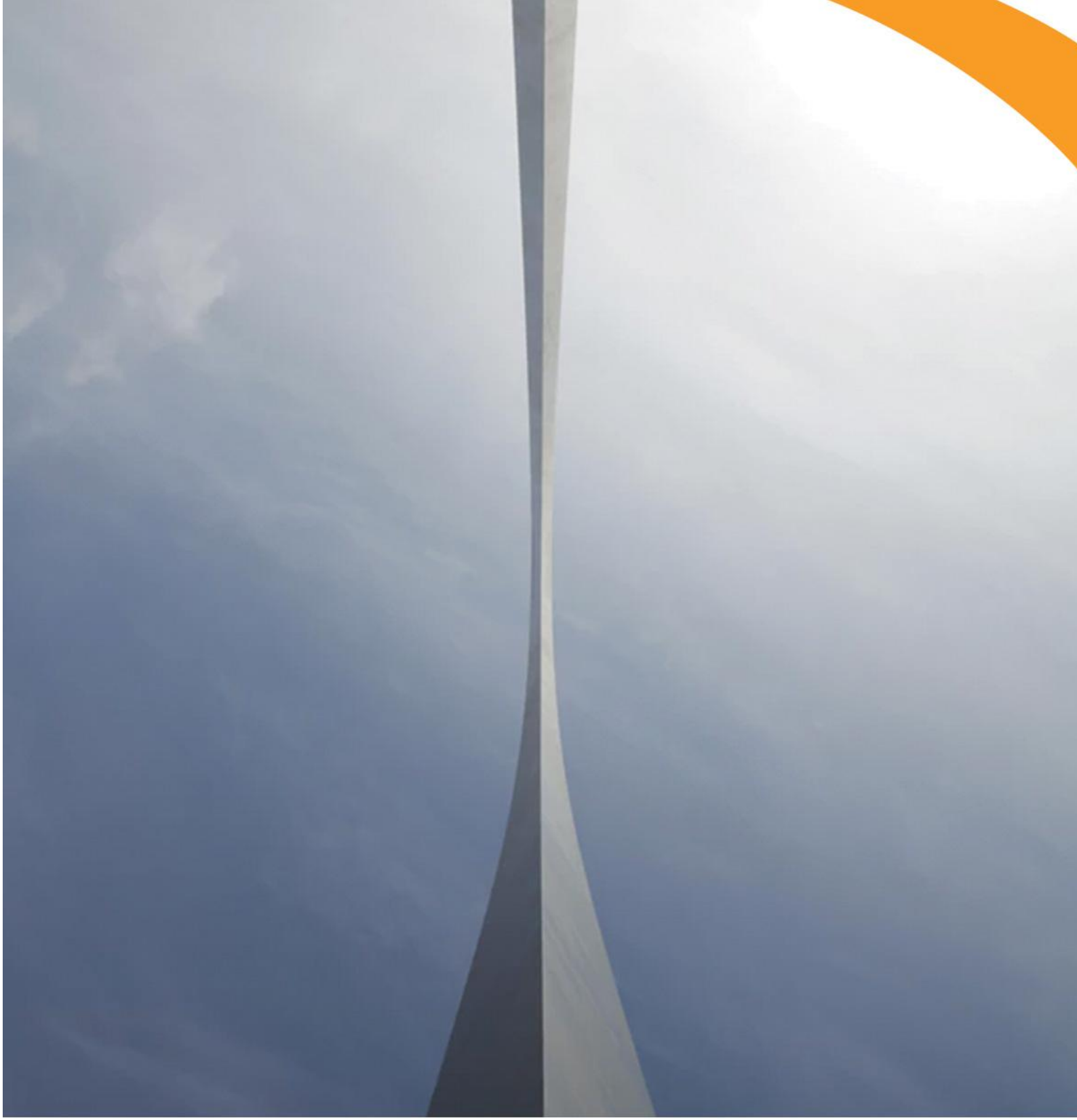


PROGRAMACIÓN CON CÓDIGO: PYTHON

INPUT – OUTPUT Y VARIABLES

Structuralia
Formación para la ingeniería

Este documento es de uso único e intransferible para el alumno matriculado en el curso. Cualquier reproducción física o digital del documento sin permiso de los autores vulnera los derechos de propiedad intelectual de los mismos.

INDICE

INDICE ***3***

1. NUESTRO PRIMER PROGRAMA ***4***

2. INTERACCIONANDO CON EL PROGRAMA ***7***

2.1 Almacenando la entrada de datos *11*

3. VARIABLES ***15***

4. MODIFICANDO VARIABLES ***20***

5. OPERANDO VARIABLES ***22***

6. PROFUNDIZA ***28***

7. DOCUMENTACIÓN OFICIAL DE PYTHON ***32***

8. INSTRUCCIONES DE CÓDIGO USADAS ***33***

9. RETO ***35***

1. NUESTRO PRIMER PROGRAMA

Una vez has instalado Python y Thonny, vamos a empezar a programar.

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 Thonny o la IDLE o, en caso de Linux o Mac, desde Terminal una vez instalada la versión de Python que queremos usar.

Vamos a comenzar con Thonny. Nuestro primer paso va a ser **mostrar un texto** como resultado de un proceso que realizará 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"** (Hello World, en inglés). La costumbre de utilizar esta expresión como primera cosa a imprimir en programación viene de los años 70, cuando Brian Kernighan, autor de uno de los libros de programación más famosos de la historia: *C Progamming Language* (1978), introdujo esa expresión en un libro que escribió en 1973 llamado *A tutorial Introduction to the Programming Language B*. Desde entonces se ha venido usando la expresión de forma habitual.

En nuestro caso, vamos a hacer que Python nos muestre el texto "Hola mundo" a través de Thonny. 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 interpretar algo literal (texto) 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**, que quiere decir cadena, por aquello de ser una cadena de caracteres).

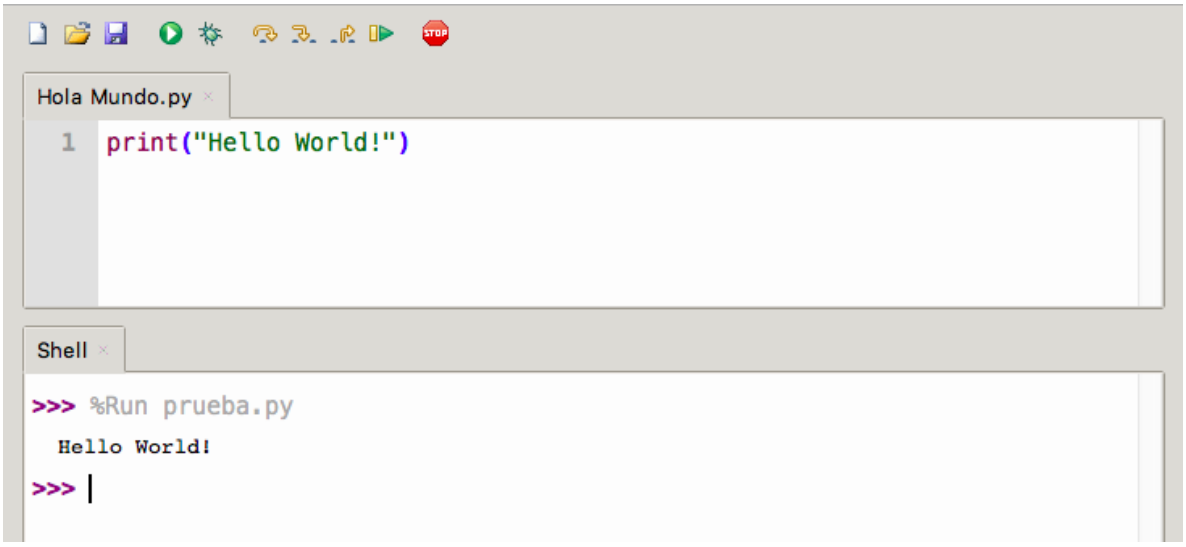
Así mismo, debemos usar paréntesis para englobar lo que queremos que Python imprima, por lo que escribiremos en nuestra terminal lo siguiente:

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

Ya iremos profundizando en ello pero, en principio, los espacios entre las instrucciones y los paréntesis no afectan en absoluto. Es lo mismo escribir *print ( )* que escribir *print( ).*

Ahora prueba tú, abre Thonny y escribe el comando anterior. Pulsa el botón verde similar a un *play* clásico. Al posicionar el cursor de ratón sobre el mismo aparece la instrucción *Run current script (F5).* Ya sabes, a partir de ahora para ejecutar un script en Thonny puedes pulsar F5 directamente.

Prueba a cambiar el texto entre las comillas y observa lo que ocurre al F5.

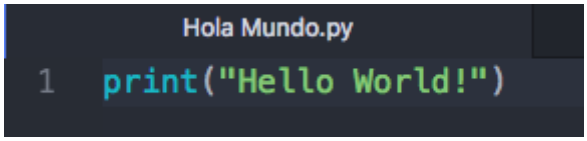


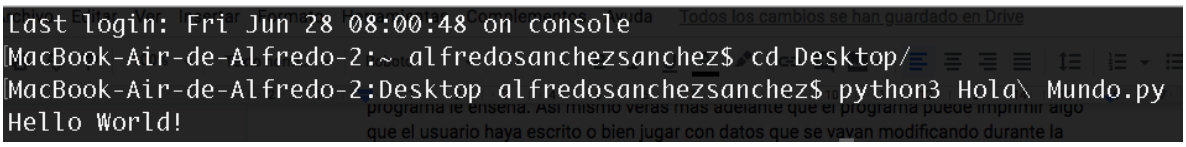
A mucha gente en este punto se le ocurre cuestionar la utilidad de imprimir algo de esta forma, especialmente porque el propio texto que imprimimos está escrito una línea más arriba. Hay que entender que este proceso se aplica generalmente en medio de un programa de Python (a partir de ahora llamaremos **script** a cualquier programa que construyamos) en el cual el usuario no ve el código, sólo lo que el programa le enseña. Así mismo verás más adelante que el programa puede imprimir algo que el usuario haya escrito o bien jugar con datos que se vayan modificando durante la ejecución y acaben dando lugar a un dato que el usuario no había introducido. Por poner un ejemplo, el programa puede imprimir el gasto total acumulado a lo largo de la ejecución de una obra o la cantidad de cubos de pintura asfáltica restante en el inventario.

Resumiendo, el código estará oculto y lo que el usuario verá será su ejecución en la pestaña *Shell* de Thonny.

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 en Mac o Linux o desde símbolo del sistema en Windows 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. En las imágenes posteriores puedes ver el programa escrito en Atom y ejecutado con la Terminal en mi MacBook Air.





2. 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 personalizada. A este flujo se le denomina **input - output** (entrada de datos, salida de datos)**.**

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 de la misma forma) o bien nos preguntase nuestra edad y fuese capaz de repetirla.

Si pensamos en cómo debería ser un programa que hiciese lo anteriormente descrito, podríamos esquematizarlo. Nota: esquematizar y utilizar un papel y un bolígrafo es una manera maravillosa de evitarse procesos largos y tediosos al programar, pues al realizar el esquema se pueden detectar y evitar problemas que luego darán muchos quebraderos de cabeza. Veamos el esquema:

El programa nos pregunta nuestra edad → contestamos una edad → el programa imprime la edad.

Para incorporar algo que el usuario tenga que escribir como respuesta usamos la función **input**, de forma que podemos 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 de la Terminal de mi Mac entenderás el porqué 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 aun así no te ha quedado claro lo mejor es que pruebes ambas opciones y veas la diferencia.

Es importante tener en cuenta que la respuesta no tiene por qué tener sentido o, incluso, si contestamos directamente a un *input* pulsando una vez la tecla intro el programa entenderá que tu entrada de datos es un dato vacío, sin contenido (a ese tipo de dato en programación se le llama *null*, de dato nulo, inexistente). Aquí tienes el ejemplo en Thonny:

```
Shell
Python 3.7.2 (bundled)
>>> print(input("¿Cuántos años tienes?: "))
  ¿Cuántos años tienes?:

>>>
```

Como ves, el programa ha impreso una línea vacía al introducir un dato vacío en el *input*. Este código se podría usar para parar la ejecución de un programa hasta que el usuario pulse intro. Observa el siguiente código y su ejecución:

```
print("Era una tarde lluviosa de abril")
print("Tras varias semanas lloviendo el camino era un lodazal")
input("... presiona intro para continuar... ")
print("Nuestro protagonista se aproximó a un hermoso pueblo")
```

```
prueba.py
1  print("Era una tarde lluviosa de abril")
2  print("Tras varias semanas lloviendo el camin
3  input("... presiona intro para continuar... "
4  print("Nuestro protagonista se aproximó a un

Shell
>>> %Run prueba.py
  Era una tarde lluviosa de abril
  Tras varias semanas lloviendo el camino era un lodazal
  ... presiona intro para continuar...
```

Como ves, el programa se queda parado en su ejecución esperando que se pulse intro para concluir el *input*. Una vez pulsado el intro se imprime la siguiente línea de texto:

```
prueba.py
1  print("Era una tarde lluviosa de abril")
2  print("Tras varias semanas lloviendo el camino
3  input("... presiona intro para continuar... ")
4  print("Nuestro protagonista se aproximó a un he

Shell
>>> %Run prueba.py
  Era una tarde lluviosa de abril
  Tras varias semanas lloviendo el camino era un lodazal
  ... presiona intro para continuar...
  Nuestro protagonista se aproximó a un hermoso pueblo
>>>
```

Puedes apreciar que el *input* no necesita un *print*, ya se ocupa de imprimir la pregunta. Si ponemos un *print* en un *input* le estamos diciendo al programa que imprima la respuesta, pues la pregunta se imprime automáticamente con el propio *input*.

Aquí tienes la diferencia entre poner *print* y no ponerlo. Empecemos con un programa que incluya un *print* en el *input*:

```
prueba.py
1  print(input("Dime algo: "))

Shell
>>> %Run prueba.py
  Dime algo: Hola
  Hola
>>> |
```

Como ves, el programa imprime mi respuesta, que es un *Hola*, justo después de presionar intro tras introducir un texto en respuesta a la petición *Dime algo*. Veamos ahora qué pasa si no pongo el *print*:

```
prueba.py
1  input("Dime algo: ")

Shell
>>> %Run prueba.py
  Dime algo: Hola
>>> |
```

En esta ocasión el programa no ha impreso el *Hola* que yo he contestado en el *input*.

¿Te ha quedado claro? es importante ir entendiendo estas pequeñas cosas. El input requiere una respuesta, que podemos imprimir o no.

2.1 Almacenando la entrada de datos

Pero, ¿qué pasa si queremos volver a usar la respuesta en el futuro? Imagina que la edad va a ser usada en múltiples ocasiones a lo largo del programa y cada vez que la necesitemos tengamos que volver a preguntarla… No parece una práctica muy inteligente.

¿Qué tal si modificamos el esquema anterior?:

El programa nos pregunta nuestra edad → contestamos una edad → el programa la almacena → el programa imprime una frase y le añade la edad.

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 (al menos esto funciona con la mayoría de los seres humanos, aunque hay excepciones, como la de mi cerebro, que no es excesivamente bueno almacenando nombres). 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 dé 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 datos necesitamos conocer un nuevo término: **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 de forma sencilla, 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 simplemente le tendré que decir a Python que imprima lo que contiene el hueco *name* de la memoria. Fíjate en el siguiente código:

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

Pruébalo en tu terminal, Thonny o IDLE.

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

Como puedes ver, el programa imprime el contenido del hueco *name*, que para el ejemplo anterior 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 entenderá que lo que hay entre paréntesis es el nombre de una variable (u otros elementos de Python que veremos más adelante) y pasará a buscar el contenido alojado en memoria con ese nombre para imprimirlo. Es similar al proceso que hace nuestro cerebro, si nos preguntan nuestro nombre buscamos en el cerebro la respuesta a la pregunta *nombre* y lo decimos.

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)
```

Prueba el ejemplo anterior en Thonny, Terminal o IDLE antes de continuar.

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…:

```
>>> name = input("¿Cómo te llamas?: ")
¿Cómo te llamas?: 67
>>> print ("Hola, " + name)
Hola, 67
>>>
```

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.

También podríamos almacenar la frase elaborada con la respuesta en una variable para usarla posteriormente más veces si fuese necesario. Por ejemplo, preguntar el nombre la primera vez que iniciemos el programa y luego ya saludar de la misma forma cada vez que lo iniciemos de nuevo.

Para guardar la frase podemos hacerlo tal como muestra la imagen del Shell de Thonny:

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

3. 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 por qué 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:

```
>>> number = 6
>>> Number = 12
>>> print (number)
6
>>> print (Number)
12
>>>
```

Por ello, los programadores suelen evitar usar mayúsculas iniciales para las variables, así evitan errores (además, esa forma de escribir el nombre se reserva para otra instrucción de Python que veremos más adelante).

Los ordenadores, dispositivos móviles y cualquier aparato electrónico con un sistema operativo tienen diferentes espacios donde almacenan datos, información, programas… Por un lado tienen una memoria permanente, donde se almacenan datos y quedan almacenados de forma permanente hasta que el usuario o el sistema por algún proceso concreto lo borre. Por otro lado tienen una memoria de trabajo, llamada memoria RAM (Random Access Memory o memoria de acceso aleatorio). Dicha memoria está funcionando y almacena datos mientras tiene corriente eléctrica, una vez se apaga el dispositivo que la incluye se borra el contenido.

Podemos encontrar un símil de este proceso en el cerebro humano, que tiene una memoria de trabajo y una memoria de largo plazo, al dormir se traspasan datos de la memoria de trabajo a la memoria de largo plazo (tampoco quiero entrar en una definición o explicación más exacta médicamente hablando del proceso anterior, es una imagen un poco vaga de lo que ocurre en realidad). Es muy interesante darse cuenta, mientras se aprende a programar y entender cómo funcionan los dispositivos electrónicos, que el ser humano sólo es capaz de reproducir lo que conoce y por ello todos los sistemas electrónicos que usamos siguen leyes físicas que conocemos y se parecen a la manera de funcionar de los organismos vivos.

Las variables sirven para reservar un espacio en la memoria RAM y poder usarlo para el programa. Se llama memoria de acceso aleatorio porque el dispositivo es capaz de acceder a cualquier dato que contiene sin recorrer toda la memoria. No hay que pasar por todas las ubicaciones existentes para llegar a una en concreto, lo cual hace que sea más rápido guardar y rescatar información de la memoria. Esto sería como si para llegar a una ubicación concreta del supermercado no hubiese que recorrer todos y cada uno de los pasillos que nos separan de esa ubicación, la compra sería bastante más ágil y rápida (y acabaríamos no comprando esos malditos donuts en oferta que tan mal nos vienen para nuestra estilizada figura).

Resumiendo, al poner la siguiente instrucción...

```
name = "Alfredo"
```

… le estamos diciendo al dispositivo que ejecuta nuestro programa que guarde un hueco en su memoria para almacenar un dato, y que ese dato de momento será el texto *Alfredo*. Digo de momento porque una de las cosas interesantes es que las variables pueden cambiar su contenido, que a efectos prácticos significa que el espacio de memoria reservado puede cambiar su contenido.

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 ladrillos que para guardar clavos y, por extensión, cada caja ocupa un tamaño de la estantería y no conviene almacenar un clavo en una caja para ladrillos, ¡menuda pérdida de espacio!

De la misma forma, no necesitamos el mismo tamaño para guardar un número entero (con entero me refiero a no decimal) 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 tamaño del espacio necesario.

A esto se le denomina **tipado dinámico**. En otros lenguajes hay que indicar al declarar una variable si va a contener texto o un número (ladrillos o clavos).

Es interesante profundizar un poco más en el significado de por qué las variables llevan asociado un tamaño de uso en memoria. Imagina una estantería, para guardar libros, que no tiene baldas ni compartimentos. Si queremos situar baldas y compartimentos lo mejor es saber qué va a contener y realizarlo de forma que todo quepa de la forma más idónea posible y la colmemos con el mayor número de cosas antes de llenarla (sin seguir la filosofía minimalista de Marie Kondo). De esta forma la altura entre baldas tendrá que ser la del libro más alto, pero si todos los libros van a ser poco altos, o sólo vamos a almacenar DVDs, no tiene sentido poner las baldas con la altura de un libro de imágenes del National Geographic. Tampoco es útil dejar compartimentos para que quepan enciclopedias si todo lo que tenemos son colecciones de unos pocos libros.

Hoy en día es difícil llenar la memoria de un ordenador, pero antiguamente el usar los espacios necesarios y reservar la mayor cantidad de memoria era imprescindible, y aún sigue siendo algo importante en tecnologías como el Internet of Things y dispositivos de muy poco tamaño que tienen poca memoria.

Así que ya sabes, mejor utilizar la memoria de forma austera y concisa, ¡no derroches memoria!

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"
```

Como Python es un lenguaje anglosajón, usaremos el punto (.) como separador entre la parte entera y decimal de un número.

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)
```

Veamos el proceso ejecutado:

```
>>> name = "Alfredo"
>>> type (name)
<class 'str'>
>>>
```

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.

4. 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:

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

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:

```
>>> name = "Alfredo"
>>> type (name)
<class 'str'>
>>> name = 6
>>> type (name)
<class 'int'>
>>>
```

Si no has programado nunca, al llegar aquí será difícil que veas el gran potencial y utilidad de todo esto, pero a lo largo del curso irás viendo que es necesario conocer y jugar con el tipo de las variables. Ya hemos visto que la memoria es importante, y cualquiera entiende que no ocupa la misma cantidad de espacio un texto que un número.

5. 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 antes de continuar.

```
>>> number1, number2 = 6, 14
>>> print(number1+number2)
  20
>>> |
```

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.

```
>>> number1, number2 = 6, 14
>>> suma = number1 + number2
>>> print (suma)
  20
>>> |
```

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

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

```
word1, word2 = "Hello", "world"
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", "world"
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.

Existe otra posibilidad para imprimir las dos variables anteriores seguidas con un espacio entre medias, la opción de **concatenar**:

```
word1, word2 = "Hello", "world"
print (word1, word2)
```

El resultado es el siguiente:

```
>>> word1, word2 = "Hello", "world"
>>> print(word1, word2)
  Hello world
>>> |
```

Concatenar es una opción válida para imprimir diferentes variables de forma seguida, y siempre aparecerá un espacio separando cada valor a imprimir.

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.

```
Hola Mundo.py
1  number1 = input("Primer número a sumar: ")
2  number2 = input("Segundo número a sumar: ")
3  print (number1 + number2)

Shell
>>> %Run 'Hola Mundo.py'
  Primer número a sumar: 6
  Segundo número a sumar: 7
  67
>>> |
```

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 en el ejemplo de *Hello world*, 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.

```
>>> number1 = input("Primer número a sumar: ")
  Primer número a sumar: 6
>>> type (number1)
<class 'str'>
>>> |
```

Cuando el usuario introduce cualquier carácter desde el teclado, incluidos los números, Python considera que ese elemento es un string, por lo que la respuesta a un input siempre va a ser un string, y si lo almacenamos en una variable como en el ejemplo anterior, esa variable será de tipo string.

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 considerado 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, Thonny o la IDLE. Contesta un número a cada *input*.

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.

6. 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
```

Como verás, el resultado de una división, aunque sea un entero, se considera un decimal o *float*.

Suma:

```
2+3
5
```

Resta:

```
5-2
3
```

Potencia:

```
5**2
25
```

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
```

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
```

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, aunque importar una librería es básicamente importar un código ya creado e incorporarlo en el programa), en concreto, tenemos que importar la función *floor* de la librería *math:*

```
from math import floor
floor (4.75)
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 prueba las opciones matemáticas en la terminal, Thonny o en la IDLE.

7. DOCUMENTACIÓN OFICIAL DE PYTHON

La mayoría de lenguajes de programación tienen un espacio web con una gran base de documentación sobre funcionalidades del lenguaje, tutoriales, recomendaciones, etc.

Python no es una excepción, posee una gran cantidad de documentación. A todo lo existente en red sobre el lenguaje (cursos, tutoriales, foros de ayuda…) hay que sumarle una página oficial con documentación (en inglés) que contiene un gran número de reseñas a instrucciones propias del lenguaje.

La dirección para la documentación oficial de Python 3 es https://docs.python.org

Dentro del enlace verás muchos apartados. En estos primeros niveles lo más interesante lo encontrarás en la sección *Tutorial*. El orden de los contenidos no es el mismo que seguimos en éste curso, pero hay multitud de ejemplos y códigos para cada una de las funcionalidades de Python que veremos y para muchas otras que no tienen cabida en este curso pero que podrías necesitar en el futuro.

Sería una práctica excelente que, según vayas avanzando en el curso desarrolles la capacidad de buscar en la documentación oficial de Python para mejorar tus competencias con el lenguaje. Además, seguro que en multitud de ocasiones puedes resolver una duda o problema con un programa simplemente acudiendo a dicha documentación.

Una vez terminado el módulo puedes visionar los videos de la unidad para ver todo esto en práctica y posteriormente pasar al ejercicio propuesto para afianzar esta sesión.

8. 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)
```

9. 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 raíz cuadrada de un número negativo o no se puede dividir entre cero).