Python para

ESO

y

Bachillerato

Juan Rodríguez Aguilera
I.E.S. López de Arenas

Nota:

Estos apuntes están elaborados con el objetivo de ser una herramienta para introducir al alumnado de Secundaria y Bachillerato en el mundo de la programación.

Se ha elegido Python debido a las características que tiene este lenguaje de programación: fácil de aprender, fácil de leer, open source, multiplataforma, muy versátil, sirve para cualquier propósito, etc.

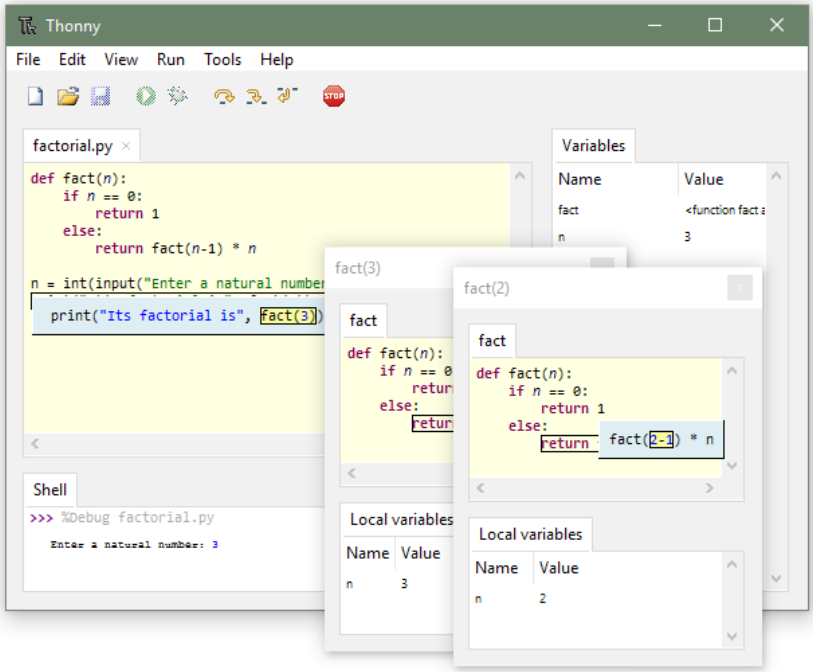
Esta versión está aún incompleta puesto que aún se encuentran en estado de elaboración.

Marzo de 2018.

Juan Rodríguez Aguilera

Capítulo 0

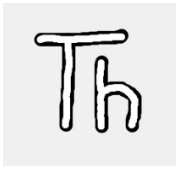
Instalación
Entorno Thonny
Generalidades



Python es un lenguaje de programación interpretado que por defecto ya se encuentra instalado en muchos sistemas operativos. De todas formas, se trata de un lenguaje desarrollado bajo la filosofía de software libre y está disponible de manera gratuita para todos los sistemas operativos. (http://www.python.org)

Para escribir programa en Python, sólo necesitamos un editor de textos (Bloc de Notas, gEdit, etc...) y el intérprete que será el que haga que las instrucciones escritas en el editor de textos se ejecuten.

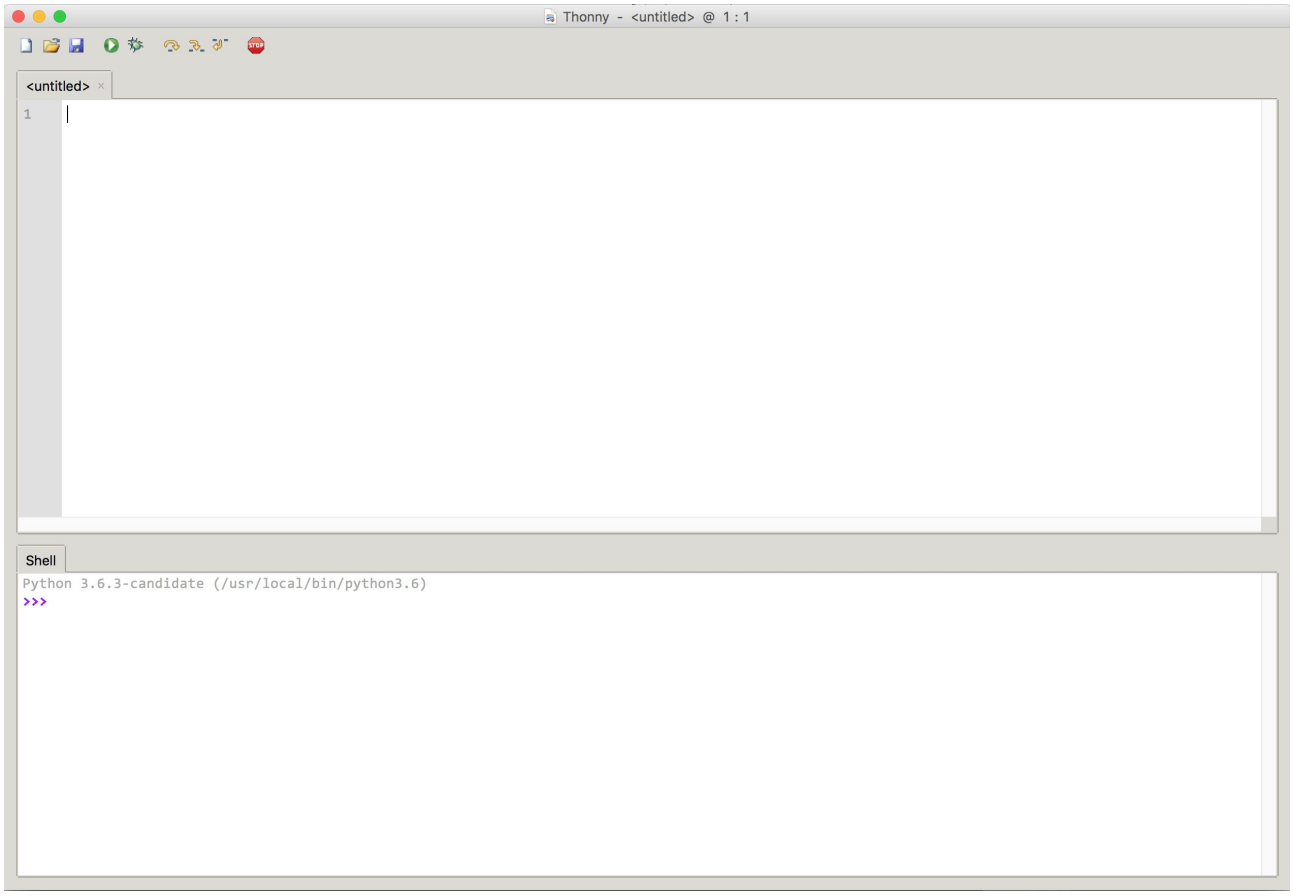
Hay muchos (programas) entornos de desarrollo integrados (IDE's) de gran calidad y con muchísimas posibilidades para desarrollar aplicaciones en Python de manera profesional. Sin embargo, estos entornos tienen tantas características y opciones, que suelen abrumar a las personas se están iniciando en el mundo de la programación y resultan más bien una barrera, si de lo que se trata es de dar los primeros pasos en programación.



Por lo expuesto en los párrafos anteriores, propongo usar el entorno de programación **Thonny**, el cual está diseñado y pensado precisamente para las personas que desean iniciarse en el mundo de la programación en Python. En mi opinión es un entorno ideal para el alumnado de Secundaria y Bachilleratos.

Sin embargo, todo lo recogido en este libro se puede llevar a cabo con cualquiera de los restantes IDE's existentes en la actualidad para Python.

El IDE Thonny, al igual que Python, está desarrollado bajo la filosofía de software libre y se encuentra disponible para los principales sistemas operativos. Podemos obtenerlo desde la web http://www.thonny.org, la instalación es muy fácil y se encuentra bien explicada en la página de descarga del programa. **Thonny ya trae Python incluido en su instalación.**



El programa inicialmente divide la ventana en 2 partes bien diferenciadas, aunque esta configuración se puede modificar mediante el menú View y acomodar al gusto de cada usuario. De todas formas, para el aprendizaje recomiendo que se respeten únicamente las 2 regiones que se muestran en la imagen anterior, puesto que facilita la comprensión de la programación por parte del alumno.

El editor de código

La región superior de la ventana, se trata de un editor de texto como el Bloc de Notas, gEdit, o cualquier otro al que estemos acostumbrados a utilizar, es decir, un programa que guarda el contenido como texto plano, sin ningún tipo de formato.

Este editor al estar pensado para escribir texto en Python, tiene además la particularidad de que nos resaltará la sintaxis y nos ayudará con la escritura de instrucciones facilitándonos las propiedades y métodos disponibles en cada situación y auto completando nombres de funciones y variables. Para ello será de gran ayuda el uso de la tecla del **Tabulador** o de la combinación de teclas **Ctrl + Espacio**.

El editor de código nos permitirá guardar el trabajo en archivos que deben tener la extensión **.py**, de esta forma en cualquier momento podemos recuperar dicho archivo y continuar con el trabajo en el punto donde lo hayamos dejado.

En el editor de texto, una vez que tengamos nuestro código elaborado y listo para su ejecución, sólo es necesario pulsar el botón Play (o pulsar F5) y nuestro programa se ejecutará sin más complicaciones.

Es importante destacar que también tenemos la posibilidad de ejecutar el programa paso a paso e ir comprobando en cada momento el resultado de cada una de las operaciones que se están llevando a cabo.

El IDE Thonny tiene por defecto un diseño simple que ayuda a centrar la atención en la escritura del código y en los resultados que se obtienen de su ejecución, descarga al alumno de infinidad de opciones que son muy útiles para programadores experimentados, pero que distraen y dificultan el aprendizaje de los alumnos que se están iniciando en el mundo de la programación.

El intérprete de Python

La región inferior de la ventana, muestra lo que en Python se denomina el intérprete. Se trata de una línea de comando donde podemos introducir instrucciones y comprobar el resultado que produce su ejecución.

Este intérprete es similar al que podemos obtener desde la línea de comandos de nuestro sistema operativo al ejecutar el comando python (c:\>python, usuario$ python, ...)

El intérprete es ideal para hacer pruebas de pequeñas porciones de código y para aprender el funcionamiento de determinadas instrucciones.

En el siguiente capítulo de este libro, comenzaremos a utilizar este intérprete para familiarizarnos con aspectos básicos de Python.

El intérprete de Python dispone de una ayuda que podemos consultar para conoce la sintaxis de cualquier instrucción. Se trata de un pequeño programa interactivo que podemos ejecutar con las función help()

```
>>> help()

help>
```

Observa que al ejecutar el función help(), se obtiene un nuevo prompt en el intérprete.

El aspecto de la ayuda es: help>

En el nuevo prompt podemos escribir cualquiera de las palabras reservadas de Python y obtener la ayuda que está disponible:

```
help> for
```

```
help> for
The "for" statement
*******************

The "for" statement is used to iterate over the elements of a sequence
(such as a string, tuple or list) or other iterable object:

   for_stmt ::= "for" target_list "in" expression_list ":" suite
                ["else" ":" suite]

The expression list is evaluated once; it should yield an iterable
object.  An iterator is created for the result of the
"expression_list".  The suite is then executed once for each item
provided by the iterator, in the order returned by the iterator.  Each
item in turn is assigned to the target list using the standard rules
for assignments (see Assignment statements), and then the suite is
executed.  When the items are exhausted (which is immediately when the
sequence is empty or an iterator raises a "StopIteration" exception),
the suite in the "else" clause, if present, is executed, and the loop
terminates.

A "break" statement executed in the first suite terminates the loop
without executing the "else" clause's suite.  A "continue" statement
executed in the first suite skips the rest of the suite and continues
with the next item, or with the "else" clause if there is no next
item.

The for-loop makes assignments to the variables(s) in the target list.
This overwrites all previous assignments to those variables including
those made in the suite of the for-loop:

   for i in range(10):
       print(i)
       i = 5             # this will not affect the for-loop
                         # because i will be overwritten with the next
                         # index in the range
```

Para salir de modo help>, podemos dar la instrucción **quit** y volver al intérprete:

```
help> quit

>>>
```

Desde el intérprete también podemos obtener ayuda directamente:

>>> help('for')

Palabras reservadas de Python (palabras claves):

and	as	assert	break	class
continue	def	del	elif	else
except	finally	for	from	global
if	import	in	is	lambda
nonlocal	not	or	pass	raise
return	try	while	with	yield
False	None	True		

Generalidades

Para finalizar este primer capítulo introductorio, vamos a exponer algunos aspectos generales sobre Python que resultan de vital importancia para poder escribir código en este lenguaje.

Sensible a mayúsculas y minúsculas, Python es un lenguaje que diferencia entre mayúsculas y minúsculas, esto significa: For es diferente de for, nombre es diferente de Nombre, etc.

Bloques de código. Hay determinadas instrucciones (for, while, id, elif, else, def, try, except) que tienen por objetivo ejecutar un conjunto de instrucciones que están relacionadas entre sí, a estas instrucciones se le llaman bloque de código.

Las instrucciones que definen el comienzo del bloque de código, deben terminar en el carácter de los 2 puntos (**:**).

Indentación, es la forma que tiene Python de realizar bloques de código. Se llama así una especie de sangría antes del texto que se establece para clarificar la escritura del código. Veámoslo mejor con un ejemplo:

```
1     for n in (1,2,3,4,5,6):
2         print(n, n*10, n*100)
3         print('------', n, '------')
4         print()
```

En la línea 1 se ha usado una instrucción que lleva implícita la ejecución de un bloque de código, por ese motivo esa instrucción finaliza en **:**

Las líneas 2, 3, y 4 forman el bloque de código asociado a la instrucción de la línea 1. Estas líneas están asociadas porque están desplazadas unos espacios hacia la derecha con respecto a la línea 1.

Aunque no es obligatorio, se ha acordado que la indentación estándar sea de 4 espacios.

La indentación es una práctica habitual en la mayoría de lenguajes de programación porque mejora la lectura del código, es una buena forma de visualizar las líneas que están relacionadas entre sí y las que están incluidas, unas dentro de otras. Sin embargo, en Python es una regla para escribir el código, es decir, no existen otros caracteres para agrupar código, necesariamente hay que hacerlo a través de la indentación.

Thonny, nos ayudará con la indentación en la escritura del código cada vez que sea coherente con el código que estemos escribiendo.

Comentarios

Un comentario se llaman a un texto incluido dentro del código pero que no tiene ningún efecto sobre la ejecución del código. Un comentario es una nota aclaratoria sobre la funcionalidad del programa, el uso de una variable, o cualquier otra aclaración que se considere necesaria para comprender mejor el código que se está escribiendo.

En Python tenemos 2 formas de escribir comentarios:

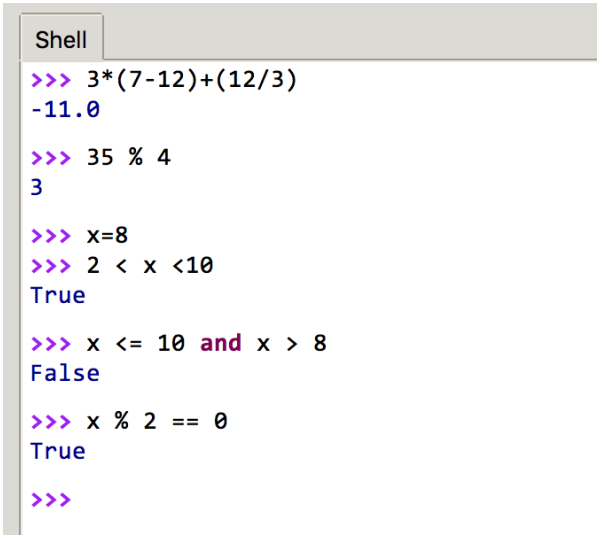
Con la almohadilla #, podemos escribir comentarios de una sola línea: # Comentario

Con las triples comillas podemos escribir código de varias líneas. Son válidas las comillas dobles y las comillas simples:

''' Este texto aunque sea de varias líneas contiene un comentario '''

Capítulo 1

Tipos de datos
Operadores
Variables



Vamos a comenzar a usar Python desde el intérprete como si se tratase de una calculadora inteligente capaz de hacer cuentas y cálculos de forma rápida y precisa.

```
>>> 3 + 4 * (8 – 2)

27

>>> 45 / 7

6.428571428571429

>>> 45 // 7

6

>>> 7**2

49

>>> 43 % 7

1
```

Como puedes ver en los ejemplos Python puede realizar las operaciones básicas suma(+), resta(-), multiplicación(*) y división(/) y además, sabe aplicar correctamente la prioridad de unas operaciones sobre otras.

También habrás observado que hace algunas operaciones más como la potenciación(**), división entera(//) y el resto de la división(%, esta última también recibe el nombre de módulo o residuo).

Más adelante veremos como también es capaz de hacer muchísimas operaciones más, pero por el momento, es suficiente con estas.

Vamos a seguir dándole algunas órdenes más a Python, que aunque nos puedan parecer un poco extrañas, ya veréis como más tarde resultarán útiles.

```
>>> “Hola”

'Hola'

>>> “Hola ” + “Arturo y Julio”

'Hola Arturo y Julio'

(Observa que en esta ocasión después de Hola se ha incluido un espacio para que el resultado se lea mejor y aparezca escrito correctamente).

>>> 3 * 'Hola'

'HolaHolaHola'
```

En estos ejemplos hemos escrito texto entrecomillado, este tipo de información recibe el nombre de **cadenas de texto** (strings). En Python las cadenas de texto se pueden escribir entre comillas dobles o entre comillas simples.

En el segundo ejemplo hemos sumado 2 textos, visto así, esto te debería resultar un poco extraño. La explicación es que realmente no hemos sumado, lo que hemos hecho ha sido concatenar (unir) 2 cadenas de texto y hemos obtenido una nueva cadena texto con todo el texto.

Siguiendo la misma lógica anterior, multiplicar un número por un texto, es concatenar varias veces el texto consigo mismo.

Vamos a continuar dándole órdenes al intérprete de Python para conocer las respuesta que nos devuelve.

```
>>> 8 > 3

True

>>> 5 < 2

False

>>> 6 >= 4 and 8 > 7

True

>>> 7 != 5

True
```

Ahora nuestras órdenes lo que han hecho ha sido comparar valores numéricos y Python se ha encargado de decirnos si el resultado de las comparaciones es cierto (True) o falso (False).

A los valores True y False, se le llaman valore lógicos o booleanos.

Estos valores se utilizan para comprobar si se cumple alguna condición en un determinado momento. Estos valores en programación tienen una gran importancia porque sirve para tomar decisiones dentro del programa en función de las condiciones que se den.

En los ejemplos anteriores hemos realizado operaciones que seguramente te habrán resultado muy sencillas y fáciles de comprender. Es probable que pienses que no serán de mucha utilidad, pero no es así, comprender como es la información que manipula un ordenador y qué tipo de operaciones se puede hacer con ellas, es la base para conseguir darle instrucciones a un ordenador de forma correcta.

Antes de continuar, vamos a extraer algunos aspectos importantes de los ejemplos que hemos escrito antes.

Hemos usado diferentes tipos de datos: **numéricos** (enteros y decimales), **textos** (strings) y datos **lógicos** (booleanos). Python tiene más tipos de datos, pero por el momento, nos quedaremos con estos que son los más comunes cuando se empieza a programar.

Hemos realizado diferentes tipos de operaciones con los datos: operaciones matemáticas, operaciones con cadenas de texto y comparaciones cuyo resultado ha sido un valor lógico. A los símbolos que hemos utilizado para realizar estas operaciones, se les llama operadores, por lo tanto, hemos usado: **operadores matemáticos**, **operadores de cadenas** y **operadores de comparación**.

Operador	Tipo	Operación que realizan
+	Matemático	Sumas datos numéricos
-	Matemático	Restar datos numéricos
*	Matemático	Multiplicar datos numéricos
/	Matemático	División de 2 valores numéricos
//	Matemático	División entera, toma la parte entera del resultado
**	Matemático	Potenciación, para obtener un número elevado a un exponente
%	Matemático	Resto, módulo o residuo, se obtiene el resto de la división
+	Textos	Concatenación, unión de cadenas de texto
*	Textos	Repetir varias veces una misma cadena de texto
[]	Textos	Obtener partes de un texto
<	Comparación	Un valor menor que otro
<=	Comparación	Un valor menor o igual que otro
>	Comparación	Un valor mayor que otro
>=	Comparación	Un valor mayor o igual que otro
==	Comparación	Comprobar si dos valores son iguales
!=	Comparación	Comprobar si dos valores son diferentes
& (and)	Comparación	Comprobar si se cumplen varias condiciones
| (or)	Comparación	Comprobar si se cumple alguna de las condiciones
=	Asignación	Asignarle un valor a una variable. Ej.: x=5
+=	Asignación	Ej.: x+=1, es equivalente a, x=x+1
-=	Asignación	Ej.: x-=1, es equivalente a, x=x-1
*=	Asignación	Ej.: x*=2, es equivalente a, x=x*2
/=	Asignación	Ej.: x/=2, es equivalente a, x=x/2
%=	Asignación	Ej.: x%=2, es equivalente a, x=x%2

Nota:

Dentro de los operadores de comparación, Python también permite operadores ternarios, es decir, expresiones así: **1 <= x <= 10**. Esta forma de escribirlo es similar a como se haría en matemáticas y es equivalente a: x>=1 and x<=10, sin embargo resulta más simplificada y más intuitiva de leer.

Existen algunas variantes más de estos operadores y algunos conceptos más que poco a poco iremos conociendo, practicando y asimilando.

En los ejemplos anteriores hemos realizado operaciones y gestionado información que de poco serviría si nuestros programas no pueden guardar esos datos para volver a utilizarlos posteriormente. La forma de guardar en la memoria del ordenador estos datos es dándole un nombre y asignándole el valor a dicho nombre.

Este es el concepto de variable, se trata de un nombre al cual se le asigna un valor. Este valor puede ser directo o como resultado de una operación.

```
>>> x = 5
>>> y = x**2
>>> y
25
>>> x + y
30
>>> nombre1 = 'Arturo'
>>> nombre2 = 'Julio'
>>> nombre1 + ' ' + nombre2
'Arturo y Julio'
```

Las variables son un almacén donde guardar una información y que en cualquier momento podemos recuperarla para volver a utilizarla.

El nombre de las variables deben cumplir ciertas reglas:

- No deben se palabras reservadas de Python
- Deben comenzar por una letra
- No pueden contener espacios ni caracteres extraños
- Python hace distinción entre mayúsculas y minúsculas

En los lenguajes de programación, **el símbolo = debe entenderse como una asignación**, no como una igualdad matemática. La expresión: **x = x +1**, en matemáticas no tendría mucho sentido puesto que sería afirmar que 1 = 0, sin embargo, en cualquier lenguaje de programación, es totalmente válida y **debe entenderse como asígnale a x el valor que tenga en este momento más una unidad**.

Importante: En Python, las variables no es necesario declararlas ni indicar qué tipo de información va a contener, basta con asignarles un valor y nada más.

A continuación vamos a seguir avanzando dentro del intérprete de Python con algunas instrucciones que resultan básicas en la elaboración de cualquier programa. Estas funciones son las funciones de entrada y salida de información, es decir, las principales funciones para mostrar alguna información por pantalla y para introducir datos en el programa.

Las funciones están formadas por un nombre, seguidas de paréntesis, y dentro de estos paréntesis se escriben los argumentos o parámetros (valores que se le pasan a la función).

La primera de estas funciones que vamos a ver, es la función print, que se utiliza para mostrar información por pantalla y cuya sintaxis es:

print(argumento1, argumento2, argumento3, ...)

Vamos a practicar en el intérprete las siguientes instrucciones:

```
>>> print('¡¡ Hola Mundo !!')
¡¡ Hola Mundo !!
>>> nombre1 = 'Arturo'
>>> nombre2 = 'Julio'
>>> print("Buenos días", nombre1,"y",nombre2)
Buenos días Arturo y Julio
>>> print("El resultado de", 5, "por", 9, "es",5*9)
El resultado de 5 por 9 es 45
>>> print("Buenos"+" "+"días")
Buenos días
```

La instrucción print la usaremos para mostrar mensajes por pantalla.

Observa que cuando se le pasan varios **argumentos separados por comas**, la función print al mostrar estos valores, **introduce un espacio** entre ellos para que se visualicen mejor.

Un buen uso de las cadenas de texto nos ayudará a mostrar bien la información.

Al igual que la función print sirve para mostrar información, disponemos de la función input para introducir datos en el programa para que puedan ser procesados. Estos datos se almacenan en una variable y se utilizan cuando el programa los necesita en función de las tareas que deba llevar a cabo.

variable = input('Mensaje a mostrar al solicitar el dato')

```
>>> n = input('Escribe un número: ')
      Escribe un número:
```

A partir de este momento la variable n tendrá como contenido el texto que escribamos hasta la pulsación de Intro.

Es importante resaltar que desde la versión 3.x de Python la función input devuelve una cadena de texto, en las versiones anteriores (2.x de Python), esta función devolvía el resultado de evaluar matemáticamente la expresión escrita.

Por lo tanto, si queremos usar numéricamente el valor introducido mediante input, tendremos que usar alguno de los métodos que Python facilita para convertir un texto en un valor numérico.

```
>>> n = input('Escribe un número: ')
      Escribe un número: 12
>>> n
'12'
>>> v = int(n)
>>> v
12
>>> print(3*n,3*v)
121212 36
```

En estos ejemplos podemos comprobar que el valor de n es el texto '12', sin embargo, el valor de v es el número 12.

La conversión de n en un valor numérico podemos hacer de varias formas distintas:

```
v = int(n)
v = float(n)
v = eval(n)
```

eval permite que n contenga cualquier expresión válida en Python, por ejemplo: 3*(7-5)**2

Algunas aclaraciones sobre la función input y sus conversiones:

Los lenguajes de programación permiten realizar todo tipo de operaciones con la información que le vamos suministrando, sin embargo es lógico que para determinadas operaciones, los datos deban ser de un determinado tipo, por ejemplo, no tiene mucho sentido hacer determinadas operaciones matemáticas con cadenas de textos

Si llevamos a cabo las instrucciones:

```
>>> n = input('Escribe un número: ')
      Escribe un número: 5.3
>>> v = float(n)
>>> print(n,'----',v)
5.3 ---- 5.3
>>> v2 = int(n)
```

Para que la conversión de una cadena de texto a entero o float, se realice correctamente, si usamos las funciones int() o float() los valores deben ser del tipo correspondiente.

La función eval() es más versátil que las funciones int() y float().

Se produce un **error por ser n un valor inválido** para convertirlo en un valor entero.

En este sentido, es necesario ser cuidadoso al realizar operaciones con valores de diferentes tipos de datos. Generalmente será necesario convertir alguno de los datos al tipo conveniente para que se puedan llevar a cabo las operaciones. Por ejemplo: "Hola" + 5, generará un error porque no se puede concatenar un texto con un valor entero entendido como tal, sin embargo, sí podemos convertir el número 5 a un texto y de esa forma poder concatenarlo con el texto 'Hola'. Esto podríamos hacerlos así: 'Hola' + str(5), se obtendría 'Hola5'.

```
>>> v3 = eval(n)
>>> print(v3)
5.3
>>> n = input('Escribe una expresión evaluable: ')
      Escribe una expresión evaluable: 4*(8-2*3)
>>> v = eval(n)
>>> print(n,'----',v)
4*(8-2*3) ---- 8
```

En la mayoría de ocasiones la función eval() es la más adecuada para convertir cadenas de texto en valores numéricos.

Incluso si tienes una variable x con un determinado valor, puedes usar la función eval, para evaluar expresiones del tipo: 2*(x-1)**2

Tipos de datos más elaborados

Ya conocemos básicos de Python, sin embargo, disponemos de otros tipos de datos que resultan de gran utilidad en la programación. Estos datos más elaborados suelen estar compuestos por colecciones de datos básicos (enteros, decimales, textos o lógicos).

Listas

Las listas en Python son una colección de datos que resultan de gran importancia y ahorran muchísimo trabajo en la realización de un programa. Por el momento vamos a indicar como se crean, como podemos hacer un uso básico de ellas y más adelante profundizaremos en su conocimiento.

```
>>> lista = ['Arturo', 11, 'Julio', 7.5, True]
>>> lista[0]
'Arturo'
>>> lista[4]
True
>>> lista[3]
7.5
```

En las listas se almacenan varios datos que posteriormente pueden ser referidos mediante un índice.

Para indicar el elemento hay que utilizar corchetes [].

Las listas comienzan a numerarse por 0 (cero).

El uso de listas es muy común al programar con Python.

En Python tenemos muchas opciones disponibles que facilitan manipular el contenido de las listas. Vamos a ver algunas de estas posibilidades continuando con el ejemplo anterior:

```
>>> lista.append('Saludos')
>>> lista
['Arturo', 11, 'Julio', 7.5, True, 'Saludos']
>>> lista.remove(11)
>>> lista
['Arturo', 'Julio', 7.5, True, 'Saludos']
>>> lista.reverse()
>>> lista
['Saludos', True, 7.5, 'Julio', 'Arturo']
>>> lista.insert(1, 'Hola')
>>> lista
['Saludos', 'Hola',True, 7.5, 'Julio', 'Arturo']
```

Observa que en las listas se puede almacenar información de diferentes tipos de datos.

En una misma lista puede haber texto, números, valores lógicos, etc.

Seleccionar uno o varios elementos de una lista, es muy fácil de hacer en Python, vamos a ver algunos ejemplos más que mostrarán algunas de estas posibilidades disponibles.

```
>>> a=[7, 'Hola', 12, True, 'Saludos', 4.3, False]
>>> a[1:4]
['Hola', 12, True]
>>> a[-3]
'Saludos'
>>> a[:3]
[7, 'Hola', 12]
>>> a[4:]
['Saludos', 4.3, False]
```

Nota importante: Las cadenas de texto, también permiten que se puedan acceder a sus caracteres de la misma forma que las listas. Esto no quiere decir, que las cadenas de texto sean listas, pero sí podemos acceder a letras de la siguiente forma:

```
>>> nombre = 'Arturo'
>>> nombre[0]
'A'
>>> nombre[1:4]
'rtu'
>>> nombre[:4]
'Artu'
>>> nombre[-3:]
'uro'
```

En varios de los ejemplos anteriores has podido observar que al indicar dos índices separados por 2 puntos, se obtienen todos los elementos cuyos índices se encuentran comprendidos entre ambos índices, incluido el primero de los índices, pero excluido el último, es decir, nombre[1:4] incluirá los caracteres que se correspondientes a los índices 1, 2 y 3.

Recuerda que los índices se comienzan a numerar por el 0.

Cuando indicamos [:n] estamos diciéndole que seleccione desde el principio y cunado indicamos [n:], le decimos que llegue hasta el final.

Las listas son una herramienta muy poderosa en Python y las utilizaremos en multitud de ocasiones puesto que facilitan muchas operaciones con la información que gestionen nuestros programas.

```
lista1=[]          #Esta instrucción define una lista llamada lista1 pero vacía,
                   sin ningún elemento por el momento.
```

Las listas disponen de funciones (métodos) que permiten gestionar su contenido de forma bastante cómoda: append, clear, copy, count, extend, index, insert, popremove, reverse y sort. Cada una de estas funciones realiza una tarea específica con el contenido de una lista.

En Python disponemos de otros tipos de colecciones de datos: tuplas, diccionarios y conjuntos, los cuales tienen funcionalidades parecidas a las listas aunque con diferencias entre unas y otras.

Por el momento, nos vamos a quedar únicamente con las listas.

Ejercicios:

Ejercicio 1:

Evaluar las expresiones y escribir el resultado:

4*3-5*2+3*(-1)	5*(8-2*7+1)	4-6*(3+2*4)	4*(3-5)+(12+8)*3
2+4**3	(2+4)**3	2**10	36 % 8
36 % 7	36 % 6	16 / 5	16 // 5
3 < 7	5 <= 6	5 >= 5	6 < 8 and 7 > 9
5 == 4	7 != 8	6 == 4 or 5 < 7	6 == 4 and 5 < 7

Ejercicio 2:

Realizando previamente la asignación n=5, realiza en el orden que aparecen las siguientes operaciones y escribe el resultado obtenido (primero de izquierda a derecha y después hacia abajo):

y=2*n+2	y	n+=1	x=3*n
x	x**2	x%=5	x*=4
n**(1/2)	n / x	x // y	n /= 2

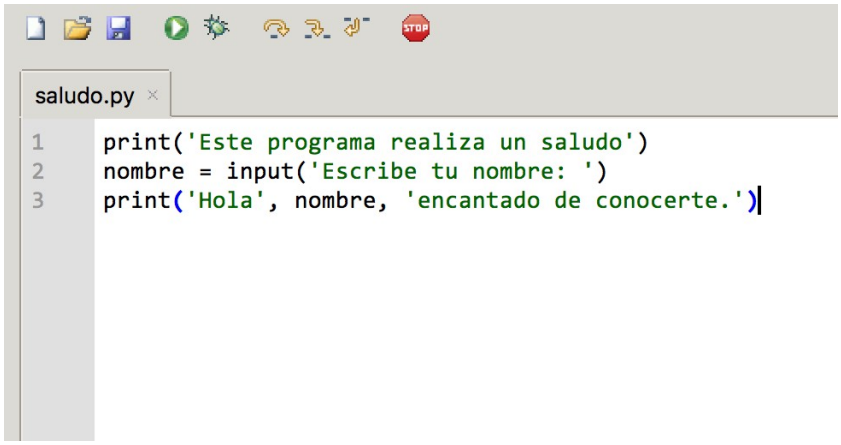
Ejercicio 3:

Realizando previamente la asignación palabra1='Informática' y palabra2='programación', obtén los resultados de estas expresiones:

palabra1+palabra2	palabra1+'---'+palabra2	palabra1[3]
palabra1[0]+palabra2[2]	palabra1[:4]	palabra2[-4]
palabra2[6:]	palabra1[:]	list(palabra1)

Capítulo 2

Primeros programas



Es muy habitual en el mundo de la programación, comenzar escribiendo un programa que visualice un mensaje por pantalla y de esta forma muestre como se hace una de las tareas más simples y habituales, la demostrar información.

A este programa se le suele llamar ¡Hola Mundo!, que viene a representar un saludo a este nuevo lenguaje de programación.

A partir de este momento y salvo que digamos lo contrario, nuestras instrucciones las escribiremos en el editor de código, y así comenzaremos a escribir pequeños programas que ayudarán a desarrollar la habilidad y la lógica de programación. Por lo tanto, ya no veremos en nuestros ejemplos el prompt del intérprete (>>>).

En Python, como ya vimos en el capítulo anterior, mostrar el mensaje ¡Hola Mundo! es tan simple como escribir una única instrucción:

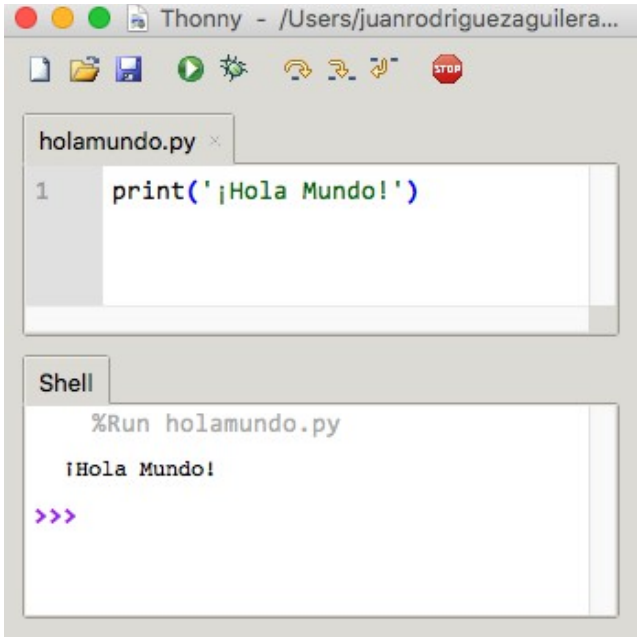
```
1     print('¡Hola Mundo!')
```

En este programa hemos usado la función print para mostrar la cadena de texto '¡Hola Mundo!', recuerda que también se podría haber escrito con comillas dobles print("¡HolaMundo!")

Antes de ejecutar el programa vamos a guardar nuestro código un archivo al cual le pondremos el nombre: holamundo.py

Thonny antes e ejecutar cualquier código escrito en el editor, nos pedirá que lo guardemos en un archivo. Recuerda que en Python se recomienda que los archivos de código tengan la extensión .py

Para ejecutar nuestro programa sólo hay que pulsar sobre el botón **Run** o la tecla **F5**. Obtendremos una salida similar a la siguiente imagen:



Ya hemos conseguido hacer nuestro primer programa.

Te animo a que hagas algunas pruebas para familiarizarte con Thonny y el **modo depuración** (**Debug**) pulsando el botón que se encuentra justo a la derecha de Run (o Ctrl+F5) a partir de aquí ve haciendo pruebas con los botones siguientes o sus correspondientes acciones con el teclado (F6, F7 y F8).

Este modo depuración te ayudará a comprender mejor el funcionamiento del código que has escrito.

En muchas ocasiones y sobre todo al principio, el programa no realizará las acciones que nosotros teníamos previstas. Para esos casos, usar el modo depuración puede resultar de mucha ayuda.

F5	Ejecuta el programa que tengamos en el editor de código en modo normal
Ctrl+F5	Ejecuta el programa en modo depuración (Debug)
F6	En el modo Debug, ejecuta completamente la instrucción resaltada
F7	En el modo Debug, ejecuta cada una de las partes de la instrucción resaltada
F8	En el modo Debug, sale del modo ejecución indicado con F7

En ventana de nuestro intérprete, que se denomina **Shell**, estamos viendo el resultado de la ejecución de nuestro programa. Es normal, que esta ventana se llene pronto de contenido, mensajes de error, pruebas, etc. En cualquier momento puedes limpiarla pulsando con el botón derecho del ratón sobre ella y eligiendo la opción **Clear Shell**. Esta opción también se encuentra disponible en el **menú Edit**.

Programa: saludo.py

Como ya sabemos como escribir un programa y también como mostrar información, vamos a seguir avanzando y nuestro próximo programa hará lo siguiente:

1º Mostrará un mensaje indicando que se trata de un programa para elaborar un saludo.

2º Solicitará el nombre de la persona que está usando el programa.

3º Mostrará un nuevo mensaje saludando a la persona con el nombre que haya escrito.

Para conseguir hacer este programa vamos a usar 2 conceptos expuestos en el capítulo anterior: la función input para introducir información al programa y el uso de una variable para almacenar la información introducida.

```
1     print('Este programa realiza un saludo')
2     nombre = input('Escribe tu nombre: ')
3     print('Hola', nombre, 'encantado de conocerte.')
```

Aunque se trata de un programa muy simple, es importante que se comprenda perfectamente cada una de sus instrucciones:

La línea 1 muestra un mensaje de texto, en este caso el uso de la función print no tiene mayor explicación, se trata de la funcionalidad habitual que tiene.

La línea 2 muestra un mensaje mediante la función input y queda a la espera de que el usuario introduzca alguna información y finalice con la pulsación de la tecla Intro (o Enter). Una vez que el usuario ha escrito su nombre, en este caso, la información introducida queda guardada en la variable **nombre** en formato **cadena de texto**.

La línea 3 vuelve a mostrar un mensaje, pero en esta ocasión a la función print se le han pasado varios parámetros separados por comas.

La función print cuando recibe varios parámetros separados por comas, los muestra introduciendo un espacio de separación entre los textos mostrados.

Nuestro programa generará una salida similar a la siguiente imagen:

```
>>> %Run saludo.py

   Este programa realiza un saludo
   Escribe tu nombre: Arturo
   Hola Arturo encantado de conocerte.

>>>
```

Es importante resaltar que la información introducida quedará guardada en la variable **nombre** en **formato texto**, incluso si se introduce un valor numérico.

¿Qué ocurre si quiero tratar la información introducida como un valor numérico?

Si leíste atentamente el capítulo 1, ya tendrás la respuesta.

Disponemos de varias funciones para convertir texto en su correspondiente valor numérico:

int Interpreta el texto introducido como un valor numérico **entero.**

float Interpreta el texto introducido como un valor numérico **decimal.**

eval Evalúa matemáticamente el texto introducido. **Es la forma más versátil.**

Ten en cuenta que si usas estas funciones y la expresión introducida no se puede convertir a dicho tipo numérico, en dicho caso el programa generará un error. Más adelante veremos como podemos capturar los errores para mostrar algún mensaje cuando se produzcan y que el programa no se interrumpa.

Como práctica del asunto que estamos tratando, vamos a hacer un sencillo programa donde apliquemos lo explicado en los últimos párrafos.

Programa: sumar2numeros.py

Queremos hacer un programa que nos pida 2 números y nos muestre la suma de dichos números.

1º El programa mostrará un mensaje indicando para que sirve.

2º Solicitará el primero de los números.

3º Solicitará el segundo de los números.

4º Mostrará los 2 números introducidos y el resultado de la suma.

Como es habitual en todos los programas, se desarrolla un bloque que interactúa con el usuario en donde se muestra información y se solicita información. Esto ya sabemos hacerlo con las funciones **print** e **input**, por lo que ya no prestaré mucha más atención a dichas funciones salvo que sea necesario aclarar algún aspecto.

Veamos el código de nuestro programa, más bien una de las formas de hacerlo. Ten en cuenta que generalmente siempre existirán varias formas de realizar un programa que cubra nuestro objetivo:

```
1     print('Programa para calcular la suma de 2 números')
2     n1 = input('Escribe el primero de los números: ')
3     n2 = input('Escribe el segundo de los números: ')
4     suma=eval(n1)+eval(n2)
5     print('La suma de', n1, 'y', n2, 'es', suma)
```

Al ejecutar nuestro programa, obtendremos una salida similar a esta:

```
>>> %Run suma2numeros.py

  Programa para calcular la suma de 2 números
  Escribe el primero de los números: 32
  Escribe el segundo de los números: 47
  La suma de 32 y 47 es 79

>>>
```

De nuevo y aunque el programa sea simple, es conveniente que todos los conceptos queden totalmente claros puesto que estamos estableciendo los cimientos para el futuro.

En la líneas 2 y 3, hemos solicitado al usuario que introduzca 2 números y se han almacenado en las variables n1 y n2, pero se han almacenado como cadenas de texto. Si en nuestro caso '32' y '47', concatenamos dichas cadenas, obtendríamos '3247'.

En la línea 4 hemos realizado la operación indicándole que lo haga de la siguiente forma:

Evalúa como un número la cadena de texto que contenga **n1**, evalúa también la cadena que contenga **n2** y suma el resultado que te hayan dado. Además, asigna el resultado de la suma a la variable **suma**.

En la línea 5 únicamente hemos construido una forma de mostrar el resultado, podríamos haberlo hecho también así: **print(n1,'+',n2,'=',suma)**

Aunque hemos comprobado que nuestro programa funciona correctamente, vamos a ver otras posibles maneras de resolverlo.

Las líneas 2 y 3 podríamos hacerlas escrito así:

```
2     n1 = eval(input('Escribe el primero de los números: '))
3     n2 = eval(input('Escribe el segundo de los números: '))
```

En esta ocasión le estamos diciendo que al mismo tiempo de solicitar los números, el texto introducido sea evaluado matemáticamente y asignado a las correspondientes variables.

De haberlo hecho así, las línea 4 tendría que haber sido:

```
4     suma = n1 + n2
```

Vamos a ver también una última forma de resolverlo:

```
1     print('Programa para calcular la suma de 2 números')
2     n1 = input('Escribe el primero de los números: ')
3     n1 = eval(n1)
4     n2 = input('Escribe el segundo de los números: ')
5     n2 = eval(n2)
6     suma = n1 + n2
7     print(n1,'+',n2,'=',suma)
```

Esta última forma es interesante por lo siguiente:

En la línea 2, introducimos una cadena de texto en la variable n1, por lo que su contenido es de tipo string.

En la línea 3, a n1 le asignamos el resultado de evaluar numéricamente la misma cadena de texto n1, por lo que a partir de este momento el tipo de dato contenido en n1 será un valor numérico.

Observa también que los valores no tienen porqué ser números enteros, pueden ser valores decimales siempre que usemos **el punto (.) como separador decimal**.

```
>>> %Run suma2numeros.py

  Programa para calcular la suma de 2 números
  Escribe el primero de los números: 3.5
  Escribe el segundo de los números: 2.8
  La suma de 3.5 y 2.8 es 6.3

>>>
```

Ejercicios:

Ejercicio 1:

Hacer un programa que solicite 2 números y a continuación muestre el resultado de la multiplicación de los números introducidos.

Ejercicio 2:

Hacer un programa que calcule potencias, es decir, solicite la base y el exponente, y después calcule el resultado de la base elevada al exponente.

Ejercicio 3:

Realiza un programa cuya salida sea similar a la que se muestra en la imagen siguiente.

```
Programa para realizar cálculos
Escribe un número: 23
Escribe un 2º número: 6
Operaciones con 23 y 6
23 + 6 = 29
23 - 6 = 17
23 x 6 = 138
23 / 6 = 3.8333333333333335
23 // 6 = 3 (División entera)
23 % 6 = 5 (Resto de la división)
```

Ejercicio 4:

Realiza un programa cuya salida sea similar a la que se muestra en la imagen siguiente.

```
Programa para calcular
el cuadrado y la raíz de un número
Escribe el número para los cálculos: 17
El cuadrado de 17 es 289
La raíz de 17 es 4.123105625617661
```

Ejercicio 5:

Realiza un programa cuya salida sea similar a la que se muestra en la imagen siguiente.

```
Programa que analiza una palabra
Escribe una palabra: Informática
La palabra Informática tiene 11 caracteres
La primera letra es I
La última letra es a
La 2 primeras letras son In
Las 2 últimas letras son ca
```

Ejercicio 6:

Realiza un programa cuya salida sea similar a la que se muestra en la imagen siguiente.

```
Programa que compara un número con el número 10
Escribe un número: 9
¿El número 9 es mayor o igual que 10? False
¿El número 9 esmenor que 10? True
```

Capítulo 3

Estructuras condicionales

```
n=input('Escribe un número: ')
n=eval(n)
if n%2==0:
    div2=n/2
    print('El número',n,'es divisible por 2.')
    print('Al dividir',n,'entre 2 se obtiene',div2)
elif n%3==0:
    div3=n/3
    print('El número',n,'es divisible por 3.')
    print('Al dividir',n,'entre 3 se obtiene',div3)
elif n%5==0:
    div5=n/5
    print('El número',n,'es divisible por 5.')
    print('Al dividir',n,'entre 5 se obtiene',div5)
```

Todos los lenguajes de programación disponen de estructuras que permiten evaluar expresiones cuyo resultado es True o False y dependiendo del resultado llevar a cabo un determinado bloque de instrucciones u otro.

A este tipo de estructuras se le llaman estructuras condicionales, evalúan una condición y cuando la condición resulta True ejecutan un determinado código y cuando resultan False pueden ejecutar otro bloque de código.

En Python estas estructuras vienen definidas con las palabras reservadas: **if**, **else** y **elif**.

Podemos encontrarnos con las siguientes construcciones:

Tipo 1

```
If condicion:
    instruccion1
    instruccion2
    …
```

Tipo 2

```
If condicion:
    bloque1-instruccion1
    bloque1-instruccion2
    …
else:
    bloque2-instruccion1
    bloque2-instruccion2
    …
```

Tipo 3

```
If condicion1:
    bloque1-instruccion1
    bloque1-instruccion2
    …
elif condicion2:
    bloque2-instruccion1
    bloque2-instruccion1
    …
elif condicion3:
    bloque3-instruccion1
    bloque3-instruccion2
    …
else:
    bloque4-instruccion1
    bloque4-instruccion2
    …
```

En todas las estructuras anteriores, la condición representa una expresión cuyo resultado será True o False.

Vamos a poner algunos ejemplos. Supongamos que en un programa tenemos una variable numérica, podríamos hacer condiciones de las formas:

n<10	Esta expresión sólo dará como resultado True cuando n contenga un valor inferior a 10.
n>=1000	Esta expresión sólo dará como resultado True cuando n contenga un valor mayor o igual que 1000.
n>10 and n<100	Sólo resultará True cuando n esté comprendido entre 10 y 100.

En la estructura **tipo 1**, el bloque de instrucciones sólo se ejecutará cuando el resultado de evaluar la condición sea True, si el resultado es False, no se ejecutará ningún código asociado a dicho caso.

Programa: condicionaltipo1.py

Vamos a hacer un programa que nos pida un número y a continuación nos diga si dicho número es múltiplo de 3 y nos muestre la división de ese número entre 3.

1º El programa mostrará unos mensajes informando lo que hace.

2º Solicitará un valor numérico.

3º Sólo si el valor introducido es múltiplo de 3, nos mostrará el resultado de dividir dicho número entre 3.

Veamos el código que hace esta funcionalidad. Como ya dijimos en ejemplos anteriores, existen varias formas de hacer el programa, nos limitaremos a hacerlo de una de ellas.

```
1     print('Programa para saber si un número es múltiplo de 3')
2     print('Si el número que escribes es divisible por 3, te mostraré')
3     print('el resultado de dividir dicho número entre 3.')
4     print('En caso contrario, no mostraré nada más.')
5     n = input('Escribe un número: ')
6     n = eval(n)
7     if n%3==0:
8         print('Enhorabuena, has escrito un número múltiplo de 3')
9         div = n/3
10        print('El resultado de dividir',n,'entre 3 es',div)
```

Observa la línea 7: te recuerdo que el operador % no calcula el resto de la dividir n entre 3 (en este caso).

Vamos a ver 2 posibles ejecuciones del programa, una introduciendo 21 (múltiplo de 3) y otra introduciendo 31 (que no es divisible por 3).

```
>>> %Run condicionaltipo1.py

  Programa para saber si un número es múltiplo de 3
  Si el número que escribes es divisible por 3, te mostraré
  el resultado de dividir dicho número entre 3.
  En caso contrario, no mostraré nada más.
  Escribe un número: 21
  Enhorabuena, has escrito un número múltiplo de 3
  El resultado de dividir 21 entre 3 es 7.0

>>>
```

```
>>> %Run condicionaltipo1.py

  Programa para saber si un número es múltiplo de 3
  Si el número que escribes es divisible por 3, te mostraré
  el resultado de dividir dicho número entre 3.
  En caso contrario, no mostraré nada más.
  Escribe un número: 31

>>>
```

En la estructura condicional **tipo 2**, el bloque 1 de instrucciones sólo se ejecutará si la condición resulta True, pero a diferencia de la estructura condicional tipo 1, en el caso de que dicha condición resulte False se ejecutaría el bloque 2 de instrucciones.

Programa: condicionaltipo2.py

Vamos a hacer un programa que nos pida un número y a continuación nos diga si dicho número es múltiplo de 3 y nos muestre la división de ese número entre 3.

1º El programa mostrará un mensaje informando lo que hace.
2º Solicitará un valor numérico.
3º Sólo si el valor introducido es múltiplo de 3, nos mostrará el resultado de dividir dicho número entre 3.
4º Si el valor introducido no es múltiplo de 3, el programa nos informará de ello y mostrará la división de dicho número entre 3.

Realizaremos algunas modificaciones más sobre el código anterior que debes comprender sin mayor dificultad

```
1     print('Programa para saber si un número es múltiplo de 3')
2     n = input('Escribe un número: ')
3     n = eval(n)
4     div = n/3
```

```
5     if n%3==0:
6         print('Enhorabuena, has escrito un número múltiplo de 3')
7         print('El resultado de dividir',n,'entre 3 es',div)
8     else:
6         print('El número que has escrito NO es múltiplo de 3')
7         print('Al dividir',n,'entre 3 se obtiene',div)
9         print('La división no es exacta.')
```

Vamos a ver nuevamente las 2 ejecuciones anteriores con este nuevo programa:

```
>>> %Run condicionaltipo2.py

   Programa para saber si un número es múltiplo de 3
   Escribe un número: 21
   Enhorabuena, has escrito un número múltiplo de 3
   El resultado de dividir 21 entre 3 es 7.0

>>>
```

```
>>> %Run condicionaltipo2.py

   Programa para saber si un número es múltiplo de 3
   Escribe un número: 31
   El número que has escrito NO es múltiplo de 3
   Al dividir 31 entre 3 se obtiene 10.333333333333334
   La división no es exacta.

>>>
```

En el **tipo 3** de estructuras condicionales podemos comprobar que entran en juego **varias condiciones**, tantas como consideremos oportuno, pero el programa sólo ejecutará el bloque de instrucciones de la primera condición que resulte True, es decir:

Si la condición 1 resulta True, ejecutará el bloque 1 de instrucciones y no seguirá evaluando las siguientes condiciones. Sólo seguirá evaluando la condición 2 en el caso de que la condición 1 resulte False.

Si la condición 2 resulta True, ejecutará el bloque 2 de instrucciones y no seguirá evaluando las siguientes condiciones. Por lo tanto, la condición 3 sólo será evaluada en el caso de que las condiciones 1 y 2 resulten False.

Siguiendo el mismo razonamiento, el bloque 4 de instrucciones sólo será ejecutado en el caso de que todas las condiciones evaluadas con anterioridad resulten False.

Programa: condicionaltipo3.py

Vamos a hacer un programa que nos pida un número y a continuación nos diga si dicho número es múltiplo de 2, en cuyo caso mostrará la división entre 2. Sólo en el caso de que el número no sea múltiplo de 2, comprobará si es múltiplo de 3 y sólo en caso afirmativo, nos mostrará la división de ese número entre 3. Si el número tampoco es múltiplo de 3, el programa comprobará si es múltiplo de 5, en cuyo caso mostrará la división del número introducido entre 5.

En el caso de que no sea múltiplo de 2, ni de 3, ni de 5, nos informará de ello.

1º El programa mostrará unos mensajes informando lo que hace

2º Solicitará un valor numérico

3º Si el valor introducido es múltiplo de 2, nos mostrará el resultado de dividir dicho número entre 2.

4º Si el valor introducido no es múltiplo de 2, pero sí es múltiplo de 3, el programa nos informará de ello y mostrará la división de dicho número entre 3.

5º Sólo si el número introducido no es múltiplo de 2, ni de 3, pero sí lo es de 5, en dicho caso, nos mostrará el resultado de dividirlo entre 5.

6º Si el número introducido no es múltiplo de 2, ni de 3, ni de 5, el programa nos informará de dicha circunstancia.

Antes de continuar con este ejemplo, vamos a aclarar que el objetivo es mostrar el mecanismo de las estructuras condicionales. Si deseamos encontrar los divisores de un número, esta manera de hacerlo no es la adecuada, precisamente ese ejercicio de calcular los divisores de un número lo resolveremos en el capítulo siguiente cuando estudiemos los bucles.

Vamos a escribir el código de nuestro programa para comprobar el funcionamiento de las estructura condiciones cuando hay varias condiciones:

```
1     print('Este programa te informará si un número es divisible por 2')
2     print('Si no es divisible por 2, comprobará si lo es por 3')
3     print('Si tampoco es divisible por 3, comprobará si lo es por 5')
4     print('Si no es divisible por 2, ni por 3, ni por 5, informará de ello')
5     n=input('Escribe un número: ')
6     n=eval(n)
7     if n%2==0:
8         div2=n/2
9         print('El número',n,'es divisible por 2.')
10        print('Al dividir',n,'entre 2 se obtiene',div2)
11    elif n%3==0:
12        div3=n/3
13        print('El número',n,'es divisible por 3.')
14        print('Al dividir',n,'entre 3 se obtiene',div3)
15    elif n%5==0:
16        div5=n/5
17        print('El número',n,'es divisible por 5.')
18        print('Al dividir',n,'entre 5 se obtiene',div5)
19    else:
20        print('El número no es múltiplo de 2, ni de 3, ni de 5.')
21        print('Prueba otra vez con otro número.')
```

Veamos el resultado de algunas ejecuciones del programa:

```
>>> %Run condicionaltipo3.py

  Este programa te informará si un número es divisible por 2
  Si no es divisible por 2, comprobará si lo es por 3
  Si tampoco es divisible por 3, comprobará si lo es por 5
  Si no es divisible por 2, ni por 3, ni por 5, informará de ello
  Escribe un número: 30
  El número 30 es divisible por 2.
  Al dividir 30 entre 2 se obtiene 15.0

>>>
```

```
>>> %Run condicionaltipo3.py

  Este programa te informará si un número es divisible por 2
  Si no es divisible por 2, comprobará si lo es por 3
  Si tampoco es divisible por 3, comprobará si lo es por 5
  Si no es divisible por 2, ni por 3, ni por 5, informará de ello
  Escribe un número: 45
  El número 45 es divisible por 3.
  Al dividir 45 entre 3 se obtiene 15.0

>>>
```

```
>>> %Run condicionaltipo3.py

  Este programa te informará si un número es divisible por 2
  Si no es divisible por 2, comprobará si lo es por 3
  Si tampoco es divisible por 3, comprobará si lo es por 5
  Si no es divisible por 2, ni por 3, ni por 5, informará de ello
  Escribe un número: 125
  El número 125 es divisible por 5.
  Al dividir 125 entre 5 se obtiene 25.0

>>>
```

```
>>> %Run condicionaltipo3.py

  Este programa te informará si un número es divisible por 2
  Si no es divisible por 2, comprobará si lo es por 3
  Si tampoco es divisible por 3, comprobará si lo es por 5
  Si no es divisible por 2, ni por 3, ni por 5, informará de ello
  Escribe un número: 61
  El número que has introducido no es múltiplo de 2, 3 o 5.
  Prueba otra vez con otro número.

>>>
```

Es muy probable que se te haya ocurrido hacer un programa que nos diga si un número es divisible por 2, si también los es por 3 y si también los es por 5, o alguna variante del él.

Vamos a hacer exactamente eso, un programa que nos pedirá un número y después nos informará si el número es divisible por 2, si es divisible por 3 y si es divisible por 5. Para ello vamos a usar varias estructuras condicionales. Es muy fácil.

Programa: multiplos235.py

```
1     print('Este programa te dice si un número es múltiplo de 2, 3 o 5')
2     n=input('Escribe un número: ')
3     n=eval(n)
4
5     if n%2==0:
6         div2=n/2
7         print('El número',n,'es múltiplo de 2')
8         print('Al dividir',n,'entre 2 se obtiene',div2)
9     else:
10        print('El número', n, 'no es múltiplo de 2')
11
12    if n%3==0:
13        div3=n/3
14        print('El número',n,'es múltiplo de 3')
15        print('Al dividir',n,'entre 3 se obtiene',div3)
16    else:
17        print('El número', n, 'no es múltiplo de 3')
18
19    if n%5==0:
20        div5=n/5
21        print('El número',n,'es múltiplo de 5')
22        print('Al dividir',n,'entre 5 se obtiene',div5)
23    else:
24        print('El número', n, 'no es múltiplo de 5')
```

En este ejemplo hemos dejado algunas líneas vacías únicamente para que código resulte más fácil de leer, pero no es necesario dejar dichas líneas en blanco. De todas formas, cuando los programas crecen en tamaño, verás que es conveniente dejar líneas en blanco y añadir comentarios para comprender mejor el código en un futuro.

Veamos una posible salida del programa:

```
>>> %Run multiplo235.py

  Este programa te dice si un número es múltiplo de 2, 3 o 5
  Escribe un número: 12
  El número 12 es múltiplo de 2
  Al dividir 12 entre 2 se obtiene 6.0
  El número 12 es múltiplo de 3
  Al dividir 12 entre 3 se obtiene 4.0
  El número 12 no es múltiplo de 5

>>>
```

Vamos a terminar este capítulo dedicado a las estructuras condicionales haciendo un último ejemplo en que vamos a **anidar** varias estructuras condiciones.

Anidar estructuras significa incluir unas estructuras dentro de otras.

Programa: anidarcondicionales.py

Vamos a hacer un programa que nos pida el nombre del usuario y compruebe la longitud en caracteres del nombre introducido. No dirá si el nombre es largo o se trata de un nombre corto y además en el caso de que el nombre sea largo, nos debe decir si el nombre es compuesto o no.

El programa mostrará un mensaje indicando lo que hace.

Solicitará el nombre del usuario.

Contará el número de caracteres del nombre.

Si el nombre tiene 7 caracteres o menos, sólo contará mostrará el numero de caracteres del nombre.

Si el nombre tiene más de 7 caracteres, comprobará además si contiene algún espacio.

En el caso de que contenga algún espacio dirá que el nombre es compuesto.

Si no contiene ningún espacio, dirá que el nombre no es compuesto.

El código del programa es el siguiente:

```
1     print('Este programa contará los caracteres de tu nombre')
2     nombre = input('Por favor, escribe tu nombre: ')
3     longitud = len(nombre)
4     print('Hola',nombre,'encantado de saludarte')
5
6     if longitud<=7:
7         print('Tu nombre tiene',longitud,'caracteres')
8     else:
9         print('Tienes un nombre largo,',longitud,'caracteres')
10        if ' ' in nombre:
11            print('Además, tienes un nombre compuesto')
12        else:
13            print('Además, no se trata de un nombre compuesto')
```

Vamos a explicar algunas líneas:

En la línea 3, con la función **len()** contamos el número de caracteres que tiene la variable de texto nombre, y además asignamos dicho número a la variable **longitud**.

En la línea 6 comparamos la longitud con el número 7.

En la línea 10, comprobamos si el carácter **Espacio** (' ') se encuentra dentro de la variable nombre, para ello hemos usado la palabra clave **in** que hace exactamente eso.

Observa que el condicional de la línea 10, se encuentra dentro del else de la línea 8, en esto consiste la anidación de bloque de estructuras condicionales, en incluir unos bloques dentro de otros.

Ejercicios:

Ejercicio 1:

Escribe un programa que genere las siguientes salidas en su ejecución.

```
Programa sobre números pares
Escribe un número que sea par: 24
¡¡¡ CORRECTO !!!
El número 24 es un número par
24 / 2 = 12.0
```

```
Programa sobre números pares
Escribe un número que sea par: 31
¡¡ Oh, lo siento !!
El número 31 no es un número par
31 / 2 = 15.5
```

Ejercicio 2:

Escribe un programa que genere las siguientes salidas en su ejecución.

```
Programa para comprobar si un número es
divisible o no por 2, 3, 5, 7 y 11
Escribe un número: 132
El número 132 SI es divisible por 2
El número 132 SI es divisible por 3
El número 132 NO es divisible por 5
El número 132 NO es divisible por 7
El número 132 SI es divisible por 11
```

Ejercicio 3:

El programa comprobará si la palabra tiene 5 letras o menos, en cuyo caso mostrará la primera y la última letra.

En el caso de que tenga más de 5 letras, mostrará las 4 primeras letras por un lado y las 4 últimas letras por otro, aunque haya letras coincidentes en las 2 particiones.

El programa debe generar las siguientes salidas en su ejecución.

```
Programa que analiza una palabra
Escribe una palabra: Hola
La palabra Hola tiene 4 letras
La primera letra es: H
La última letra es: a
```

```
Programa que analiza una palabra
Escribe una palabra: Saludos
La palabra Saludos tiene 7 letras
Las 4 primeras letras son: Salu
Las 4 últimas letras son: udos
```

Ejercicio 4:

Realizar un programa que calcule la letra del Número de Identificación Fiscal (N.I.F.).

El programa debe solicitar el número del Documento Nacional de Identidad y a partir de dicho número calcule la letra correspondiente.

Consultar la página: http://www.interior.gob.es/web/servicios-al-ciudadano/dni/calculo-del-digito-de-control-del-nif-nie

Nota: Sólo calcular el caso de NIF españoles.

Capítulo 4

Bucles
Estructuras repetitivas

```
entrada='algo'
contador=0
while entrada!='':
    entrada=input('Escribe un texto: ')
    contador=contador+1
else:
    contador=contador-1
```

Los ordenadores se caracterizan por la rapidez en la realización de sus operaciones, de ahí que las estructuras repetitivas adquieran una gran importancia en todos los lenguajes de programación, sirven para poder realizar un gran número de operaciones escribiendo muy pco código, con lo cual nos ahorran muchísimo trabajo.

Las estructuras repetitivas reciben el nombre de bucles, y en Python, al igual que en otros muchos lenguajes de programación, encontramos 2 tipos de construcciones, las que podemos llevar a cabo con la palabra reservada **for** y las que podemos realizar con la palabra reservada **while**.

Estructura **for**

```
for variable in objeto-iterable:
    instruccion-1
    instruccion-2
    instruccion-3
    ...
```

Estructura **while**

```
While condicion:
    instruccion-1
else:
    instruccion-2
    ...
```

Estructura for

Un bucle for consta de una variable, un objeto iterable y un bloque de código.

El concepto nuevo en esta ocasión es el objeto iterable, se trata de un conjunto de valores que la variable irá tomando en cada una de las veces que ejecutará el bloque de código.

En Python tenemos muchos objetos iterables y además podemos construir los nuestros propios. Los más comunes son: listas, cadenas de textos, tuplas, …

La mejor forma de comprender como funciona un bucle for es haciendo algunos ejemplos:

Programa: objetoiterable1.py

```
1     for n in (1,2,3,4,5,6,7,8,9,10):
2     print('El valor de n es', n)
```

```
>>> %Run objetoiterable1.py

  El valor de n es 1
  El valor de n es 2
  El valor de n es 3
  El valor de n es 4
  El valor de n es 5
  El valor de n es 6
  El valor de n es 7
  El valor de n es 8
  El valor de n es 9
  El valor de n es 10

>>>
```

Lo que ha ocurrido ha sido lo siguiente:

La variable n ha tomado el valor n=1 y ha ejecutado el bloque de código que en esta ocasión es una única instrucción.

La variable n ha tomado el valor n=2 y ha vuelto a ejecutar el bloque de código.

Así sucesivamente hasta recorrer todos los valores del objeto iterable.

Veamos otro ejemplo simple de bucle for recorriendo otro iterable, en el siguiente caso el iterable será una lista de nombres.

Programa: objetoiterable2.py

```
1     nombres = ['Arturo', 'Julio', 'Sara', 'Juan', 'Roberto', 'Dani',
'Martina', 'Aurora']
2     for n in nombres:
3         print('El valor de n es',n)
4         print(n,'tiene',len(n),'caracteres\n')
```

En esta ocasión el programa genera la siguiente salida, que pasamos a explicar:

```
>>> %Run objetoiterable2.py

   El valor de n es Arturo
   Arturo tiene 6 caracteres

   El valor de n es Julio
   Julio tiene 5 caracteres

   El valor de n es Sara
   Sara tiene 4 caracteres

   El valor de n es Juan
   Juan tiene 4 caracteres

   El valor de n es Roberto
   Roberto tiene 7 caracteres

   El valor de n es Dani
   Dani tiene 4 caracteres

   El valor de n es Martina
   Martina tiene 7 caracteres

   El valor de n es Aurora
   Aurora tiene 6 caracteres

>>>
```

Como se puede observar el objeto iterable lo hemos definido en la línea 1 en donde hemos declarado una variable nombres que contiene una lista de nombres propios.

En la línea 2 definimos el bucle, donde la variable n recorre la lista de nombres.

En la línea 3 mostramos el valor que va tomando la variable n en cada una de las iteraciones del bucle.

En la línea 4, mostramos el valor de n, la longitud de la cadena de caracteres n, y además añadimos un salto de línea para que al ejecutar el programa visualmente se observe con más claridad cada una de las iteraciones del bucle for.

Como la lista nombres tiene 8 elementos, el programa ha ejecutado el bloque de instrucciones (líneas 3 y 4) 8 veces, una por cada valor de las lista de nombres.

Programa: objetoiterable3.py

```
1     print('Este programa deletrea una palabra,')
2     print('escribe cada uno de sus caracteres y el valor unicode del
carácter')
3     palabra=input('Escribe una palabra: ')
4     for c in palabra:
5         print(c,'valor unicode:',ord(c),'\n')
```

Este programa genera la salida siguiente:

```
>>> %Run objetoiterable3.py

   Este programa deletrea una palabra,
   escribe cada uno de sus caracteres y el valor unicode del carácter
   Escribe una palabra: Informática
   I valor unicode 73

   n valor unicode 110

   f valor unicode 102

   o valor unicode 111

   r valor unicode 114

   m valor unicode 109

   á valor unicode 225

   t valor unicode 116

   i valor unicode 105

   c valor unicode 99

   a valor unicode 97

>>>
```

En el bucle for la **variable es c** y va tomando cada uno de los caracteres que hayamos introducido a través de la instrucción input y almacenado en la variable **palabra**.

Como se puede observar en ejemplo, una cadena de texto también es un objeto iterable, siendo cada uno de los caracteres los elementos que se iteran en cada uno de los ciclos.

En la línea 5 hemos utilizado la función ord() que nos devuelve el valor unicode del carácter que se le pasa como parámetro.

Vamos a finalizar estos ejemplos de bucles for mostrando el uso de la función range(), la cual es muy común encontrarla en la línea de declaración del bucle.

La función **range** tiene la siguiente sintaxis:

range(valorinicial, valorfinal, incremento)

Esta función nos devuelve un objeto iterable que incluye valorinicial como primer elemento y llega hasta una unidad menos de valorfinal.

Incremento es un parámetro opcional que indica el incremento de la variable del bucle en cada iteración. Si se omite el incremento, por defecto se establece a 1.

Ejemplos:

range(1,10)	Obtendríamos como objeto iterable: 1, 2, 3, 4, 5, 6, 7, 8, 9
range(1,20,3)	Obtendríamos como iterable: 1, 4, 7, 10, 13, 16, 19
range(10, 1, -1)	Obtendríamos como iterable: 10, 9, 8, 7, 6, 5, 4, 3, 2

Programa: objetoiterable4.py

Vamos a realizar un programa que calcule las tablas de multiplicar. El programa preguntará qué tabla quieres calcular.

1º El programa mostrará un mensaje indicando su funcionalidad.

2º Preguntará la tabla que quieres que se calcule.

3º Mostrará la tabla de multiplicar correspondiente.

```
1     print('Este programa calcula las tablas de multiplicar')
2     numero=input('Qué tabla de multiplicar quieres calcular: ')
3     numero=eval(numero)
4     for n in range(1,11):
5         print(n ,'x', numero, '=', n*numero)
```

La salida que genera el programa es:

```
>>> %Run objetoiterable4.py

  Este programa calcula las tablas de multiplicar
  Qué tabla de multiplicar quieres calcular: 8
  1 x 8 = 8
  2 x 8 = 16
  3 x 8 = 24
  4 x 8 = 32
  5 x 8 = 40
  6 x 8 = 48
  7 x 8 = 56
  8 x 8 = 64
  9 x 8 = 72
  10 x 8 = 80

>>>
```

Recuerda evaluar numero como un valor numérico, de lo contrario será interpretado como una cadena de caracteres.

Observa que en la definición del bucle hemos indicado range(1,11) para que estén incluidos desde el 1 hasta el 10 en el conjunto de valores iterables.

En la línea 5 visualizamos el signo 'x', aunque la operación se multiplicar se calcula con el operador multiplicación (*).

Estructura while

Un bucle while ejecuta un bloque de instrucciones hasta que la condición deja de ser True.

Los bucles while, al igual que los bucles for, son muy comunes y versátiles.

La cláusula **else** y el bloque de instrucciones contenidos en él, **es opcional** y se ejecutará cuando la condición deje de ser válida.

Programa: buclewhile1.py

Vamos a realizar un programa que permita introducir números de una cifra. Cuando el número escrito sea mayor o igual que 10, será equivalente a terminar.

> 1º El programa mostrará un mensaje indicando su funcionalidad.
>
> 2º Pedirá que se introduzca un número.
>
> 3º Debe estar pidiendo números **mientras** los números sean menores que 10.

```
1     print('Escribe números de una sola cifra')
2     print('Más de una cifra, será equivalente a terminar')
3     numero=0
4     while numero<10:
5         numero=input('Escribe un número: ')
6         numero=eval(numero)
```

Al ejecutar el programa podemos obtener una salida similar a esta:

```
>>> %Run buclewhile1.py

   Escribe números de una sola cifra
   Más de una cifra, será equivalente a terminar
   Escribe un número: 3
   Escribe un número: 5
   Escribe un número: 4
   Escribe un número: 3
   Escribe un número: 10

>>>
```

La construcción del bucle while incluye la condición numero<10, es decir, **el bucle estará repitiendo el bloque de instrucciones mientras que el valor de la variable número sea menor que 10**.

La línea 3 es necesaria para que la primera vez que se ejecute el bucle, número tenga algún valor y pueda compararlo con el número 10.

El mecanismo de **uso habitual de los bucles while**, siempre es el siguiente:

- En la condición del bucle while entran en juego una o más variables que han sido declaradas previamente.
- Dentro del bloque de instrucciones, algunas de las variables que forman parte de la condición son modificadas según el objetivo del programa.
- El bloque de instrucciones se repite mientras que las condición resulte verdadera.

Vamos a ver un nuevo ejemplo de bucle while en el que usaremos una técnica habitual en programación que consiste en:

1º Definir una variable de control del bucle con valor True. Esta variable será la condición del bucle.

2º Dentro del bucle realizaremos alguna comprobación con una estructura condicional que servirá para modificar el valor de la variable de control que hemos definido en el paso anterior.

3º El bloque de instrucciones del bucle se estará repitiendo mientras no cambie la variable de control.

Vamos a ver esta técnica con un sencillo ejemplo. Haremos un programa que estará pidiendo números pares indefinidamente hasta que se introduzca un número impar. El programa finalizará cuando se introduzca el primer número impar.

Programa: buclewhile2.py

1º El programa mostrará un mensaje indicando su funcionalidad.

2º Definiremos una variable, seguir=True que será la que controlará la condición del bucle.

3º Pedirá que se introduzcan números pares.

4º Cuando se introduzca un número impar se cambiará el valor de seguir a False.

```
1     print('Escribe números pares')
2     print('Un número impar servirá para terminar')
3     seguir=True
4     while seguir:
5         numero=input('Escribe un número: ')
6         numero=eval(numero)
7         if numero%2==1:
8             seguir=False
```

Una posible ejecución de este programa sería:

```
>>> %Run buclewhile2.py

   Escribe números pares
   Un número impar servirá para terminar
   Escribe un número: 4
   Escribe un número: 6
   Escribe un número: 12
   Escribe un número: 334
   Escribe un número: 7

>>>
```

Bucles infinitos

Se llaman bucles infinitos a bucles que nunca finalizan y por tanto se están ejecutando indefinidamente. Se debe tener especial cuidado porque algunos bucles infinitos podrían agotar los recursos del equipo.

Cuando se está aprendiendo a programar, es habitual cometer errores de lógica en la elaboración de bucles while y generar bucles infinitos.

Técnicas comunes de programación: contadores y acumuladores

Dentro de un bucle es muy habitual encontrarnos una o varias instrucciones como las siguientes:

Contador	**Acumulador**
n = n + 1	**suma = suma + n**

Veamos algunos ejemplos donde podemos encontrar estas instrucciones:

Vamos a hacer un programa que nos pida introducir números hasta que una entrada se deje vacía. El programa debe ir contando cuántos números se introducen y la suma de esos números.

Para ello vamos a usar varias variables que nos permitan el control y guardar la información que en este caso nos interesa.

Programa: contador-acumulador1.py

1º El programa mostrará un mensaje indicando su funcionalidad.

2º Definiremos 3 variables, **seguir**=True que será la que controlará la condición del bucle, **n** que será el contador de las veces que se ejecuta el bucle y **suma** que será donde acumularemos la suma de los números que se introducen.

3º Pedirá que se introduzcan números.

4º Cuando se pulse Intro sin escribir ningún número, el programa finalizará (seguir=False).

5º Si se introduce un número, se aumenta en una unidad la variable **n** (contador) y el valor del número introducido se suma al valor que tenga en ese momento la variable **suma**, es decir, se acumula en la variable **suma**.

```
1     print('Programa que pedirá números hasta dejar una entrada vacía')
2     print('El programa cuenta cuántos números se han introducido')
3     print('y la suma de todos esos números introducidos')
4     print('(Entrada vacía = Terminar)')
5     n=0
6     suma=0
7     seguir=True
8     while seguir:
9         numero=input('Escribe un número: ')
10        if numero=='':
11            seguir=False
12        else:
13            numero=eval(numero)
14            n=n+1
15            suma=suma+numero
16
17    print('Se han introducido',n,'números')
18    print('Los números suman',suma)
```

Con este programa podríamos obtener una salida similar a la siguiente imagen:

```
>>> %Run contador-acumulador1.py

  Programa que pedirá números hasta dejar una entrada vacía
  El programa cuenta cuántos números se han introducido
  y la suma de todos esos números introducidos
  (Entrada vacía = Terminar)
  Escribe un número: 4
  Escribe un número: 15
  Escribe un número: 31
  Escribe un número: 20
  Escribe un número: 8
  Escribe un número:
  Se han introducido 5 números
  Los números suman 78

>>>
```

Ejercicios

Ejercicio 1:

Realizar un programa que muestre **sólo los números pares** comprendidos entre 1 y 1000.

Ejercicio 2:

Realizar un programa que muestre **sólo los números impares** comprendidos entre 1 y 1000.

Ejercicio 3:

Realizar un programa que muestre todos los divisores de un número (se entiende que el número será entero positivo).

Ejercicio 4:

Realizar una programa que solicite una palabra y sólo muestre las vocales de dicha palabra.

Ejercicio 5:

Realizar una programa que solicite una palabra y sólo muestre las consonantes de dicha palabra.

Ejercicio 6:

Realizar un programa que solicite entradas de texto hasta que se deje una de las entradas de texto vacía. Las palabras se irán incluyendo en una lista.

Ejercicio 7:

Realizar un programa que solicite un número y muestre todos los divisores de ese número y además cuente el número de divisores de dicho número.

Nota: Aplicar el concepto de contador, contador=contador+1 cada vez que se encuentre un divisor.

Ejercicio 8:

Realizar un programa que solicite una palabra, genere una lista a partir de dicha palabra y muestre la lista.

```
Programa que solicta una palabra
y la convierte en una lista
Escribe una palabra: Instituto
['I', 'n', 's', 't', 'i', 't', 'u', 't', 'o']
```

Ejercicio 9:

Realizar un programa que solicite una palabra, genere una lista a partir de dicha palabra y muestre la lista en orden inverso.

```
Programa que solicta una palabra
y la convierte en una lista en orden inverso
Escribe una palabra: Instituto
['o', 't', 'u', 't', 'i', 't', 's', 'n', 'I']
```

Ejercicio 10:

Realizar un programa que solicite una palabra y deletree dicha palabra en orden inverso. La salida del programa podría ser algo así.

```
Programa que solicta una palabra
y deletrea dicha palabra en orden inverso
Escribe una palabra: Insituto
o
t
u
t
i
s
n
I
```

Ejercicio 11:

Realizar un programa que solicite entradas de números hasta que se deje una de las entradas se deje vacía. El programa debe contar cuántos números se han introducido, calcular la suma de los números y la media.

Capítulo 5

Módulos

math

random

time

datetime

...

Hasta este momento hemos estado usando las palabras claves que forman el esqueleto básico de Python, así como algunas otras funciones específicas que hacen algunas tareas también de carácter general. Sin embargo, Python es un lenguaje de programación tremendamente rico en funcionalidades disponibles a través de módulos que tenemos a nuestra disposición para usar en cualquier momento. Si se quiere obtener un listado exhaustivo de los módulos disponibles, lo más adecuado es consultar la documentación en la web oficial de Python (www.python.org).

Por defecto, cualquier instalación de Python ya trae un elevado número de módulos que nos permitirán realizar todo tipo de tareas, pero además podemos instalar módulos desarrollados por terceros para añadir aún más funcionalidad a nuestra instalación.

Para comenzar y familiarizarnos con el uso de los módulos vamos a ver qué son, como se usan y cuales son los más habituales cuando uno se está iniciando con Python.

¿Qué son los módulos?

Módulos, librerías, bibliotecas, …, estos términos suelen ser muy parecidos y su objetivo es poner a disposición del programador una serie de herramientas que le faciliten la realización de algunas tareas específicas.

Los módulos suelen ser colecciones de funciones, constantes, clases, objetos, ... que tenemos agrupados bajo un mismo nombre.

Ejemplo:

En el módulo math, hay disponible funciones para la realización de cálculos matemáticos como logaritmos, exponenciales, raíces, redondeos, trigonometría, m.c.d., etc.

¿Cómo se usan los módulos?

Tenemos 2 formas de usar los módulos, dependiendo del número de funciones que vayamos a usar y dependiendo de que no existan interferencias entre diferentes módulos que puedan coincidir los nombres de algunas de las funciones que contengan.

Forma 1: import math

Al usar el módulo math de esta manera estamos importando **todas las funciones** que contiene el módulo y por tanto a partir de este momentos todas están disponibles **anteponiéndoles el nombre del módulo**.

Ejemplo:

math.sqrt(9)	De esta forma podríamos calcular la raíz cuadrada de 9.
math.gcd(60,48)	Calculamos el M.C.D. de 60 y 48.

Forma 2: from math import sqrt, gcd

De esta segunda forma, sólo importamos las funciones sqrt y gcd, es decir, sólo importamos las funciones enumeradas en la importación. Además, en este caso no es necesario anteponer el nombre del módulo para poder hacer uso de la función.

Ejemplo:

sqrt(9)	De esta forma podríamos calcular la raíz cuadrada de 9.
math.gcd(60,48)	Calculamos el M.C.D. de 60 y 48.

Una variante de esta segunda forma es: **from math import *** así estamos importando todas las funciones y podemos usarlas directamente sin necesidad de anteponer el nombre del módulo.

Módulos comunes y de uso frecuente

Hablar de módulos frecuentes no tiene mucho sentido puesto que se usan los módulos en función de las tareas que se quieren hacer, por lo tanto, la expresión módulos comunes sólo va a tener sentido en el ámbito de la aplicación que le daremos, que en nuestro caso es un uso académico en los niveles que trata este libro.

Como ya se adelanta en el nombre del capítulo, trabajaremos con los módulos: math, random, time y datetime. Más adelante veremos otros módulos en función de las necesidades que vayan surgiendo.

El módulo math

```
import math
math.
```

```
acos
acosh
asin
asinh
atan
atan2
atanh
```

Como su propio nombre indica, incluye funciones para realizar operaciones matemáticas.

Podemos encontrar una amplia descripción de cada una de las funciones y constantes que incluye este módulo en la documentación de Python, nosotros vamos a usar algunas de ellas que son bastante habituales.

Programa: operaciones-varias1.py

Vamos a hacer un programa que nos genere una tabla con valores y su raíz cuadrada (sqrt). Calcularemos esta operación para los número del 1 al 10.

1º Realizaremos la importación del módulo

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Generaremos un bucle para construir la tabla

4º Realizaremos los cálculos necesarios

```
1     import math
2     print('Este programa calculará las raíz cuadrada,')
3     print('de los números: 1 al 10')
4     for x in range(1,11):
5         print(x,math.sqrt(x))
```

La salida del programa debe ser algo similar a la siguiente imagen:

```
>>> %Run operaciones-varias1.py

  Este programa calculará las raíz cuadrada,
  de los números: 1 al 10
  1 1.0
  2 1.4142135623730951
  3 1.7320508075688772
  4 2.0
  5 2.23606797749979
  6 2.449489742783178
  7 2.6457513110645907
  8 2.8284271247461903
  9 3.0
  10 3.1622776601683795

>>>
```

Programa: mcd-mcm.py

Vamos a hacer un programa que nos calcule el máximo común divisor y el mínimo común múltiplo de 2 números que le introduzcamos al programa.

Nota: Vamos a recordar una propiedad sobre mcm y mcd, que es la siguiente:

mcm(x,y)*mcd(x,y)=x*y

1º Realizaremos la importación del módulo

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Solicitará los números

4º Realizaremos los cálculos necesarios

5º Mostraremos los resultados

```
1     import math
2     print('Este programa calculará el mcd y mcm,')
3     print('de los 2 números que introduzcamos')
4     n1=input('Escribe el número 1: ')
5     n1=eval(n1)
6     n2=input('Escribe el número 2: ')
7     n2=eval(n2)
8     mcd=math.gcd(n1,n2)
9     mcm=(n1*n2)/mcd
10    print('El Máximo Común Divisor es',mcd)
11    print('El Mínimo Común Múltiplo es',mcm)
```

Este programa generará la siguiente salida:

```
>>> %Run mcd-mcm.py

  Este programa calculará el mcd y mcm,
  de los 2 números que introduzcamos
  Escribe el número 1: 60
  Escribe el número 2: 48
  El Máximo Común Divisor es 12
  El Mínimo Común Múltiplo es 240.0

>>>
```

La librería math pone a disposición algunas constantes como son los números pi y e.

La forma de acceder a ellas es anteponiendo el nombre del módulo, es decir, math.pi y math.e nos dará los valores que observamos en la imagen:

```
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>>
```

Programa: suma-iterable.py

Vamos a hacer un programa que nos calcule la suma de un conjunto de valores. Para simplificar el programa vamos a suponer que los valores ya los tenemos datos previamente.

Este programa se puede hacer con un bucle y un acumulador: suma = suma + valor, pero vamos a usar la función fsum del módulo math que facilita esta operación.

1º Realizaremos la importación de la función fsum del módulo math

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Realizaremos los cálculos necesarios

4º Mostraremos los resultados

```
1    from math import fsum
2    print('Este programa realiza la suma de un conjunto de valores')
3    valores=(2,4,12,5,31,45,32,13,15)
4    print(valores)
5    suma=fsum(valores)
6    print('La suma de los números es',suma)
```

Observa que en esta ocasión sólo hemos importado la función fsum del módulo math. Al hacer la importación de este modo, podemos usar directamente la función sin necesidad de anteponer el nombre del módulo.

La salida de nuestro programa debería ser algo similar a:

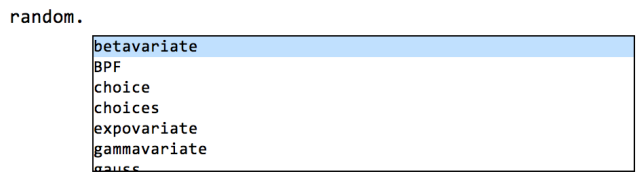
```
>>> %Run suma-iterable.py

  Este programa realiza la suma de un conjunto de valores
  (2, 4, 12, 5, 31, 45, 32, 13, 15)
  La suma de los números es 159.0

>>>
```

El módulo random

El módulo random nos permite disponer de herramientas para la generación de resultados aleatorios dentro de nuestros programas.

El uso de valores aleatorios es algo muy frecuente en la programación puesto que permite que el programa reaccione de diferentes formas ante una misma situación. Estas herramientas son fundamentales en la elaboración de juegos, simulación de escenarios, pruebas, generación de datos, etc.

Como ya dijimos con el módulo math, para conocer todas las funciones disponibles en el módulo random, lo mejor es consultar la documentación en la web www.python.org.Nosotros veremos unos ejemplos con algunas de las funciones.



Veamos con un ejemplo algunas de las funciones que resultan útiles dentro del módulo random.

Random.random()	Genera un número decimal aleatorio entre 0 y 1.
random.randint(a,b)	Genera un número entero aleatorio entre a y b.
random.uniform(a,b)	Genera un número decimal aleatorio entre a y b.
random.choice(conjunto)	Elige un elemento del conjunto pasado como argumento.

Programa: aleatorio1.py

Vamos a hacer un programa que genere un número decimal entre 0 y 1; un número entero entre 10 y 100; un número decimal entre 10 y 20; y por último elija un nombre de una lista de nombres que le pasaremos como parámetros.

1º Realizaremos la importación del módulo random

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Realizaremos realizaremos la generación que hemos indicado en el párrafo anterior

4º La lista es nombres=['Arturo','Julio','Dani','Aurora','Roberto','Martina','Hugo']

5º Mostraremos los resultados

```
1     import random
2     print('Programa para generar algunos datos aleatorios')
3     v1=random.random()
4     print('Valor aleatorio entre 0 y 1:',v1)
5     v2=random.randint(10,100)
6     print('Valor entero aleatorio entre 10 y 100:',v2)
7     v3=random.uniform(10,20)
8     print('Valor decimal entre 10 y 20:',v3)
9     nombres=['Arturo','Julio','Dani','Aurora','Roberto','Martina','Hugo']
10    nom=random.choice(nombres)
11    print('El nombre elegido es:',nom)
```

El programa debe realizar una salida similar a esta:

```
>>> %Run aleatorio1.py

   Programa para generar algunos datos aleatorios
   Valor aleatorio entre 0 y 1: 0.6803400712801796
   Valor entero aleatorio entre 10 y 100: 21
   Valor decimal entre 10 y 20: 12.346800023462455
   El nombre seleccionado es: Julio

>>>
```

Programa: aleatorio2.py

Vamos a hacer un programa que genere números aleatorios enteros entre 1 y 10, los vaya imprimiendo y finalice cuando salga el número 10.

1º Realizaremos la importación de la función randint del módulo random

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Realizaremos un bucle que se ejecutará mientras la generación sea distinta de 10

4º Dentro del mismo bucle iremos mostrando los valores generados

```
1     from random import randint
2     print('Programa que generará número aleatorios')
3     print('entre 0 y 10 hasta que salga el número 10')
```

```
4       n=0
5       while n!=10:
6           n=randint(1,10)
7           print('Ha salido el número:',n)
```

```
>>> %Run aleatorio2.py

  Programa que generará número aleatorios
  entre 0 y 10 hasta que salga el número 10
  Ha salido el número: 5
  Ha salido el número: 3
  Ha salido el número: 9
  Ha salido el número: 4
  Ha salido el número: 5
  Ha salido el número: 3
  Ha salido el número: 5
  Ha salido el número: 7
  Ha salido el número: 5
  Ha salido el número: 3
  Ha salido el número: 10

>>>
```

El programa debería mostrar una salida similar a esta:

El módulo time

El módulo time nos va a servir para manejar fechas, horas, medir tiempos, etc.

Generalmente en la mayoría de los programas es necesario introducir datos relacionados con el momento, la fecha o la hora en el que se producen determinados actos. Para estas ocasiones podemos usar este módulo que aunque tiene muchas opciones, las más comunes las vamos a utilizar en los siguientes ejemplos.

Antes de comenzar vamos a aclarar una cuestión. Python al igual que otros muchos lenguajes de programación utiliza el concepto de marca de tiempo (timestamp), con na precisión de una millonésima de segundo (no está nada mal), para esto se basa en el reloj interno que tienen nuestros equipos. Esta instante (timestamp), Python lo gestiona internamente como un valor numérico.

```
>>> import time
>>> time.time()
1516882907.320448
```

En el momento del diseño de Python, se acordó gestionar establecer el momento 0 en el **1 de enero de 1970 as las 00:00:00**. Por lo tanto, el número que obtenemos al ejecutar la función time() del módulo time, representa la cantidad de segundos transcurridos desde el momento 0. La precisión llega hasta la millonésima parte de un segundo.

También podemos obtener el momento actual con una cadena de texto y como una estructura de tiempo.

Veamos con un ejemplo como obtener la fecha y momento actual de cada una de estas formas:

Programa: tiempos1.py

Vamos a hacer un programa que mostrará la fecha y hora en diferentes formatos.

1º Realizaremos la importación del módulo time

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Vamos a utilizar las funciones: time, ctime, gmtime y localtime.

4º Dentro del mismo bucle iremos mostrando los valores generados

```
1       import time
2       print('Obtener la fecha y hora actual en diferentes formatos')
3       print('Tiempo en segundos:',time.time())
4       print('Fecha y hora como un texto:',time.ctime())
5       print('Estructura tiempo (UTC):',time.gmtime())
6       print('Estructura tiempo (local):',time.localtime())
```

Obtenemos el siguiente resultado:

```
>>> %Run tiempos1.py

   Obtener la fecha y hora actual en diferentes formatos
   Tiempo en segundos: 1516896310.2412071
   Fecha y hora como un texto: Thu Jan 25 17:05:10 2018
   Estructura tiempo (UTC): time.struct_time(tm_year=2018, tm_mon=1, tm_mday=25, tm_hour=16, tm_min=5,
   tm_sec=10, tm_wday=3, tm_yday=25, tm_isdst=0)
   Estructura tiempo (local): time.struct_time(tm_year=2018, tm_mon=1, tm_mday=25, tm_hour=17, tm_min=5
   , tm_sec=10, tm_wday=3, tm_yday=25, tm_isdst=0)

>>>
```

Como puedes observar en el ejemplo, las funciones gmtime() y localtime() sólo diferencian la hora dependiendo del huso horario en el que nos encontremos. UTC es el huso horario en el meridiano de Greenwich o en el meridiano cero.

Generalmente vamos a preferir tener la fecha y hora en un formato diferente, para ello el módulo time de Python facilita una función strftime(), que permite varios parámetros para construir cualquier cadena de texto representando la fecha. Por ejemplo:

strftime('%d/%m/%y',localtime()) nos serviría para obtener la fecha en el formato habitual 25/01/18

%d	Para mostrar los días
%m	Para mostrar el mes
%y	Para mostrar el año (2 dígitos)
%Y	Para mostrar el año (4 dígitos)
%H	Para mostrar las horas
%M	Para mostrar los minutos
%S	Para mostrar los segundos

Existen otras muchas opciones que nos permiten construir formatos personalizados de fechas, horas, fecha y hora.

De todas formas, con estos valores tendremos cubiertas la mayoría de los casos que se suelen necesitar.

Programa: tiempos2.py

Vamos a hacer un programa que mostrará la fecha y hora en varios formatos habituales.

1º Realizaremos la importación de las funciones strftime, gmtime, localtime del módulo time

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Construiremos los diferentes formatos

4º Mostraremos los formatos construidos

```
1     from time import gmtime,localtime,strftime
2     print('Fecha y hora en formatos habituales')
3     f_corto=strftime('%d/%m/%y',localtime())
4     print('Fecha en formato corto:',f_corto)
5     fh_corto=strftime('%d/%m/%y %H:%M:%S',localtime())
6     print('Fecha y Hora:',fh_corto)
7     f_sql=strftime('%Y-%m-%d',localtime())
8     print('Fecha SQL:',f_sql)
9     fh_sql=strftime('%Y-%m-%d %H:%M:%S',localtime())
10    print('Fecha y Hora SQL:',fh_sql)
```

```
>>> %Run tiempos2.py

   Fecha y hora en formatos habituales
   Fecha en formato corto: 25/01/18
   Fecha y Hora: 25/01/18 17:32:01
   Fecha SQL: 2018-01-25
   Fecha y Hora SQL: 2018-01-25 17:32:01

>>>
```

A continuación vamos a desarrollar un programa que nos pida introducir una palabra y nos indique el tiempo que hemos tardado en introducir la palabra.

Programa: tiempos3.py

1º Realizaremos la importación de la función time del módulo time

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Tomaremos la marca de tiempo antes de solicitar la entrada de texto

4º Solicitaremos la entrada de texto

5º Tomaremos luna nueva marca de tiempo después de la entrada de texto

4º Mostraremos el resultado

```
1     from time import time
2     print('Calcularé el tiempo que tardas en escribir una palabra')
3     t1=time()
4     palabra=input('Escribe una palabra y pulsa intro: ')
5     t2=time()
6     tiempo=int(t2-t1)
7     print('En escribir la palabra',palabra,'has tardado',tiempo,'segundos')
```

El resultado de este programa sería:

```
>>> %Run tiempos3.py

   Calcularé el tiempo que tardas en escribir una palabra
   Escribe una palabra y pulsa intro: Informática
   En escribir la palabra Informática has tardado 4 segundos

>>>
```

En algunas ocasiones queremos nuestro programa espere un determinado tiempo, por ejemplo cuando mostramos alguna infromación, es posible que sea necesario dar tiempo al usuario para leerla.

Dentro del módulo time, disponemos de una función muy útil para este tipo de tareas, se trata de la función **sleep()**. A esta función hay que indicarle mediante un parámetro, el número de segundos que deseamos que espere:

sleep(2), esperará 2 segundos para continuar con la ejecución del programa.

Vamos a hacer un programa que genere 10 números aleatorios entre 1 y 100. Esta tarea es inmediata, pero nuestro programa lo hará a lo largo de 10 segundos, generando un número cada segundo.

Programa: tiempos4.py

1º Realizaremos la importación de las funciones sleep y randint de los módulos time y random

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Crearemos un bucle for para las 10 repeticiones.

4º Generaremos 1 número y mostramos la información.

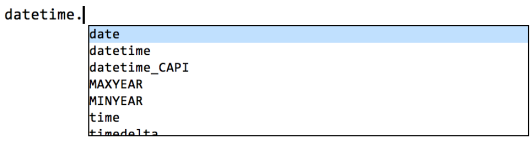
5º Esperamos 1 segundo antes de continuar con el bucle.

```
1    from random import randint
2    from time import sleep
3
4    print('Programa que genera 10 nùmeros aleatorios')
5    for n in range(10):
6        numero=randint(1,100)
7        print('Generación:',n+1,'Número generado:',numero)
8        sleep(1)
```

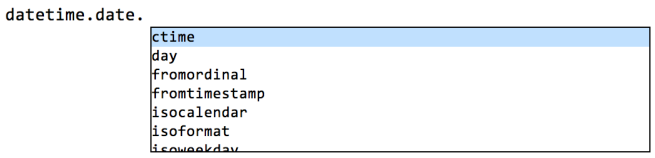
Este programa generará una salida similar a la siguiente imagen:

```
>>> %Run tiempos4.py

   Programa que genera 10 nùmeros aleatorios
   Generación: 1 Número generado: 79
   Generación: 2 Número generado: 66
   Generación: 3 Número generado: 55
   Generación: 4 Número generado: 44
   Generación: 5 Número generado: 1
   Generación: 6 Número generado: 87
   Generación: 7 Número generado: 38
   Generación: 8 Número generado: 54
   Generación: 9 Número generado: 33
   Generación: 10 Número generado: 38

>>>
```

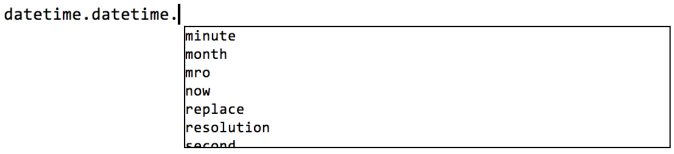
El módulo datetime



Dentro de las librerías estándar de Python también tenemos otro módulo que nos permite gestionar fechas de una forma un poco más fácil, este es el módulo datetime.



Este módulo pone a disposición varios objetos (clases) que disponen de muchas características, se trata de **datetime.date, datetime.time** y **datetime.datetime**:



Tanto el objeto date, como el datetime disponen del método today().

datetime.date.today() nos suministra la fecha actual.

datetime.datetime.today() nos suministra la fecha y la hora actual.

Veamos un ejemplo con estos objetos.

Programa: fechas1.py

1º Realizaremos la importación del módulo datetime.

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Mostraremos la fecha actual y algunos datos más.

4º Mostraremos la fecha y hora actual y algunos datos más.

```
1     import datetime
2     print(40*'-')
3     print('Información con datetime.date')
4     hoy=datetime.date.today()
5     print(hoy)
6     print(hoy.weekday())
7     print(hoy.day)
8     print(40*'-')
9     print('Información con datetime.datetime')
10    ahora=datetime.datetime.today()
11    print(ahora)
12    print(ahora.hour)
13    print(ahora.minute)
```

El programa genera una salida similar a la imagen:

```
>>> %Run fechas1.py

  ----------------------------------------
  Información con datetime.date
  2018-01-28
  6
  28
  ----------------------------------------
  Información con datetime.datetime
  2018-01-28 11:07:30.880630
  11
  7

>>>
```

La función weekday() devuelve un valor numérico que indica el día de la semana teniendo en cuenta que 0=lunes, 1=martes, 2=miércoles, 3=jueves, 4=viernes, 5=sábado y 6=domingo.

Ejercicios:

Nota: Recuerda que para realizar los siguientes ejercicios debes importar los módulos que correspondan en cada caso.

Ejercicio 1:

Realizar un programa que obtenga la raíz cuadrada de todos los números impares entre 1 y 100.

Ejercicio 2:

Realizar un programa que obtenga la raíz cuadrada de todos los números pares entre 1 y 100.

Ejercicio 3:

Realizar un programa que obtenga el logaritmo en base 10 de todos los números entre 1 y 100. **Nota:** log10(x) sirve para calcular el logaritmo en base 10.

Ejercicio 4:

Realiza un programa que solicite una palabra y a continuación muestre un carácter aleatorio de los que contenga la palabra introducida.

Ejercicio 5:

Realiza un programa que tenga 10 nombres escritos en una lista, y que elija aleatoriamente uno de los nombres de la lista. **Nota:** usa el módulo random y la función **choice** de este módulo.

Ejercicio 6:

Realiza un programa que tenga 10 nombres escritos en una lista, y que elija aleatoriamente tres de los nombres de la lista. **Nota:** usa el módulo random y la función **sample** de este módulo.

Ejercicio 7:

Realiza un programa que tenga 10 nombres escritos en una lista, y que muestre los nombres desordenados, es decir, desordenarlos aleatoriamente. **Nota:** usa el módulo random y la función **suffle** de este módulo.

Ejercicio 8:

Realiza un programa que genere 6 números aleatorios entre 1 y 49. Como el juego de la primitiva.

Ejercicio 9:

Realiza un programa que genere 15 resultados 1, x o 2. Como una apuesta de la quiniela.

Ejercicio 10:

Realiza un programa que genere 2 número aleatorios entre 1 y 6 simulando el lanzamiento de 2 dados. Debe mostrar los 2 números obtenidos y la suma de dichos números.

Capítulo 6

Definición de Funciones

```
def strPos(s,n):
    b=False
    if s[n]=='1': b=True
    return b


def pantalla(k):
    c=0
    for n in range(100):
        if strPos(strBin(n),-k):
            c+=1
            print(str(n).rjust(3),end=' ')
        if c==10:
            c=0
            print()
```

En el capítulo anterior hemos visto algunos módulos de Python que resultan muy útiles para llevar a cabo determinadas tareas.

Python es de los lenguajes de programación que dispone de un mayor número de librerías (módulos) que amplían con nuevas funcionalidades las características generales del lenguaje, pero a pesar de disponer de muchísimas funciones en los módulos de podamos encontrar, es muy probable que tengamos que crear nuestras propias funciones que realicen tareas específicas y personalizadas para el programa que tengamos que realizar.

¿Cuando es necesario crear una función?

Las funciones nos ayudan a organizar el código de nuestra aplicación. Si nuestro programa comienza a tener una extensión considerable, será recomendable separar nuestro código en funciones que cada una de ellas realice una tarea determinada del programa.

En otras ocasiones, hay una cierta tarea que hay que repetir en múltiples ocasiones dentro del programa, para estos casos, en lugar de repetir el código, se debe crear una función que haga dicha tarea y llamar a la función cada vez que necesitemos esa funcionalidad.

¿Cómo definir nuestras funciones?

Las funciones **debemos definirlas al principio de nuestro archivo**, de esta forma estarán disponibles en el resto del código para poder utilizarlas.

Las funciones se definen como un bloque de código que comienza con la palabra reservada **def**, y tiene la siguiente apariencia:

```
def nombrefuncion(parametro1, parametro2,...):
      instruccion1
      instruccion2
      …
      return valor      # Aunque es opcional, es lo más habitual
```

Cuando definimos una función, es necesario darle un nombre a la función, y generalmente tendremos que pasarle unos valores para que la función realice los cálculos o las tareas necesarias con esos valores. A los valores que se le pasan a la función se le llaman parámetros o argumentos, y una función puede recibir uno, varios o ninguno.

También es habitual que las funciones devuelvan un valor, aunque no es obligatorio. Generalmente el valor devuelto es el resultado de realizar las operaciones que lleva a cabo la función con los valores que le hemos pasado como argumentos.

Veamos un primer ejemplo de un programa en el que definiremos una función que calcule la diagonal de un cuadrado.

La función la llamaremos diagonal y recibirá un parámetro que llamaremos lado.

Dentro de nuestra función tendremos una variable que llamaremos resultado que será el valor que finalmente devolverá nuestra función con las instrucción return.

En el programa calcularemos la diagonal de varios cuadrados, de manera que llamaremos en varias ocasiones a la función diagonal.

Vamos a recordar que la diagonal del cuadrado podemos calcularla aplicando el Teorema de Pitágoras, $\mathbf{d^2=l^2+l^2}$ (en este caso los 2 catetos coinciden con el lado), por lo tanto, la diagonal es la raíz cuadrada de $\mathbf{2*l^2}$.

Programa: funciones1.py

1º Realizaremos la importación del módulo math.

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Definiremos la función diagonal.

4º Mostraremos la diagonal de los cuadrados de lado 1, 5, 10 y 12.

```
1     from math import sqrt
2     def diagonal(lado):
3         resultado=sqrt(lado**2+lado**2)
4         return resultado
5
6     print('Programa que calcula la diagonal de varios cuadrados')
7     print('El cuadrado de lado',1,'tiene de diagonal',diagonal(1))
8     print('El cuadrado de lado',5,'tiene de diagonal',diagonal(5))
9     print('El cuadrado de lado',10,'tiene de diagonal',diagonal(10))
10    print('El cuadrado de lado',12,'tiene de diagonal',diagonal(12))
```

Obtendremos como ejecución del programa:

```
>>> %Run funciones1.py

  Programa que calcula la diagonal de varios cuadrados
  El cuadrado de lado 1 tiene de diagonal 1.4142135623730951
  El cuadrado de lado 5 tiene de diagonal 7.0710678118654755
  El cuadrado de lado 10 tiene de diagonal 14.142135623730951
  El cuadrado de lado 12 tiene de diagonal 16.97056274847714

>>>
```

Ejercicios:

Nota: Recuerda que para realizar los siguientes ejercicios debes importar los módulos que correspondan en cada caso.

Ejercicio 1:

Realizar un programa que obtenga la raíz cuadrada de todos los números impares entre 1 y 100.

Ejercicio 2:

Realizar un programa que obtenga la raíz cuadrada de todos los números pares entre 1 y 100.

Ejercicio 3:

Realizar un programa que obtenga el logaritmo en base 10 de todos los números entre 1 y 100. **Nota:** log10(x) sirve para calcular el logaritmo en base 10.

Ejercicio 4:

Realiza un programa que solicite una palabra y a continuación muestre un carácter aleatorio de los que contenga la palabra introducida.

Ejercicio 5:

Realiza un programa que tenga 10 nombres escritos en una lista, y que elija aleatoriamente uno de los nombres de la lista. **Nota:** usa el módulo random y la función **choice** de este módulo.

Ejercicio 6:

Realiza un programa que tenga 10 nombres escritos en una lista, y que elija aleatoriamente tres de los nombres de la lista. **Nota:** usa el módulo random y la función **sample** de este módulo.

Ejercicio 7:

Realiza un programa que tenga 10 nombres escritos en una lista, y que muestre los nombres desordenados, es decir, desordenarlos aleatoriamente. **Nota:** usa el módulo random y la función **shuffle** de este módulo.

Ejercicio 8:

Realiza un programa que genere 6 números aleatorios entre 1 y 49. Como el juego de la primitiva.

Ejercicio 9:

Realiza un programa que genere 15 resultados 1, x o 2. Como una apuesta de la quiniela.

Ejercicio 10:

Realiza un programa que genere 2 número aleatorios entre 1 y 6 simulando el lanzamiento de 2 dados. Debe mostrar los 2 números obtenidos y la suma de dichos números.

Ejercicio 11:

Realiza un programa que genere 10 fechas aleatorias a partir de la fecha de hoy. Nota: puedes usar la función **timedelta()** que se encuentra en el módulo **timedate**.