrucciones de comparación de valores
#Igualdad
==
#Mayor que, menor que, mayor o igual que, menor o igual que
>
<
>=
<=
#Desigualdad
!=
#Importar biblioteca
import biblioteca
#Usar elemento de biblioteca
elemento.biblioteca()
#Importar parte de una biblioteca
from biblioteca import parte
#Usar una posición de un string
string[posicion]
#Longitud de un string
len(string)
```

Reto

Realiza un programa que realice una cuenta atrás de una cantidad de segundos introducida por el usuario. Dicho programa deberá contar cuántos minutos quedan a no ser que quede menos de un minuto, que contará segundo a segundo. Así mismo, deberá obligar al usuario a introducir un número como input.