TEMA 2

Input - Output y Variables

Curso de Python

Alfredo Sánchez Sánchez
@alfsan04
www.educoteca.com

ÍNDICE

Nuestro primer programa

Vamos a realizar nuestros primeros programas en Python. Como ya hemos visto en el tema 1, podemos trabajar en Python desde un archivo con extensión .py o directamente desde la IDLE o, en caso de Linux o Mac, desde Terminal una vez instalada la versión de Python que queremos usar.

Vamos a usar Python 3 y vamos a comenzar con la terminal o la IDLE. Nuestro primer paso va a ser **mostrar un texto** como resultado de un proceso que realice Python. En los vídeos se explica un poco más este proceso, pero casi siempre todo primer programa en un lenguaje de programación empieza con un "**Hola mundo"** (si tienes curiosidad puedes buscar por qué se usa la expresión Hola mundo en programación).

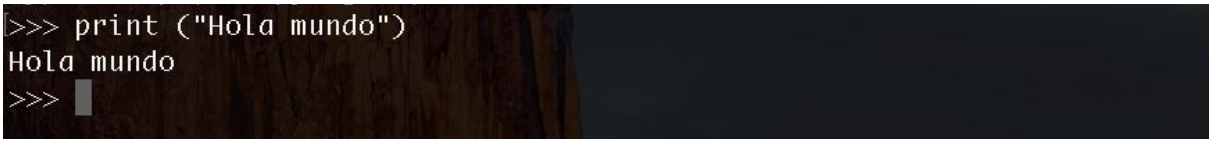
En nuestro caso, vamos a hacer que Python nos muestre el texto "Hola mundo" desde Terminal. Para ello vamos a cargar Python 3 y vamos a pedirle que muestre (o imprima) el texto "Hola mundo". Usaremos el comando **print**, que indica al programa que debe mostrar algo al usuario. Nosotros como usuarios debemos indicarle a Python qué es lo que debe mostrar (un texto, un número, el resultado de una operación…).

Para conseguir que nos muestre un texto, debemos utilizar comillas para delimitar el texto, es la forma en la que Python sabe que debe ser literal y mostrarnos lo que pone entre las mismas. Más adelante verás con más claridad por qué usar comillas para delimitar el texto (a partir de ahora al texto lo llamaremos **string**).

Así mismo, debemos usar paréntesis para englobar lo que queremos que Python imprima, por lo que escribiremos en nuestra terminal lo siguiente:

```
print ("Hola mundo")
```

Ahora prueba tú, abre la IDLE de Python o Python 3 en Terminal y escribe el comando anterior. Prueba a cambiar el texto entre las comillas y observa lo que ocurre al presionar intro.



A mucha gente en este punto se le ocurre cuestionar la utilidad de imprimir algo de esta forma, especialmente porque el propio texto que imprimimos está escrito una línea más arriba. Hay que entender que este proceso se aplica generalmente en medio de un programa

de Python (a partir de ahora **script**) en el cual el usuario no ve el código, sólo lo que el programa le enseña.

Vamos a practicar para comprenderlo mejor. Abre un documento (como hemos visto puedes hacerlo desde un bloc de notas o, más recomendable, usando un editor de código como **Atom**). Guárdalo en el escritorio con el nombre de *holamundo.py*. A continuación escribe en dicho archivo la instrucción *print ("Hola mundo")* y vuelve a guardarlo. Por último, ejecuta el archivo desde Terminal o desde la IDLE y mira lo que te muestra antes de continuar.

¿Ya lo has hecho? Como podrás observar, al cargar dicho archivo Python no nos muestra el código y la instrucción *print*, por lo que la salida que el usuario (en este caso nosotros) obtiene es el texto que había que imprimir. A este proceso se le denomina **output** o salida de datos desde el programa hacia el usuario.

Interaccionando con el programa

Ya sabemos cómo imprimir un **string** usando Python, pero el usuario en un **script** como el anterior no puede más que ejecutarlo y recibir lo que el programa le muestra, no hay ningún tipo de interacción usuario - programa.

Vamos a intentar construir nuestro primer programa en el cual se produzca una entrada de datos del usuario hacia el programa y el programa con esos datos dé una respuesta. A este flujo se le denomina **input - output.**

En nuestro ejemplo anterior, Python nos ha mostrado el texto *Hola mundo*, pero quizá sería mejor que el programa se aprendiese nuestro nombre y nos saludase (en lugar de saludar a todo el mundo).

Si pensamos en cómo debería ser un programa que hiciese lo anteriormente descrito, podríamos esquematizarlo de la siguiente forma:

El programa nos pregunta nuestro nombre → contestamos un nombre → el programa lo almacena → el programa imprime un saludo y nuestro nombre.

En el proceso anterior hay una palabra clave: **almacena.** Python no puede utilizar una entrada de datos desde el usuario directamente (o, mejor dicho, no es una buena práctica). Necesita guardarla en algún sitio y posteriormente utilizarla para el fin que hayamos programado.

Realmente nuestro cerebro funciona de la misma forma, si una persona se presenta y nos dice su nombre, nuestra memoria de trabajo almacena ese nombre y nosotros le saludaremos utilizando su nombre. El asunto es que nuestro cerebro no necesita que le indiquemos dónde tiene que almacenar ese nombre, es algo que ocurre y nosotros sabemos utilizar: tenemos memoria (unos más que otros).

En cambio Python si necesita que le digamos cómo tiene que almacenar ese nombre. Como a la hora de realizar el programa no sabemos qué va a contestar el usuario no podemos sino decirle algo parecido a: “guarda la respuesta que te de con el nombre *respuesta*”. De esta forma si quiero usar ese nombre que el usuario ha dado sólo tendré que decirle a Python que utilice la *respuesta*.

Para almacenar algo que el usuario tenga que escribir usamos la función **input**, de forma que yo puedo imprimir directamente una respuesta que dé el usuario utilizando un comando como el siguiente:

```
print(input("¿Cuántos años tienes?: "))
```

Fíjate que hay un espacio tras los dos puntos de la pregunta. Los espacios también son caracteres, en la siguiente captura de pantalla entenderás el por qué de ese espacio:

```
>>> print (input ("¿Cuántos años tienes?: "))
¿Cuántos años tienes?: 34
34
>>>
```

Si nos fijamos, el espacio tras los puntos permite que, al hacer la pregunta el programa y el usuario dar una respuesta aparezca con cierta lógica y no todo pegado y sin espacios. Si aún así no te ha quedado claro lo mejor es que pruebes ambas opciones y veas la diferencia.

No es una buena práctica usar un input dentro de un print, lo mejor sería guardar primero la respuesta y posteriormente utilizarla. Para ello necesitamos conocer el concepto de **variable.**

Una variable, término del que hablaremos en profundidad en el siguiente punto de este capítulo, no es más que un espacio en la memoria del dispositivo (ordenador, tablet, móvil…) reservado para guardar un dato.

Para entender el concepto, si creo una variable es como si pusiese una caja en una estantería donde puedo guardar cualquier cosa, ese espacio está en la estantería reservado para guardar cosas y puede contener datos hasta que está llena.

En Python, las variables se crean poniéndoles un nombre, de forma que si yo quiero crear una variable para almacenar un valor, tendré que nombrar el espacio donde va a estar guardado el dato y posteriormente asignarle a ese espacio un valor o dato, por ejemplo:

```
name = "Alfredo"
```

En el ejemplo anterior Python reserva en su **memoria de trabajo** un hueco al que llama *name* y en ese hueco almacena el string “Alfredo”, de forma que yo puedo utilizar ese nombre en cualquier momento en mi programa simplemente mencionando el hueco que lo contiene.

Para imprimir el nombre símplemente le tendré que decir a Python que imprima el hueco *name.* Fijate en el siguiente código:

```
name = "Alfredo"
print(name)
```

Pruébalo en tu terminal o IDLE.

Como puedes ver, el programa imprime el contenido del hueco *name*, que para mi ejemplo es el string *Alfredo*.

La instrucción *print* tiene entre paréntesis un nombre sin comillas, es el momento de entender por qué. Si pongo comillas imprimirá un string (un texto) y si no pongo comillas pasará a imprimir el contenido de una variable (es decir, el contenido de un hueco en la memoria).

Ahora vamos a combinar un texto fijo con el contenido de una variable. Para encadenar textos sólo tengo que sumarlos con un signo +, como muestra el ejemplo:

```
name = input ("¿Cómo te llamas?: ")
print ("Hola, " + name)
```

Fíjate que he vuelto a dejar un espacio tras el *Hola* y la coma, para que el nombre almacenado en la variable *name* aparezca con un espacio. Python va a imprimir un string y una variable, si la variable *name* contiene algo lógico imprimirá una frase lógica (por ejemplo *Hola, Alfredo*). En cambio Python no es capaz de discernir si el contenido de *name* es lógico o no. Por ejemplo podemos almacenar como input un espacio, un punto, números…:

El nombre 67 no tiene mucho sentido a nuestros ojos, pero Python no sabe de ojos (o cerebros) humanos. Empezamos a entender aquí que la programación hará fielmente lo que el programador indique (siempre y cuando sea algo que el programa pueda entender) pero nunca supondrá que algo no tiene sentido o que debería advertir al usuario de un posible error. Entiende por error algo que un humano pudiese razonar, no una instrucción errónea.

Variables

Ya hemos visto brevemente lo que es una variable. Por ejemplo, si queremos almacenar un número y usarlo en cualquier momento del programa declararemos la variable *number* con el valor que queramos:

```
number = 6
```

Ambas veces he usado como nombres para las variables textos vinculados al contenido, pero no tiene porque ser así, podrías llamar *elefante* a una variable que contenga un número. A la hora de declarar una variable debemos tener en cuenta que Python reconoce la diferencia entre mayúsculas y minúsculas. Observa el siguiente ejemplo y lo entenderás:

Por ello, los programadores suelen evitar usar mayúsculas iniciales para las variables, así evitan errores.

Una variable reserva un espacio en memoria para almacenar un dato y la memoria de un ordenador no es infinita (si bien hoy en día y al nivel que vamos a programar en Python es difícil que esto suponga un problema). Por ello existen diferentes tipos de variables que llevan asociado un tamaño en memoria. Siguiendo con el símil de la caja en una estantería podríamos decir que no necesitamos el mismo tamaño de caja para guardar naranjas que para guardar sandías y, por extensión, cada caja ocupa un tamaño de la estantería y no conviene almacenar una naranja en una caja para sandías.

De la misma forma, no necesitamos el mismo tamaño para guardar un número entero que una frase. Eso sí, a diferencia de otros lenguajes de programación, en Python no hay que indicar qué **tipo de variable** necesitas (es decir, **cuánto espacio va a ocupar**), basta con dar un nombre y un contenido y Python automáticamente, por el tipo de contenido, sabrá el tipo de variable que es necesaria.

Inicialmente vamos a ver tres tipos de variables:

- Variables que contienen números enteros (en Python ***int*** de integer).
- Variables que contienen números decimales (en Python ***float***, de coma flotante).
- Variables que contienen texto (en Python ***str*** de string).

A la hora de iniciar un programa o en diferentes partes del mismo vamos a necesitar habitualmente crear variables. Podemos definirlas en líneas sucesivas...

```
number = 6
word = "hola"
decimal = 3.14
```

... pero también podemos crear varias variables de manera simultánea separando sus nombres por comas y los contenidos de cada una de ellas igualmente por comas:

```
number, decimal, word = 6, 3.14, "hola"
```

Saber de qué tipo es una variable a veces es bastante importante, pues hay determinadas operaciones entre variables que sólo se pueden realizar si son del mismo tipo. Para saber el tipo de una variable (*str, float* o *int*) sólo tenemos que escribir el comando ***type*** y el nombre de la variable entre paréntesis:

```
type (name)
```

Veamo el proceso ejecutado:

Prueba a declarar varios tipos de variable con diferentes nombres (una que contenga un nombre, una que contenga un número entero y otra que contenga un decimal) y pide a Python que te muestre el tipo de variable de cada una.

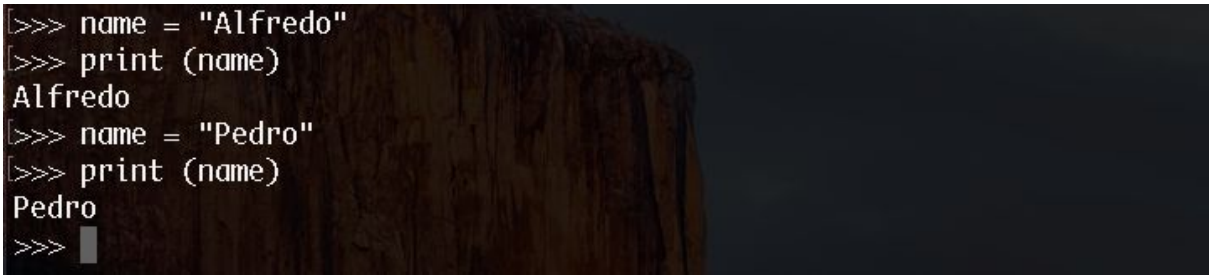
Modificando variables

En Python podemos, sin ningún problema, cambiar el contenido de una variable, simplemente volvemos a declarar con el mismo nombre otro contenido:

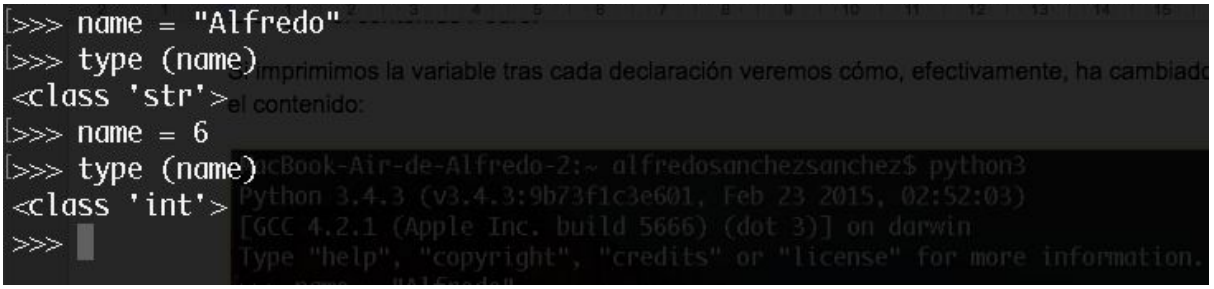
```
name = "Alfredo"
name = "Pedro"
```

En el caso anterior, la variable *name* contiene el string *Alfredo* y posteriormente el programa le asigna el contenido *Pedro*.

Si imprimimos la variable tras cada declaración veremos cómo, efectivamente, ha cambiado el contenido:



En realidad Python no cambia el contenido sino que elimina la variable anterior y crea otra con el mismo nombre y el contenido nuevo. De esta forma, podemos reasignar el contenido a una variable aunque cambie el tipo:



Operando variables

Las variables nos permiten operar con ellas siempre y cuando la operación tenga sentido para el contenido. De esta forma se puede operar con variables con contenidos numéricos como si operásemos con el contenido directamente:

```
number1, number2 = 6, 14
print (number1 + number2)
```

Prueba el código anterior en Python.

También podemos almacenar el valor de la suma en una nueva variable:

```
number1, number2 = 6, 14
suma = number1 + number2
print (suma)
```

El resultado es el mismo, pero la suma queda almacenada en una nueva variable por si queremos usarla más adelante.

De la misma forma puedo restar, multiplicar o dividir variables.

Pero, ¿qué ocurre si opero strings?. Dos variables que contienen texto se pueden sumar, prueba tu mismo el siguiente código:

```
word1, word2 = "Hello", "man"
print (word1 + word2)
```

Como verás, se suman las variables, aunque falta un espacio entre medias o algo que delimite las diferentes variables, podemos solucionarlo con el siguiente código...

```
word1, word2 = "Hello", "man"
print (word1 + " " + word2)
```

… en el cual hemos sumado primero un espacio al final de la primera palabra y luego la segunda palabra a la primera suma. Pruébalo también antes de seguir.

Vamos a ver ahora otro ejemplo de suma. Queremos realizar una suma de dos números que elija el usuario. Para ello vamos a guardar dos *input* y vamos a imprimir su suma:

```
number1 = input("Primer número a sumar: ")
number2 = input("Segundo número a sumar: ")
print (number1 + number2)
```

Prueba el código anterior y mira a ver qué ocurre.

Como habrás podido comprobar no está sumando los números sino que está uniéndolos y formando un único número con ellos, es decir, está sumando los números como si fuesen *strings*… ¿Pudiera ser que fuesen strings? Prueba el siguiente código contestando con un número al input:

```
number1 = input("Primer número a sumar: ")
type (number1)
```

Como puedes ver, el número almacenado es un string, es decir, texto. Python permite realizar ciertos cambios en el tipo de variables para usarlas con otro fin:

- Puedo cambiar un *int* o *float* a *string* en cualquier momento y el número pasará a ser texto.
- Puedo convertir un *str* que sólo contenga números enteros en un *int*.
- Puedo convertir un *str* que contenga un número decimal en un *float*.

De momento lo dejamos ahí, aunque hay más cambios aceptados.

¿Cómo se convierte una variable a otro tipo? Bien, podemos mantener el nombre o podemos guardarla con el nuevo tipo bajo otro nombre:

```
#manteniendo el nombre
number = int(number)
#cambiando el nombre
new_number = int(number)
```

En el primer caso Python está reasignando a *number* un valor, por lo que el valor anterior pasa a dejar de existir. En el segundo está convirtiendo a entero el contenido de *number* y lo almacena con otro nombre.

Para conseguir ahora hacer la suma de dos números que el usuario elija deberíamos realizar el siguiente código:

```
number1 = input("Primer número a sumar: ")
```

```
number2 = input("Segundo número a sumar: ")
number1 = int(number1)
number2 = int(number2)
print(number1+number2)
```

Prueba el código anterior guardándolo en un archivo en vez de por terminal y cargándolo desde terminal o la IDLE.

Como habrás podido ver funciona. Otra opción es directamente convertir la variable a entero en la misma sentencia que el input:

```
number1 = int(input("Primer número a sumar: "))
number2 = int(input("Segundo número a sumar: "))
print(number1+number2)
```

El efecto es el mismo, así que si sabemos que vamos a convertir la respuesta siempre a entero podemos hacer este paso directamente.

Profundiza

Vamos a indicar las operaciones matemáticas que se pueden realizar de forma directa en Python o bien que se pueden realizar entre variables que contengan números.

Multiplicación:

```
2*2
4
```

División:

```
2/2
1.0
```

Suma:

```
2+3
5
```

Resta:

```
5-2
3
```

Potencia:

```
5**2
25
```

Las raíces se expresan como potencias decimales (en el ejemplo raíz cuadrada):

```
25**0.5
5
```

Operación para obtener la parte entera de una división:

```
5//2
2
```

Operación para obtener el resto de una división:

```
5%2
1
```

Python opera siguiendo las prioridades matemáticas. En el ejemplo, primero se realizará la multiplicación y después la suma:

```
5+2*3
11
```

Otra forma de aplicar una potencia es usar la función *pow* que nos obliga a indicar la base y el exponente para realizar la operación:

```
pow (2,3)
8
```

Para redondear un número podemos usar la función *round*:

```
round (4.35)
4
```

Si queremos redondear a un número concreto de decimales podemos hacerlo con la misma función, indicando primero el número y después la cantidad de decimales que queremos:

```
round (4.3578, 2)
4.36
```

Si queremos truncar un número, es decir, redondearlo hacia abajo siempre, debemos importar una parte de una librería (más adelante hablaremos de ellas con calma), en concreto, tenemos que importar la función *floor* de la librería *math:*

```
from math import floor
floor (4.35)
```

```
4
```

Si en cambio queremos siempre redondear al número superior debemos importar la función *ceil* de la librería *math:*

```
from math import ceil
ceil (4.35)
5
```

Hasta aquí llegamos en este tema con las variables. Cuando hayas probado las opciones matemáticas en la terminal o en la IDLE puedes visionar los videos de la unidad para ver todo esto en práctica y posteriormente pasar al ejercicio propuesto para afianzar esta sesión.

Instrucciones de código usadas

```
# Imprimir texto
   print ("texto")
# Imprimir texto introducido por el usuario
   print (input ("texto"))
# Declarar variable
   name = "Nombre"
   number = 6
# Ver el tipo de una variable
   type (nombre_variable)
# Guardar respuesta de usuario en variable
   respuesta = input ("Pregunta")
# Sumar dos variables y guardar resultado en nueva variable
   suma = variable1 + variable2
# Declarar varias variables simultáneamente
   variable1, variable2 = "texto", numero
# Operaciones matemáticas
   # Multiplicación
      numero * numero
   # División
      numero / numero
   # Suma
      numero + numero
   # Resta
      numero - numero
   # Potencia
      numero ** numero
   # Parte entera de una división
      numero // numero
   # Resto de una división
      numero % numero
   # Redondear un número
      round (numero)
   # Redondear a una serie de decimales
      round (numero, decimales)
   # Truncar
      from math import floor
      floor (numero)
   # Redondear al número superior
      from math import ceil
      ceil (numero)
```

Reto

Debes hacer un script que pregunte al usuario por dos números y los almacene (en diferentes variables). A continuación el programa deberá mostrarnos varias operaciones comentando primero qué operaciones ha realizado. Al menos debe realizar cinco operaciones con los números introducidos.

Los números introducidos pueden condicionar las operaciones por lo que se debe avisar de qué números no son válidos (por ejemplo, no podemos realizar la raiz cuadrada de un número negativo o no se puede dividir entre cero).