

Curso de iniciación a Python en Raspberry Pi



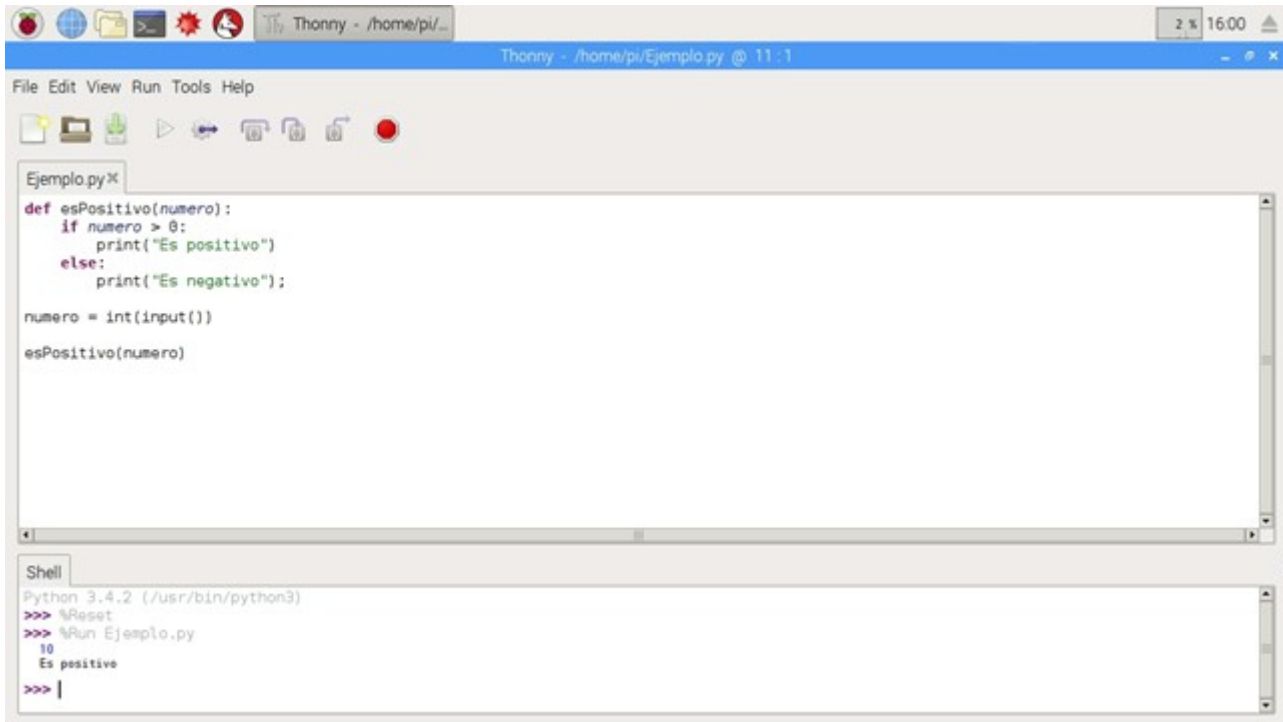
PROGRAMO
ERGO SUM



¿Qué voy a aprender en este curso?

En este curso aprenderás a **programar en Python utilizando la Raspberry Pi** para iniciarte en el mundo de la programación. Este es un tutorial de nivel iniciación para aquellos usuarios que todavía no han programado ningún lenguaje de programación en modo texto.

En el tutorial se va a explicar la versión de **Python 3 (IDLE)** que viene instalada por defecto en el sistema operativo de Raspbian para Raspberry Pi. No obstante, también podrás acceder al tutorial en caso de instalar Python en tu sistema operativo Windows, Linux o MAC.



También puedes acceder a la plataforma de [AprendeProgramando](#) para aprender a programar en Python a través de lecciones de teoría y práctica.

¿Qué materiales voy a necesitar?

Antes de continuar con las lecciones del curso asegúrate que dispones de todos los componentes y configuraciones que se van a necesitar:

- Raspberry Pi con el [sistema operativo Raspbian](#).

¿Qué es Python?

Python es un lenguaje de programación interpretado de tipado dinámico cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma y disponible en varias plataformas.

Dicho de otro modo, Python es:

- **Interpretado:** Se ejecuta sin necesidad de ser procesado por el compilador y se detectan los errores en tiempo de ejecución.
- **Multiparadigma:** Soporta [programación funcional](#), [programación imperativa](#) y [programación orientada a objetos](#).
- **Tipado dinámico:** Las variables se comprueban en tiempo de ejecución.
- **Multiplataforma:** disponible para plataformas de Windows, Linux o MAC.
- **Gratuito:** No dispone de licencia para programar.

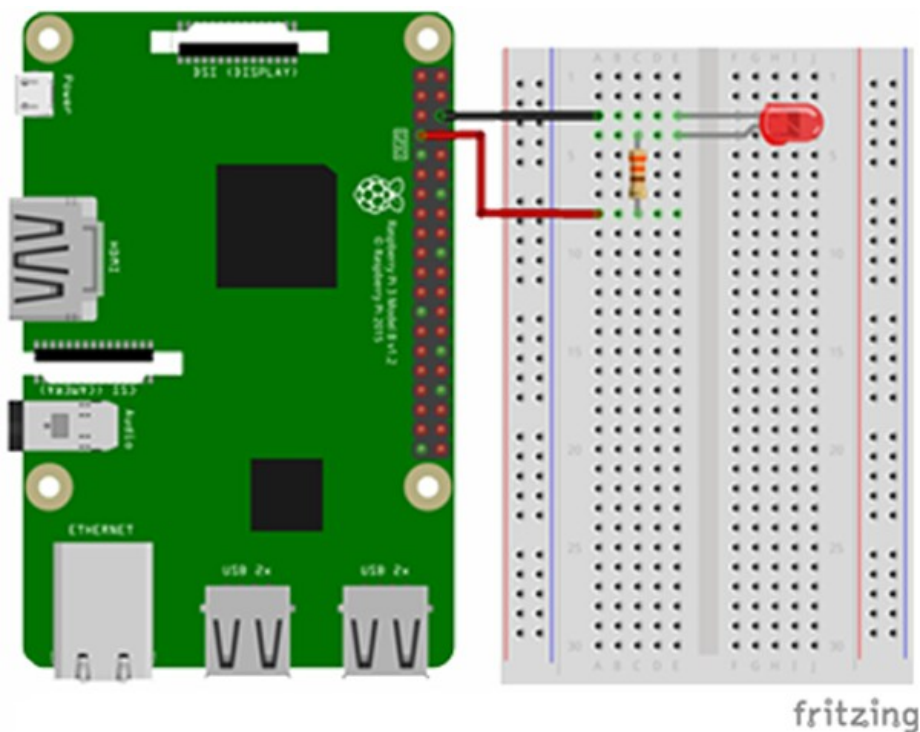
Al hacer uso de una sintaxis legible, la curva de aprendizaje es muy rápida, siendo de este modo, uno de los mejores lenguajes para iniciarse en la programación en modo texto. Por ejemplo, si comparamos un código escrito en lenguaje de programación por bloques como [Blockly](#) y el mismo código lo escribimos utilizando Python, vemos las similitudes en las instrucciones.



```
numero = 5
if numero > 3:
    print("SI se cumple")
else:
    print("NO se cumple")
```

Python contiene una gran cantidad de librerías, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas comunes sin necesidad de tener que programarlas desde cero. Pero lo que realmente le hace brillante utilizándolo en una Raspberry Pi, es por la **capacidad de poder utilizar los pines GPIO para conectar el mundo físico con el mundo digital**.

¡Recuerda! En esta misma plataforma tenemos un curso para aprender a utilizar los pines GPIO con [Scratch](#) y otro curso para utilizar los pines GPIO con [Python](#).



¿Qué es "Hola Mundo"?

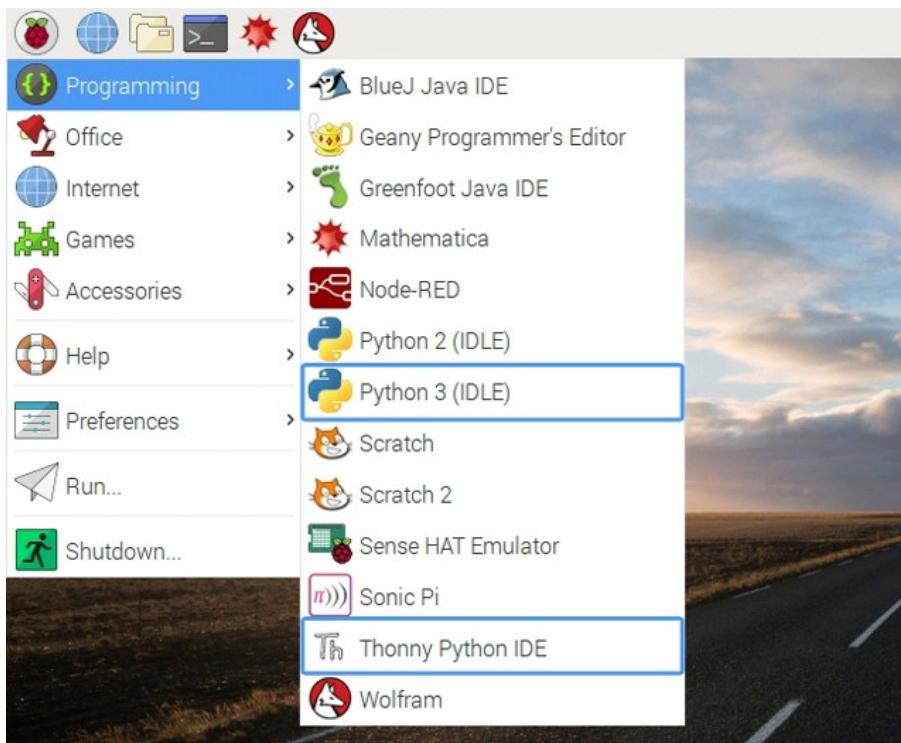
En programación, el primer programa que se crea es el que imprime el texto "Hola Mundo" en un dispositivo de visualización, terminal o shell. Este programa suele ser usado como introducción al estudio de un lenguaje de programación siendo un primer ejercicio típico, y considerándose fundamental desde el punto de vista didáctico.

El Hola Mundo se caracteriza por su sencillez, especialmente cuando se ejecuta en una terminal o shell.

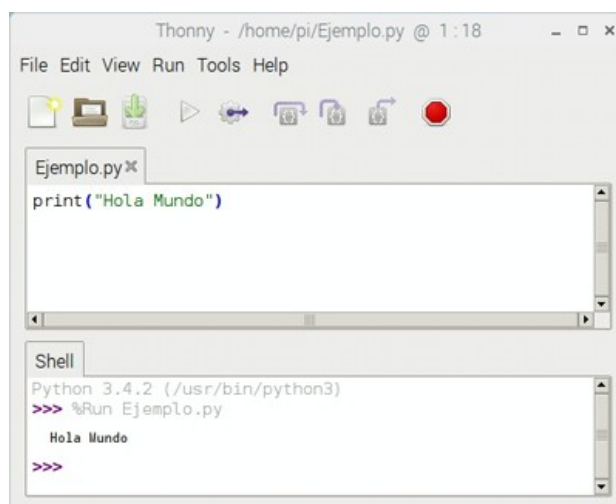
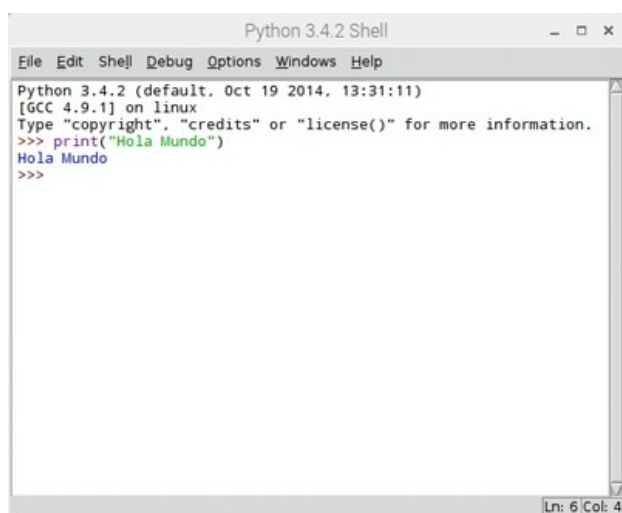
Si tenemos instalado el sistema operativo Raspbian en nuestra Raspberry Pi, encontraremos las versiones 2 y 3 de Python instaladas. En este tutorial se va a explicar la versión de **Python 3 (IDLE)**, con lo cual

recomendamos que si todavía no tienes instalada esta versión, sea el primer paso que hagas. Recuerda que Python es gratuito y multiplataforma.

Además, en este tutorial se utilizará el programa **Thony Python IDE** que viene instalado por defecto en la instalación de Raspbian. No obstante, puedes utilizar el IDE que más te gusta ya que no es importante para aprender a programar.



Suponiendo que tienes instalado Python 3 (IDLE) y Thonny Python IDE, accede a los programas y escribe tu primer programa. Recuerda que este programa mostrará por la pantalla el texto "Hola Mundo". Puedes probar a cambiar el texto y volver a ejecutar el código para ver que escribe lo que se le espera.



Operadores aritméticos

En la lección anterior se decía que Python es un lenguaje de programación que soporta el **paradigma de la programación funcional**. Esto quiere decir que está basada en una **programación declarativa de funciones matemáticas**.

La mayoría de lenguajes de programación utilizan los mismos operadores aritméticos.

Al hacer operaciones con números enteros y decimales el resultado es siempre decimal, por ejemplo, en el caso de dividir 6 entre 2. Por otro lado, en el caso de que el resultado no tenga parte decimal, Python escribe 0 como parte decimal para indicar que el resultado es un número decimal.

Suma	+
Resta	-
División	/
Multiplicación	*
Potencia	**
Resto	%
Cociente	//

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>> 3 - 3
0
>>> 3 * 2
6
>>> 3 ** 2
9
>>> 6 / 2
3.0
>>> 6 % 2
0
```

En siguientes lecciones veremos las principales funciones matemáticas utilizadas, como la raíz cuadrada de un número, el factorial de un número, redondeos, etc.

Operadores lógicos

Los operadores lógicos son utilizados en la mayoría de las ocasiones en las estructuras condicionales. A modo de ejemplo, podemos abrir el IDLE de Python y escribir directamente en la consola diferentes tipos de operaciones lógicas. Observa que los resultados que devuelven las operaciones lógicas son de tipo booleano, es decir, verdadero (True) o falso (False).

Igualdad	==
Distinto	!=
Mayor que	>
Mayor o igual que	>=
Menor que	<
Menor o igual que	<=

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 3 == 3
True
>>> 8 != 5
True
>>> 3 > 4
False
>>> 4 <= 6
True
>>>
```

Expresiones compuestas

En ocasiones nos vemos en la necesidad de comprobar varias igualdades, por ejemplo, cuando queremos calcular si un número está comprendido entre un rango de valores.

Las expresiones compuestas son utilizadas en las condiciones, bucles, etc. Se evalúa la condición y a continuación la expresión lógica. A modo de ejemplo, podemos abrir el IDLE de Python y escribir directamente en la consola diferentes tipos de expresiones compuestas.

Negación	not
Y	and
O	or

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> not True
False
>>> True and False
False
>>> True or False
True
```

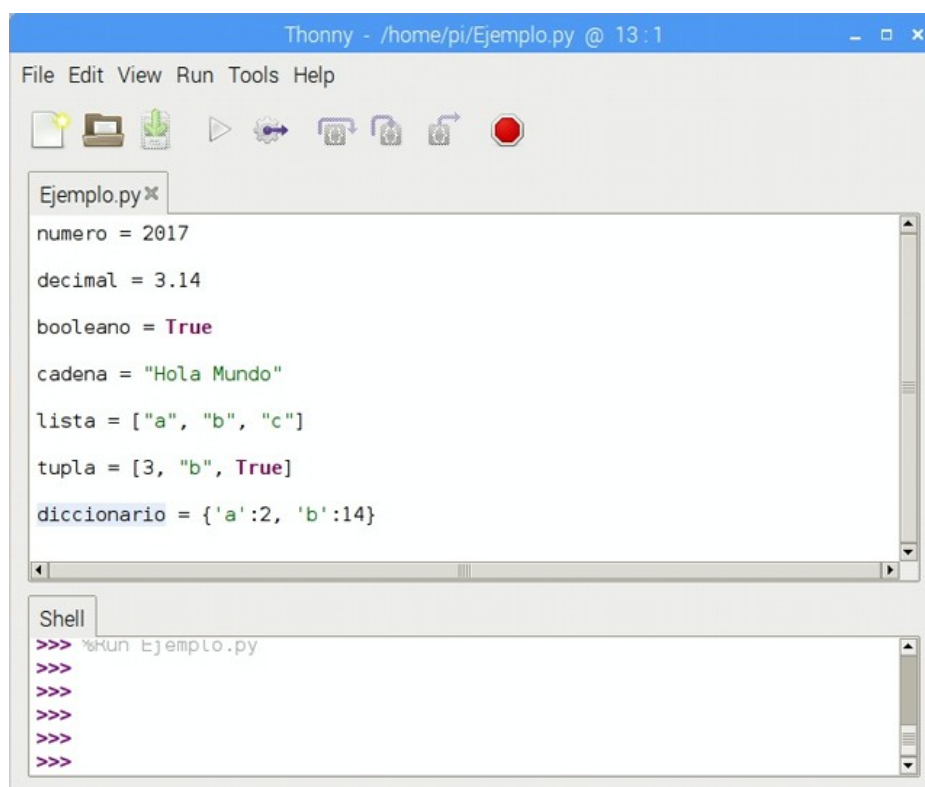
¿Qué es un tipo de dato?

En programación un tipo de dato es la forma de representación del dato e indica la clase de dato que se va a manejar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

Python es un lenguaje de programación de propósito general y por ello contiene una gran cantidad de tipos de datos con los que se puede programar.

- **Numéricos:** En Python los números pueden ser enteros, flotantes o complejos.

- **Booleanos:** Los tipos de datos booleanos se utilizan para representar verdadero y falso, mediante las palabras reservadas *True* o *False* respectivamente. Este tipo de datos es muy importante en para el control de flujo de un programa como veremos en las siguientes lecciones.
- **Cadenas:** Aunque las cadenas no son usualmente importantes para análisis numérico, sí lo son para mostrar resultados por la terminal o shell. Una cadena debe delimitarse con comillas simples o dobles y admite caracteres de escape.
- **Listas:** Una lista es una colección de objetos: números enteros, flotantes, complejos, cadenas, etc. Una lista se delimita utilizando `[]` y sus elementos han de separarse con comas. Es posible acceder a sus elementos indicando el índice del elemento deseado.
- **Tuplas:** En muchos sentidos una tupla es como una lista, contienen una colección de objetos de distinto tipo. Es decir, son lista de elementos de diferente tipo de dato.
- **Diccionario:** Un diccionario se compone de dos partes: una llave (palabra) y un valor (definición). Las llaves siempre deben ser un tipo de dato primitivo. La llave y el valor se separan con `:` y sus elementos con comas.



```

Thonny - /home/pi/Ejemplo.py @ 13:1
File Edit View Run Tools Help

Ejemplo.py
numero = 2017
decimal = 3.14
booleano = True
cadena = "Hola Mundo"
lista = ["a", "b", "c"]
tupla = (3, "b", True)
diccionario = {'a':2, 'b':14}

Shell
>>> %run Ejemplo.py
>>>
>>>
>>>
>>>
>>>

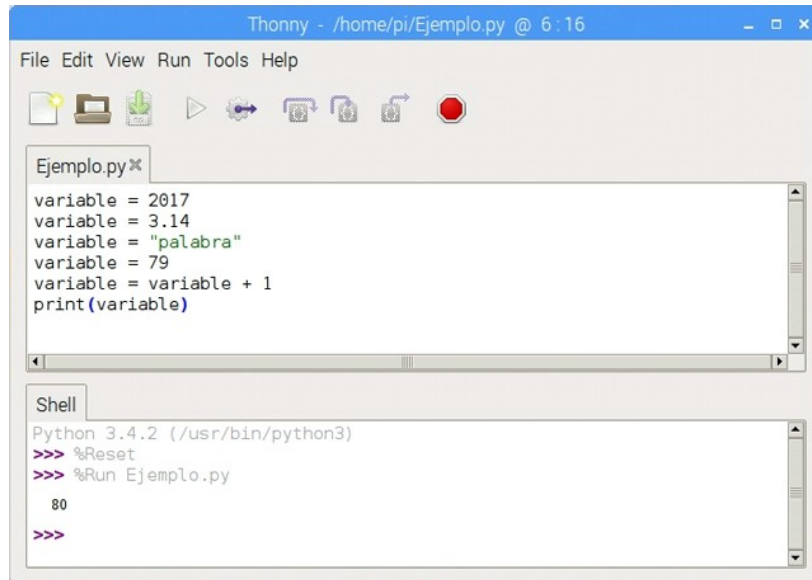
```

¿Qué es una variable?

Se define como variable al **espacio reservado de la memoria que almacena un dato**, que como su propio nombre indica, puede cambiar de valor en tiempo de ejecución. En Python tenemos tipos de datos simples (números enteros, de coma flotante) cadenas de texto, y valores booleanos. Todos estos pueden almacenarse en la misma variable sin especificar de antemano el tipo de datos que almacenará, de ahí a que Python sea un lenguaje de programación de tipado dinámico, como se explica en la primera lección de este curso.

Python es un lenguaje de programación de tipado dinámico. Las variables se comprueban en tiempo de ejecución.

Ejecutando el siguiente ejemplo, podemos observar que la variable llamada "variable" va tomando diferentes valores en las diferentes líneas del programa. En la última línea se imprime el valor, que al ser dinámico obtiene el valor de las dos anteriores instrucciones, es decir, $79 + 1$.



The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations and execution. The main editor window displays a file named 'Ejemplo.py' with the following code:

```
variable = 2017
variable = 3.14
variable = "palabra"
variable = 79
variable = variable + 1
print(variable)
```

Below the editor is a Shell window showing the execution of the script:

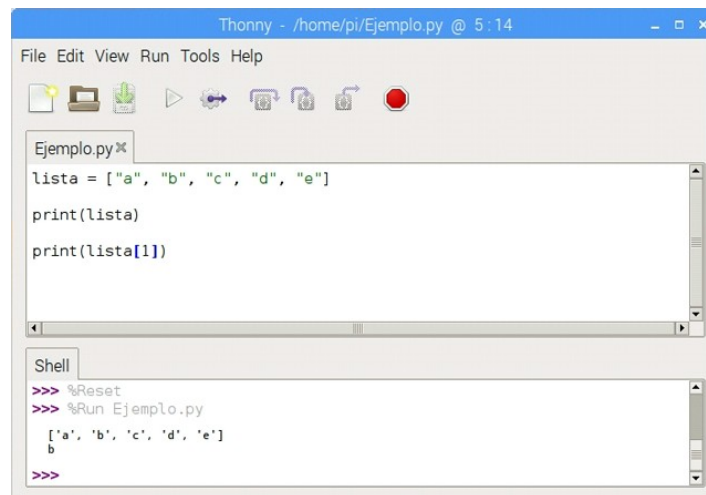
```
Python 3.4.2 (/usr/bin/python3)
>>> %Reset
>>> %Run Ejemplo.py
80
>>>
```

¿Qué es una lista?

En Python también disponemos de listas (colección de elementos), que aunque no forman parte del concepto de tipos primitivos, se pueden explicar en este nivel iniciación del curso.

Las listas son tipos de datos abstractos que se verán en tutoriales más avanzados de Python.

En el siguiente ejemplo vamos a crear una lista llamada "lista" que va a contener los caracteres "a, b, c, d y e". Este mismo ejemplo se podría reutilizar para crear una lista de personas, lista de objetos, e incluso una lista donde aparezcan diferentes tipos de datos (números, palabras, e incluso otras listas).



The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations and execution. The main editor window displays a file named 'Ejemplo.py' with the following code:

```
lista = ["a", "b", "c", "d", "e"]
print(lista)
print(lista[1])
```

Below the editor is a Shell window showing the execution of the script:

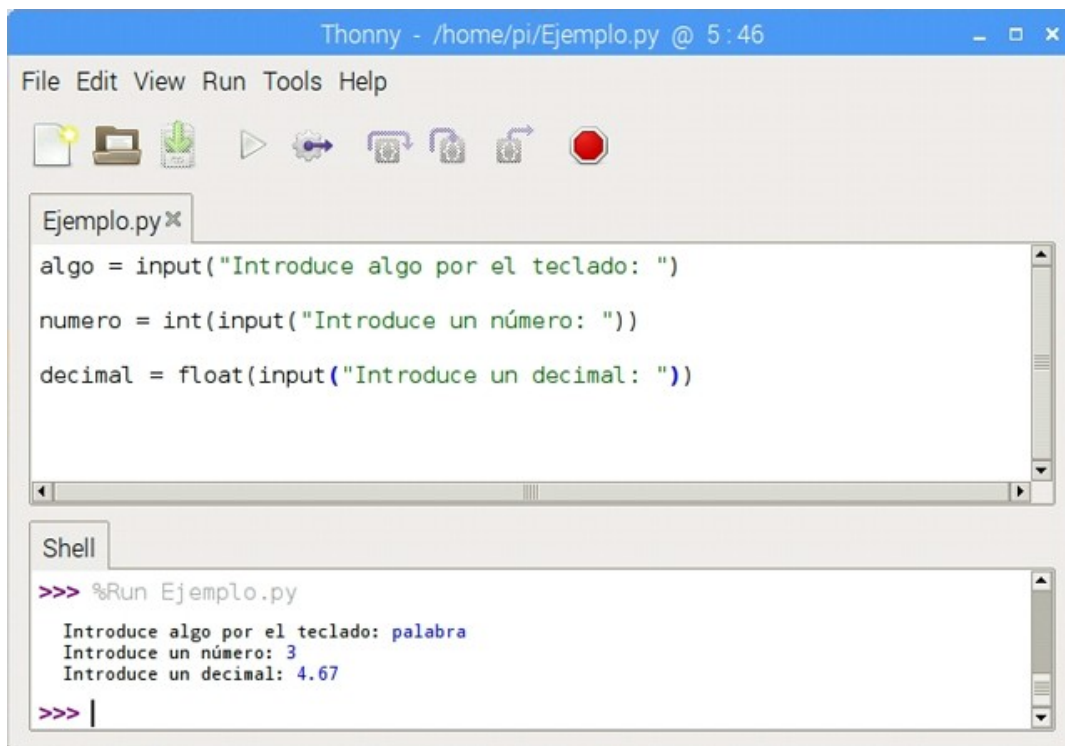
```
>>> %Reset
>>> %Run Ejemplo.py
['a', 'b', 'c', 'd', 'e']
b
>>>
```

¿Cómo solicitar datos por teclado?

En la mayoría de los programas necesitamos interactuar con el usuario para solicitarle diferentes datos. Para ello vamos a utilizar la función prevista en Python para tal fin.

Sabiendo lo que se explicaba en la anterior lección referente al tipo de datos, en este caso debemos saber qué es lo que se le solicita al usuario para saber qué es lo que nos tiene que proporcionar. De esta forma al dato introducido deberemos realizar un casting acorde con el dato esperado.

Al procedimiento de transformar una variable primitiva a otro tipo se le conoce como casting.



The screenshot shows the Thonny IDE interface. The top bar indicates the file path is `/home/pi/Ejemplo.py` and the time is 5:46. The menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for opening files, saving, running, and other functions. The main editor window displays the following Python code:

```
Ejemplo.py
algo = input("Introduce algo por el teclado: ")
numero = int(input("Introduce un número: "))
decimal = float(input("Introduce un decimal: "))
```

Below the editor is a Shell window showing the execution of the script:

```
>>> %Run Ejemplo.py
Introduce algo por el teclado: palabra
Introduce un número: 3
Introduce un decimal: 4.67
>>> |
```

Es muy importante comprobar el tipo de dato ya que como se explica en la primera lección, Python es un lenguaje de programación interpretado y sus variables se comprueban en tiempo de ejecución y podrían dar errores durante el funcionamiento del programa.

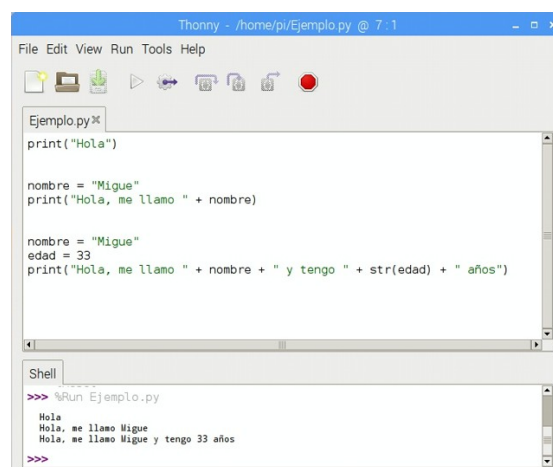
¿Cómo imprimir datos por pantalla?

En la primera lección del curso vimos como imprimir el texto *"Hola Mundo"* por la terminal o shell de Python. Sin embargo en ocasiones no solamente queremos imprimir una frase, sino que además queremos concatenar diferentes tipos de datos para mostrar una única línea de texto.

Por ejemplo, supongamos que queremos mostrar en una única línea un texto seguido de nuestro nombre. En este caso deberemos concatenar el valor de la frase con el valor de la variable, y utilizaremos el símbolo reservado `"+"`.

Pero además, podemos llegar más lejos, en el caso de querer mostrar un texto y concatenar un número entero o en coma flotante. Si recuerdas la lección anterior, cuando se hablaba de tipos de datos, si intentamos imprimir un texto con un número el intérprete de Python mostrará un error en tiempo de ejecución por la compatibilidad de tipos de datos. Para solucionarlo, tenemos que decirle al intérprete el tipo de dato que queremos mostrar, y para ello, se debe convertir a tipo texto, haciendo un casting sobre él, como se explica en la anterior lección.

Al procedimiento de transformar una variable primitiva a otro tipo se le conoce como casting.



The screenshot shows the Thonny IDE interface. The top bar indicates the file path is `/home/pi/Ejemplo.py` and the time is 7:1. The menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for opening files, saving, running, and other functions. The main editor window displays the following Python code:

```
Ejemplo.py
print("Hola")

nombre = "Migue"
print("Hola, me llamo " + nombre)

nombre = "Migue"
edad = 33
print("Hola, me llamo " + nombre + " y tengo " + str(edad) + " años")
```

Below the editor is a Shell window showing the execution of the script:

```
>>> %Run Ejemplo.py
Hola
Hola, me llamo Migue
Hola, me llamo Migue y tengo 33 años
>>>
```


¿Qué es una condición?

En programación, una **condición es toda sentencia de la cual se puede determinar su veracidad** (True) o falsedad (False). En la mayoría de las ocasiones son comparaciones que podemos encontrar tanto en las condiciones como en bucles.

A diferencia que en otros lenguajes de programación, en Python no existen llaves para incluir las instrucciones de la condición. En Python es obligatorio indentar las instrucciones que forman parte de una condición, bucle, función, etc. De esta forma el código queda muy legible como se comentaba en la primera lección del curso.

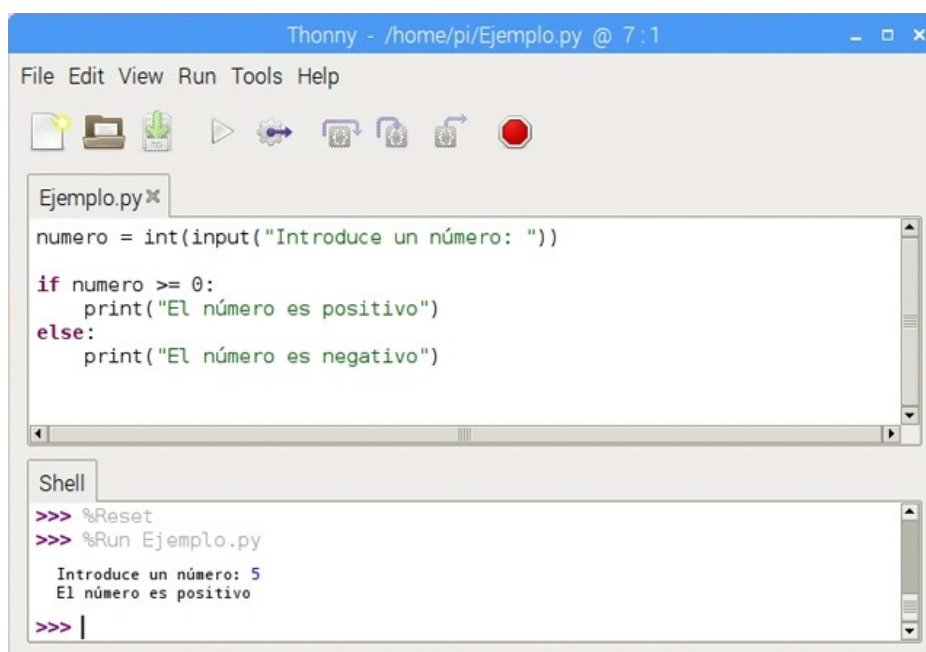
Otra diferencia de Python sobre el resto de lenguajes de programación, es que en Python no existen los famosos ";" que se sitúan al final de las instrucciones. Sin embargo, en el caso de las condiciones, bucles o funciones debemos añadir ":" para indicarle al interprete que empieza una estructura.



```
numero = 5
if numero > 3:
    print("SI se cumple")
else:
    print("NO se cumple")
```

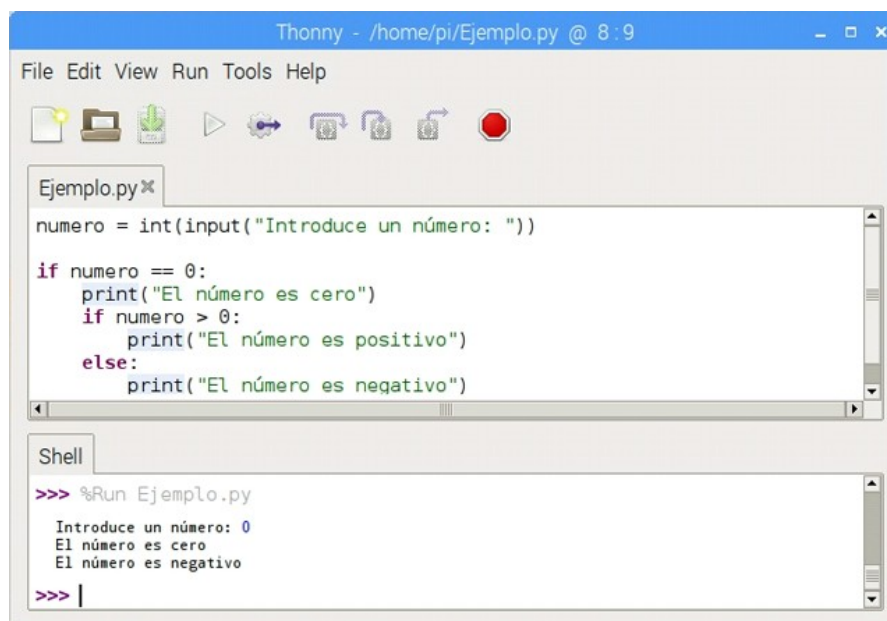
```
var numero = 5;
if(numero > 3){
    print("SI se cumple");
}else{
    print("NO se cumple");
}
```

El ejemplo más sencillo lo podemos ver cuando queremos determinar si un número introducido por el usuario es positivo o negativo. La comparación en este caso consiste en comprobar si el número es mayor o igual que 0 para el caso del positivo, y en caso contrario será negativo.



Sin embargo, en ciertas ocasiones queremos comprobar varios casos, como en este ejemplo, donde queremos saber si un número es positivo, negativo o neutro. A este tipo de condición se le conoce como condición anidada ya que aparece una condición en el interior de otra indentada correctamente.

Es importante no caer en la tentación de anidar condiciones tras condiciones, podríamos caer en lo que se conoce como código espagueti.



```
Thonny - /home/pi/Ejemplo.py @ 8:9
File Edit View Run Tools Help

Ejemplo.py
numero = int(input("Introduce un número: "))

if numero == 0:
    print("El número es cero")
if numero > 0:
    print("El número es positivo")
else:
    print("El número es negativo")

Shell
>>> %Run Ejemplo.py
Introduce un número: 0
El número es cero
El número es negativo
>>> |
```

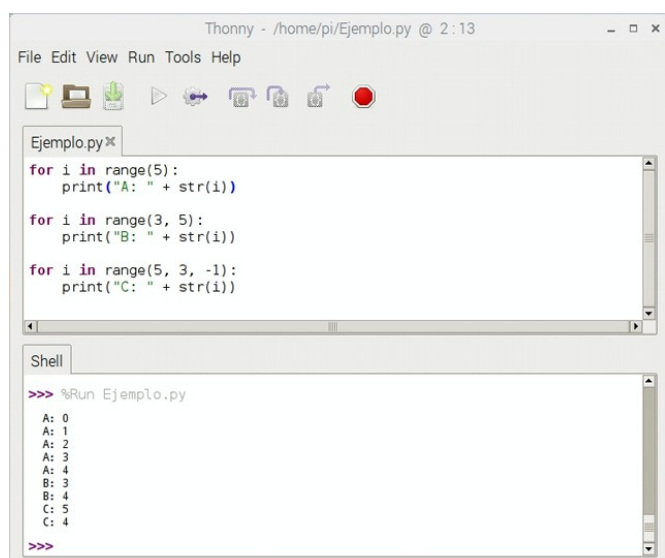
¿Qué es un bucle?

En programación, un bucle es una sentencia que ejecuta repetidas veces un conjunto de código, hasta que la condición asignada a dicho bucle deja de cumplirse. Los tres bucles más utilizados en programación son el bucle *while*, el bucle *for* y el bucle *do-while*.

Bucle (for)

Un bucle for es un bucle que repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

En los siguientes ejemplos se aprecia como podemos mostrar una determinada cantidad de iteraciones mediante la función range, o por el contrario como podemos recorrer una lista para mostrar todos los elementos de la misma.



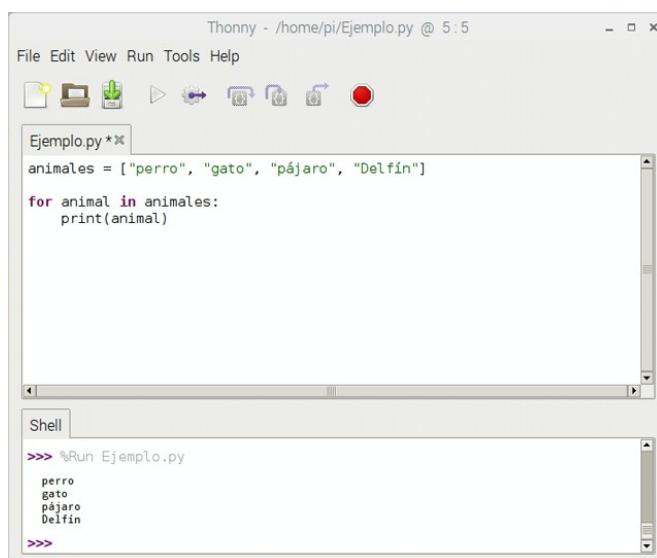
```
Thonny - /home/pi/Ejemplo.py @ 2:13
File Edit View Run Tools Help

Ejemplo.py
for i in range(5):
    print("A: " + str(i))

for i in range(3, 5):
    print("B: " + str(i))

for i in range(5, 3, -1):
    print("C: " + str(i))

Shell
>>> %Run Ejemplo.py
A: 0
A: 1
A: 2
A: 3
A: 4
B: 3
B: 4
C: 5
C: 4
C: 3
>>>
```



```
Thonny - /home/pi/Ejemplo.py @ 5:5
File Edit View Run Tools Help

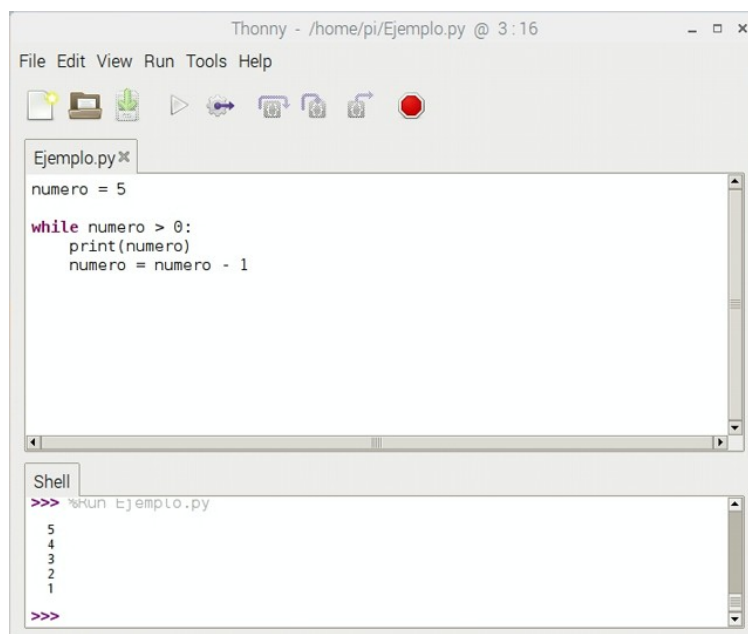
Ejemplo.py
animales = ["perro", "gato", "pájaro", "Delfín"]

for animal in animales:
    print(animal)

Shell
>>> %Run Ejemplo.py
perro
gato
pájaro
Delfín
>>>
```

Bucle (while)

Un bucle while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True). En este tipo de bucle hay que llevar cuidado porque es muy común caer en bucles infinitos cuando se empieza a programar. Es decir, fíjate como en el bucle de la siguiente imagen, decrementamos el valor de número en cada iteración.



```
Thonny - /home/pi/Ejemplo.py @ 3:16
File Edit View Run Tools Help

Ejemplo.py
numero = 5

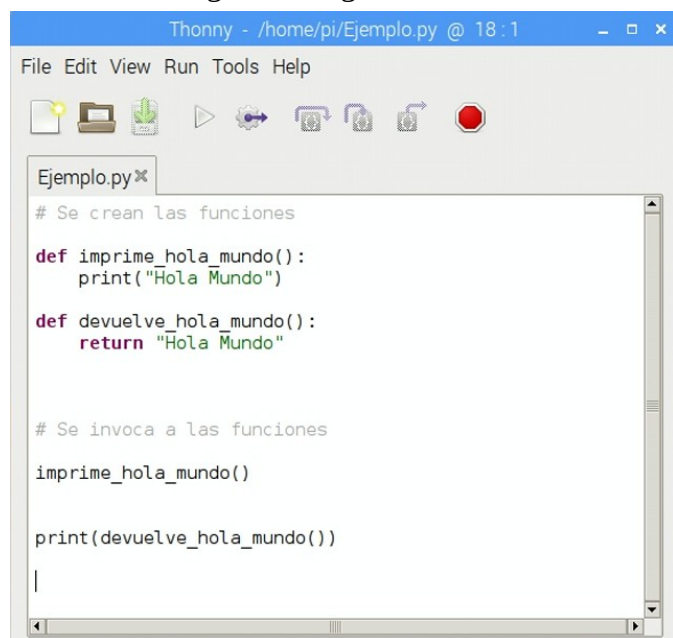
while numero > 0:
    print(numero)
    numero = numero - 1

Shell
>>> %run Ejemplo.py
5
4
3
2
1
>>>
```

¿Qué es una función?

En programación una función es un conjunto aislado de instrucciones que realizan una determinada función, como el propio nombre indica.

Las funciones pueden realizar una operación en el interior o devolver el resultado de la operación para tratarlo desde otro punto de la programación. Además, una función puede recibir diferentes parámetros como se muestra en la segunda imagen.



```
Thonny - /home/pi/Ejemplo.py @ 18:1
File Edit View Run Tools Help

Ejemplo.py
# Se crean las funciones

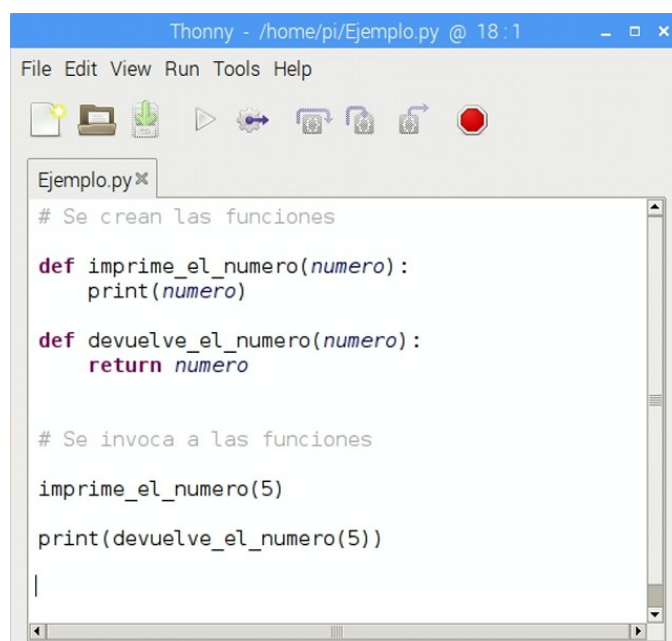
def imprimir_hola_mundo():
    print("Hola Mundo")

def devuelve_hola_mundo():
    return "Hola Mundo"

# Se invoca a las funciones

imprimir_hola_mundo()

print(devuelve_hola_mundo())
|
```



```
Thonny - /home/pi/Ejemplo.py @ 18:1
File Edit View Run Tools Help

Ejemplo.py
# Se crean las funciones

def imprimir_el_numero(numero):
    print(numero)

def devuelve_el_numero(numero):
    return numero

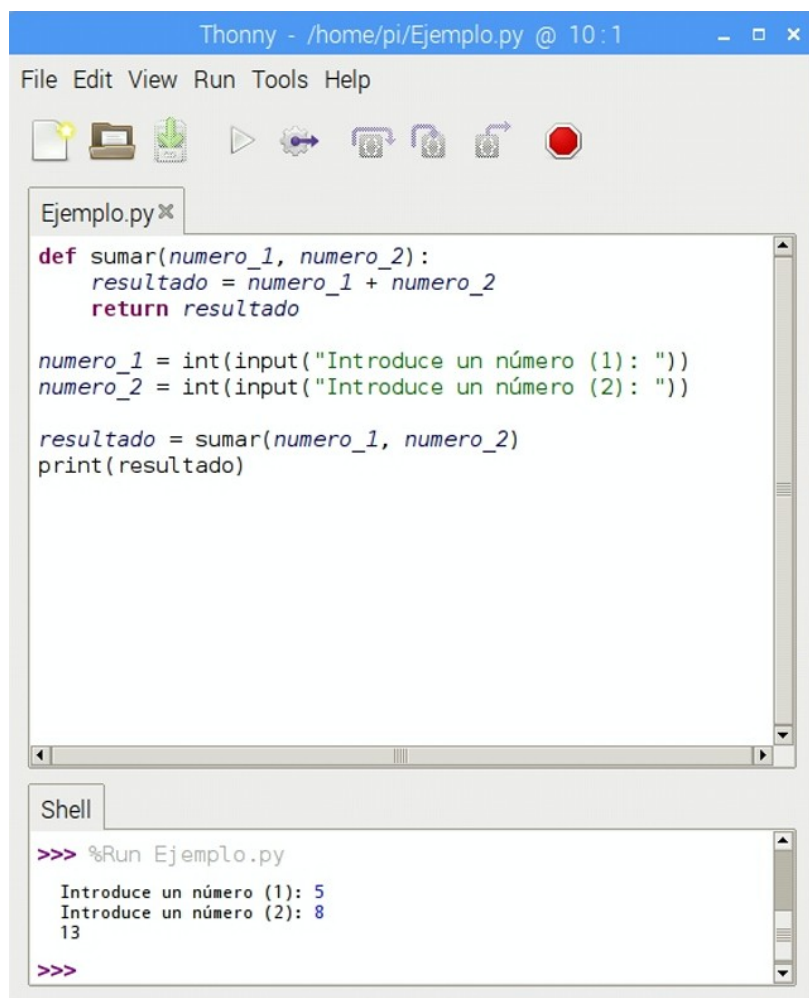
# Se invoca a las funciones

imprimir_el_numero(5)

print(devuelve_el_numero(5))
|
```

En el siguiente ejemplo se puede observar como se ha creado una función llamada *sumar* la cual además recibe dos números por parámetro *numero_1* y *numero_2*. En el interior de la función realiza la operación de la suma de ambos números y por último devuelve el resultado mediante la instrucción *return*.

En las siguientes líneas solicitamos 2 números al usuario e invocamos a la función *sumar* pasando como parámetros los números introducidos por teclado. Como hemos dicho que la función devuelve un resultado, este lo guardamos en una nueva variable llamada *resultado* (que aunque tiene el mismo nombre que la variable *resultado* de la función *sumar*, son variables distintas). Por último imprimimos el resultado.



The screenshot shows the Thonny Python IDE interface. The top window, titled 'Ejemplo.py', contains the following Python code:

```
def sumar(numero_1, numero_2):  
    resultado = numero_1 + numero_2  
    return resultado  
  
numero_1 = int(input("Introduce un número (1): "))  
numero_2 = int(input("Introduce un número (2): "))  
  
resultado = sumar(numero_1, numero_2)  
print(resultado)
```

The bottom window, titled 'Shell', shows the execution of the script:

```
>>> %Run Ejemplo.py  
Introduce un número (1): 5  
Introduce un número (2): 8  
13  
>>>
```

Ejercicios de iniciación a la programación en Python

Variables

1. Crea una variable que contenga el año actual y muéstralo por pantalla.
2. Crea una variable con el valor de PI (3.14....) e imprímela.
3. Crea una variable con el valor Verdadero (True) e imprímela.
4. Crea una lista de 5 animales y muéstrala.
5. Crea una lista de 5 frutas y muestra solamente la última fruta.

Operaciones

1. Pedir dos números por teclado e imprimir la suma de ambos.
2. Pedir dos números por teclado e imprimir la media aritmética.
3. Pedir peso y altura para calcular la masa corporal: $mc = \text{peso} / \text{altura}^2$.
4. Pedir radio para calcular la circunferencia de un círculo: $C = 2 * \pi * r$.
5. Pedir un número en Celsius y convertir a Fahrenheit: $F = 1.8 * C + 32$.

Condiciones

1. Pedir dos números por teclado y decir cual es mayor.
2. Pedir un número por teclado y decir si es par o impar.
3. Pedir tres números por teclado e imprimir el mayor de ellos solamente.
4. Pedir un número por pantalla y decir si está entre 10 y 15 o no.
5. Pedir lado y alto de un cuadrilátero y decir si es cuadrado o rectángulo.

Bucles

1. Imprimir los 25 primeros números naturales.
2. Imprimir los números impares desde el 1 al 25, ambos inclusive.
3. Calcula e imprime la suma desde el 14 hasta el 38, ambos inclusive.
4. Calcula e imprime el producto de la serie $2 \times 4 \times 6 \times 8 \times \dots \times 20$.
5. Calcula e imprime la suma de la serie $50 + 48 + 46 + \dots + 20$.

Funciones

1. Crea una función que reciba un número e imprima si es par o impar.
2. Modifica la función anterior para que en vez de imprimirlo lo devuelva.
3. Crea una función que reciba 2 números, devuelve el mayor e imprímelo.
4. Crea una función que reciba 2 números y devuelva el resto de la división del primer número dividido entre el segundo. Imprime el resultado.
5. Crea una función que reciba la base y la altura de un triángulo y devuelva su área. $A = \frac{1}{2}bh$.
6. Crea una función para calcular el IVA de un producto. Deberá recibir un precio y devolver el precio IVA incluido.
7. Crea una función que reciba un número, calcule su factorial, devuelva el resultado e imprímelo.
8. Crea una función que reciba un número y calcule si es primo o no.