programación en python 5

BUCLES

# Introducción

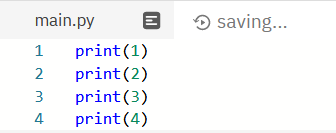
La informática nace para tratar automáticamente la información. Si una tarea completa me va a llevar cinco minutos no tiene sentido invertir media hora en ver cómo automatizarla, pero si la tarea me llevaría una semana, bien puedo invertir un día en mirar cómo aligerarla.

Una de las herramientas más útiles para no tener que repetir un mismo código muchas veces son los **bucles**.

# Bucles

Los bucles nos permiten repetir tareas que son iguales o con pequeñas variaciones, un número de veces determinado o bien hasta que se cumpla alguna condición.

Imagina que queremos pintar los números del 1 al 4. Si quisiéramos repetir una misma operación varias veces sería muy largo y pesado de escribir.

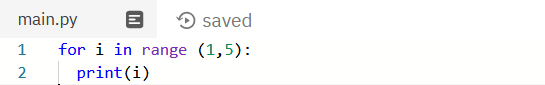


Si lo pensáis bien, todas estas líneas siguen un **patrón**. Es decir, en cada línea estamos imprimiendo el número anterior incrementado en 1. Esto se repite hasta que el número sea el 4. Imaginad que lo queremos hacer hasta el número 100, sería muy largo. Los bucles nos ayudan en esta situación.

# Estructura for

Aquí vemos un ejemplo sencillo de un bucle del tipo **for** que hace lo mismo.

### Vamos a analizar cómo funciona

1. Primero vemos que aparece una variable que se llama i. Es un **contador**, para saber cuántas “vueltas” (iteraciones) llevamos.
2. range(1,5) me indica los valores que va a ir tomando la variable i
3. Los dos puntos dan paso al bloque de código “interior” que es lo que se va a repetir cinco veces. Sacar por pantalla el mensaje y añadir un elemento a la lista.

Así que i empieza en uno y termina en cuatro.

Actividad 1: ejecuta el código y pon una captura del resultado

## Rango

Aunque es habitual que el contador de un bucle vaya desde cero, de uno en uno, hasta un máximo, no es obligatorio. La función **range** nos devuelve una lista de números:

* range(5) nos devuelve 0,1,2,3,4
* range(3,7) devuelve 3,4,5,6
* range(2,15,3) el contador va desde 2 hasta 13, saltando de tres en tres. Es decir, nos da los números 2,5,8,11 y 14
* range (14,2,-3) el contador va decreciendo de 3 en 3. Es decir, nos da los números 14,11,8,5 y 2, en ese orden

## Números pares

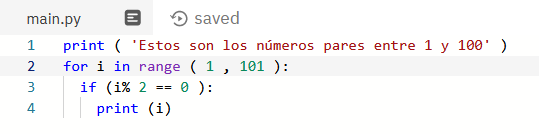
Ahora vamos a hacer un programa que muestre todos los números pares entre 1 y 100.

Los números pares son divisibles por 2. Es decir, dan 0 de residuo al dividir. Aprovecharemos la función % que da el resto de una división.

Es decir:

* Dividir 8 entre 3 da cociente 2 y de resto 2. Por tanto, 8%3 = 2
* Dividir 6 entre 3 da cociente 2 y resto 0. Por tanto, 6%3 = 0

8 no es divisible entre 3, pero 6 sí lo es entre 2



Hacemos correr el contador entre 1 y 101 (ya sabes que se para justo antes, por eso para llegar al 100 tenemos que poner 101). **i**. Recuerda que para comparar usamos el doble ==

Si se cumple, lo escribe, si no nada, y pasaría a la iteración siguiente, porque no hay más instrucciones. Corto, sencillo y rápido. Para eso queremos lo bucles.

Actividad 2: modifica el código anterior para muestre los múltiplos de 5. Adjunta una captura

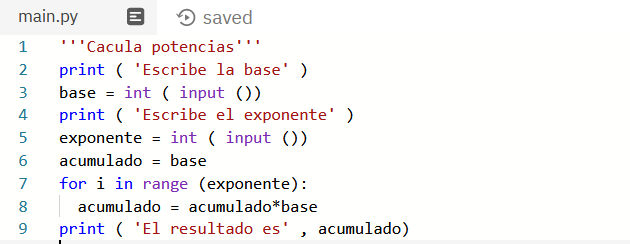
## Calcular potencias

Hagamos otro ejemplo, un programa que calcula potencias. Sería algo así como, me dices que la base vale 2, que el exponente vale 5 y te devuelvo un treinta y dos. Para hacerlo, tengo que ir multiplicando dos por dos, me sale cuatro, luego otra vez por dos… hasta hacerlo cinco veces.

**¿Cuántas variables necesito?**

* Necesito tener guardada la **base**, para ir multiplicándola una y otra vez.
* Necesito tener guardado el **exponente** para saber cuándo he hecho el número de iteraciones suficiente y parar.
* Necesito OTRA variable para ir guardando los números intermedios que me van saliendo. 4, 8, 16 y finalmente 32. A esta variable la voy a llamar **acumulado**.

### Primera prueba

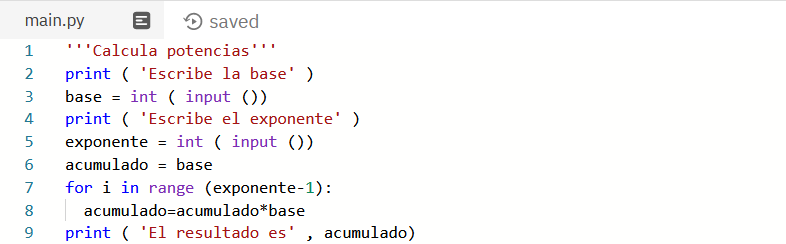


Pongo base 2, exponente 5 y me sale… 64

Algo pasa, vamos a ver. Si acumulado lo inicio con el valor de base, en la primera iteración, ya tengo dos por dos, ya tengo cuatro. Así que en la primera iteración ya tengo dos al CUADRADO.

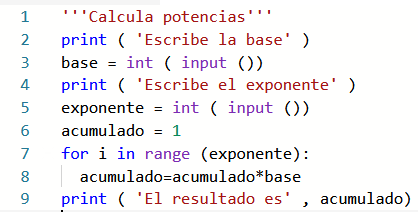
Por lo tanto, tengo que hacer una iteración menos. Probemos.

### Ejemplo arreglado

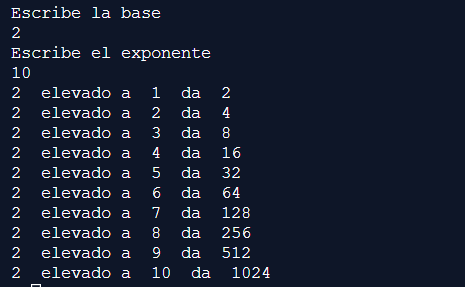


Perfecto, así funciona.

A veces cuesta ver a la primera donde tiene que empezar un contador o hasta donde tiene que ir, pero si probáis con lo que os parece resulta fácil ajustarlo.



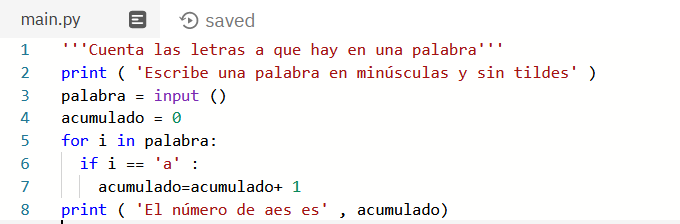
Actividad 3: modifica el programa anterior para que dé la siguiente salida. Adjunta una captura



## Contador de letras en una palabra

Aunque es muy usual numerar las iteraciones, también podemos ir recorriendo con nuestro “contador” cualquier cosa contable… por ejemplo, una palabra.

### Contador de letras “a”



### Funcionamiento del programa

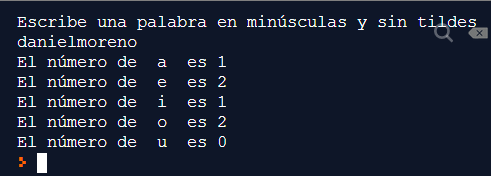
El “contador” i va recorriendo cada carácter de la palabra, y toma como valor ese carácter.

Si ponemos “Dani”.

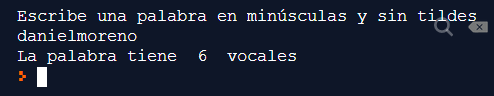
1. En la primera iteración i toma el valor ‘D’, se pasa al condicional, como no es igual a ‘a’ pues no hacemos nada.
2. En la segunda iteración i toma el valor ‘a’, se pasa al condicional, como sí es igual a ‘a’ pues el acumulado se incrementa en una unidad.
3. Etcétera.

Finalmente imprimimos el valor acumulado y listo.

Actividad 4: modifica el programa anterior para que, diciendo tu nombre y apellidos te diga cuantas a, e, i, o, u. La salida por pantalla debe ser la siguiente. Adjunta una captura



Actividad 5: modifica el programa anterior para que te diga cuantas vocales en total tiene la palabra. Adjunta una captura



# BUCLES CONDICIONALES

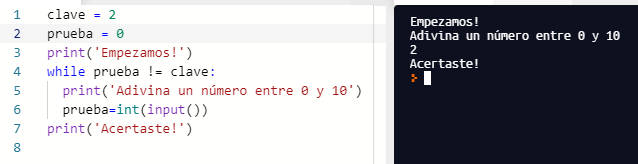
En los ejemplos anteriores conocíamos el número de iteraciones, pero no siempre es el caso.

Esto es lo que se denomina un bucle condicional. Antes de cada iteración se comprueba si se cumple la condición, en el caso afirmativo se realiza una nueva iteración, en caso contrario, se sale del bucle.

## Adivinar número

Pongamos un ejemplo sencillo, adivina un número... hasta que aciertes. Mejor dicho, MIENTRAS falles. Para ello, pediremos al usuario que adivine un número usando un **while.** Utilizaremos varias variables para ir guardando la información.

* **Clave** guarda el número que tenemos que acertar (Se supone que el jugador solo ve la parte de la derecha).
* En **prueba** guardaremos el número que va escribiendo el jugador



### Explicación del programa

El bucle condicional empieza con “while” que significa mientras, se añade la condición que te permite seguir iterando (en este caso que la prueba que escribe en usuario sea distinta a la clave).

Cuando esa condición deje de cumplirse, salimos del bucle y ejecutamos el resto del programa, en este caso, ¡“Acertaste!”.

Algo muy importante en los bucles condicionales es que dentro del bloque de código del condicional SE CAMBIE LA VARIABLE SOBRE LA QUE SE INSPECCIONA LA CONDICIÓN. Si no lo hacemos así, ¿cómo podría dejar de cumplirse la condición y cómo podríamos salir del bucle? Nos quedaríamos “embuclados” ejecutando iteraciones infinitamente o hasta que algo fallara (por ejemplo, un número que anduviera creciendo y que desbordase la memoria, como pasa en la calculadora cuando llegamos a un número demasiado grande para la pantalla).

# BREAK Y CONTINUE. Interrumpiendo bucles.

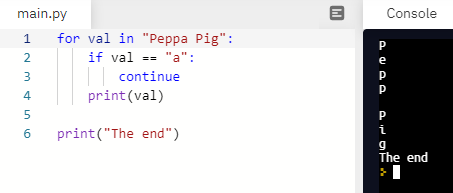
A veces estamos recorriendo un bucle “buscando” algo, de manera que cuando lo encontramos ya no queremos seguir haciendo iteraciones, queremos “interrumpir” la búsqueda de alguna manera.

* BREAK te saca del bucle completamente de forma inmediata.
* CONTINUE interrumpe abruptamente la iteración en la que estás y pasa a la siguiente.

Vamos a poner unos códigos de prueba sólo para ver cómo funcionan

## Ejemplo continue

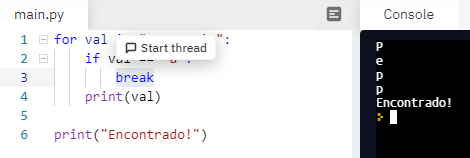
Al cumplirse la condición, continúe hace que se interrumpa la iteración y se pase a la siguiente, por eso no imprime la letra “a”, que aparece. El resto si se imprimen.



Como ves al estar CONTINUE nos salimos de la antes de imprimir ese valor .

## Ejemplo con break

Al cumplirse la condición, break hace que se interrumpa el bucle entero y por eso no se imprimen letras hasta la a.

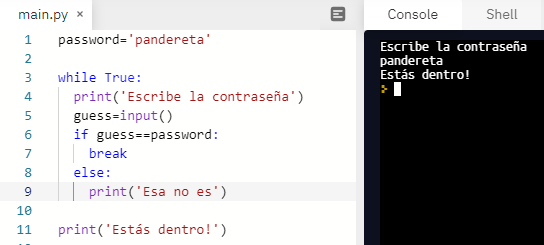


## Ejemplo de password

La estructura “while True” es una manera de escribir un bucle infinito. Significa que algo se va a repetir siempre hasta que cerremos el programa.

Sería como decir “mientras que verdadero sea verdadero...” Funcionaría igual que “while 1=1” o cualquier expresión lógica que sepamos que siempre es verdadera.

Si nos aciertan la contraseña, entramos en el condicional y break nos saca del bucle while, de forma que ya empezamos a ejecutar el código que hay después.



## Calculadora de números primos

Un programa que va calculando los números primos. ¿Cómo podemos construir la lista de los números primos?

### Explicación del problema

El algoritmo que nos enseñaron en el cole. Un número primo solo se puede dividir por 1 y por él mismo.

1. Empezamos nuestra lista con el 2. Este es primo.
2. Cogemos el número siguiente, el 3, y lo dividimos entre los primos que tenemos en nuestra lista (en este caso el 2).
   1. Si sale resto cero con alguno es que no es primo
   2. Si no sale resto cero y el cociente es menor que el divisor, entonces sabemos que ya será primo.

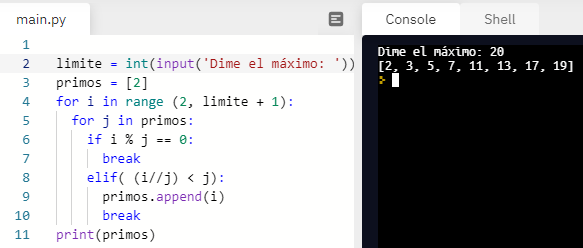
### Cómo lo programamos

Ahí veréis que hay dos puntos de ruptura, uno si encontramos que no es primo y otro si después de dividir el cociente es menor que el divisor.

Por ejemplo, imagina que vamos por el llevamos la lista llena hasta el 13, esto quiere decir que será [2,3,5,7,11,13].

1. Nos toca probar el 14. Hacemos la división 14 entre 2, resto cero. Vaya, no es primo.
2. Nos toca probar el 15. Hacemos la división 15 entre 2, resto uno, cociente 7, mayor que 2, puede ser primo.
3. Siguiente primo de la lista, el 3. Hacemos la división 15 entre 3, vaya, resto cero. No es primo.
4. Con el 16 pasa lo mismo que con el 14, son pares. BREAK.
5. Nos toca ahora el 17. Hacemos la división 17 entre 2, resto uno, cociente 8, mayor que 2, puede ser primo, siguiente primo de la lista, el 3. 17 entre 3, resto 2, cociente 5, mayor que tres, puede ser primo. Siguiente primo de la lista, el 7, resto 3, ¡cociente 2 MENOR que 7... es primo! Lo añadimos a la lista de primos y BREAK.

Vamos calculando los números primos y metiéndolos en una lista



Te explico línea a línea. Pedimos un límite.

Inicializamos la lista de primos con el número 2. El bucle en i recorrerá todos los números de uno en uno hasta el máximo que nos han indicado.

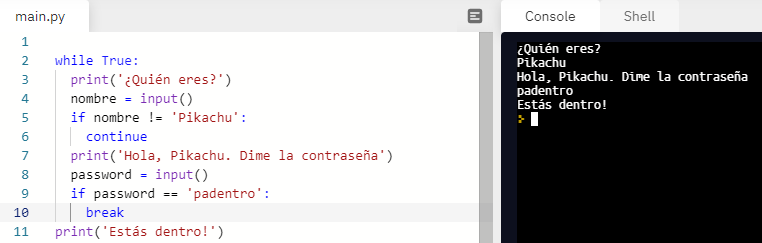
El bucle en j recorre todos los primos para cada número que vamos eligiendo en el bucle en i Comprueba si el resto de dividir el número i entre el primo j da cero. Si es así, salimos del bucle en j, no hay que comprobar con más primos, ya sabemos que no lo es.

Si no sale cero (elif), comprueba si el cociente es menor que el divisor, en ese caso, sabemos que es primo, lo añadimos a la lista de primos y podemos dejar de comprobar con el resto de primos (bucle en j).

Finalmente, cuando ya hemos terminado con todos los números hasta el que nos dijeron, cuando se termine el bucle en i, podemos imprimir la lista completa de primos.

## Programa nombre y contraseña

Programa que te pregunta el nombre y sólo si eres el indicado te pregunta la contraseña. Te pregunta quién eres y sólo si eres la persona indicada te pregunta la contraseña.



### Funcionamiento

Como antes tenemos un bucle infinito (while True) (del que sólo podremos salir a las bravas). Nos pregunta quién somos y recoge la respuesta.

1. Si no digo el nombre correcto, interrumpe ahí el bucle y lo vuelve a empezar, preguntándome de nuevo quién soy.
2. Si digo el nombre correcto, sigue adelante
   1. Me pregunta la contraseña, que guarda en una variable.
   2. Si la contraseña es correcta sale del bucle infinito y me dirá “Estás dentro”, si no, vuelve a empezar el bucle y me preguntará el nombre de nuevo.