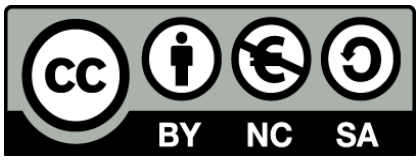
Manual básico, iniciación a Python 3

Manual básico, iniciación a Python 3
por José Miguel Ruiz Torres – jmruizt.94@gmail.com



Índice de contenido

Unas palabras del autor

Escribo este manual con la intención de que sirva como herramienta de aprendizaje para aquellas personas que deseen iniciarse en la programación con Python.
Aquí se irán explicando los diferentes conceptos conforme sea necesario, en un lenguaje coloquial, procurando así no agobiar al lector con una lluvia de definiciones y tecnicismos que una persona normal es incapaz de entender.

Parto de que el lector no tiene ningún conocimiento de programación e intento transmitirle lo imprescindible para que sea capaz de progresar por sí mismo.

Es mi deseo poner esta obra a disposición de toda persona que desee compartirla o adaptarla, siempre que se respeten las condiciones anteriormente citadas.

Python es un lenguaje de programación que puede usarse para crear programas en diferentes sistemas operativos como Linux, Mac OS y Windows.
Su filosofía es ser un lenguaje sencillo a la vez que elegante y flexible.

Python es realmente potente y puede usarse para casi cualquier propósito, desde la creación de un diccionario hasta un videojuego y mucho más.

Espero querido lector que te diviertas y aprendas leyendo las líneas de este libro.

Un cordial saludo.

José Miguel Ruiz Torres

Antes de empezar

Antes de empezar a programar necesitaremos instalar Python en el ordenador. Como no sé qué sistema operativo está usando el lector, explicaré cómo proceder a la instalación en cada caso.

Instalación en Linux

En Linux la instalación resulta realmente sencilla. Si eres usuario de Ubuntu (Debian y derivados), basta con abrir la terminal y escribir lo siguiente:

sudo apt-get install python3

Si eres usuario de otra distribución no puedo guiarte desde aquí, ya que puede variar el gestor de paquetes y el nombre de la paquetería. Escríbeme a mi correo si necesitas ayuda y estaré encantado de echarte un cable.

Instalación en Mac OS

Python viene preinstalado en este sistema. Si tu versión estuviera desactualizada deberás descargar uno de los siguientes paquetes según te convenga:

Si tienes un Macintosh con procesador Intel debes descargar el siguiente instalador:

http://www.python.org/ftp/python/3.2.3/python-3.2.3-macosx10.6.dmg

Si por el contrario tu máquina tiene un procesador PowerPC, debes descargar este:

http://www.python.org/ftp/python/3.2.3/python-3.2.3-macosx10.3.dmg

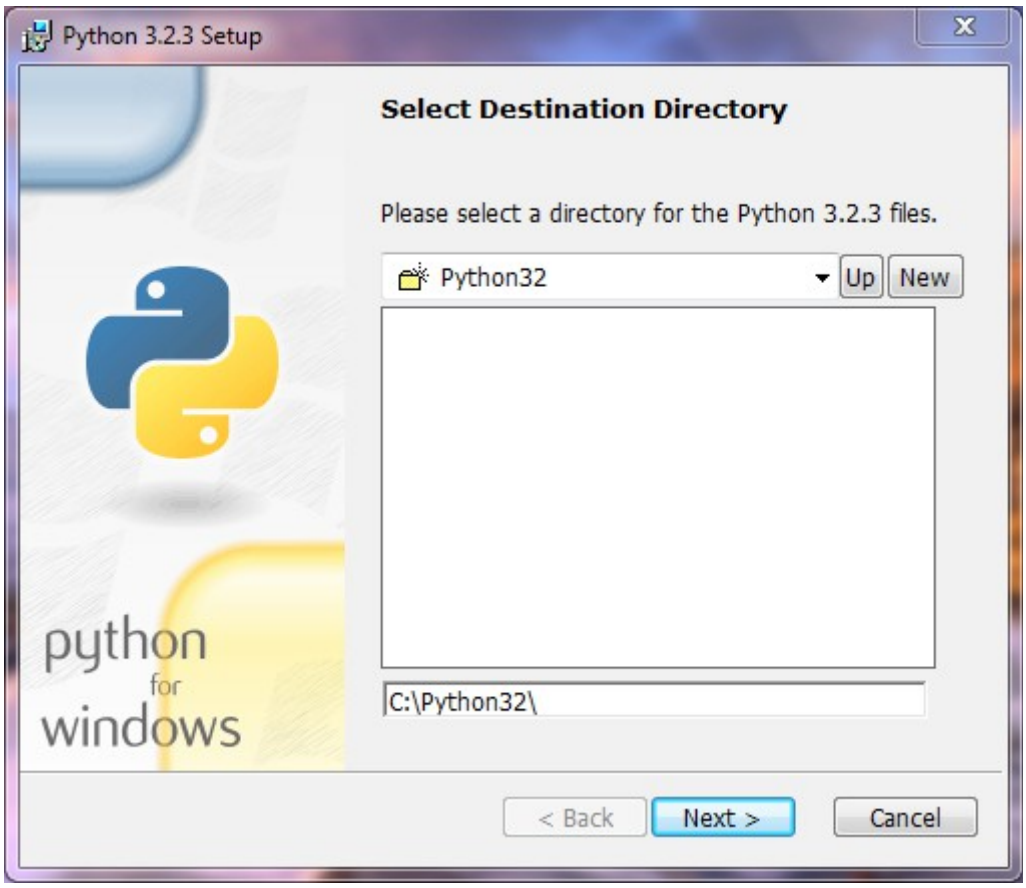
No tengo el privilegio de poseer estas máquinas, así que más no puedo hacer.

Instalación en Windows

Por suerte (o por desgracia) todo el mundo dispone de una copia de Windows en casa, incluso yo; así que en esta instalación si os puedo guiar.

Python no viene preinstalado en Windows, por lo que obligatoriamente deberás descargar este paquete:

http://www.python.org/ftp/python/3.2.3/python-3.2.3.msi



La instalación en Windows no tiene mayor complicación: basta con hacer clic en “Siguiente” repetidas veces hasta finalizar.

Y con esto ya tenemos instalado Python en nuestro ordenador.

¿Cómo hago mi programa?

Un programa está compuesto por códigos, órdenes simples que las personas podemos comprender. Por ejemplo:

- Lenguaje humano:

Orden: Dime, ¿cuánto son 2+2?
Respuesta: Son 4.

- Lenguaje Python:

Orden: print(2+2)
Respuesta: 4

Fácil, ¿verdad?.

Para escribir dichos códigos nosotros usaremos un editor de texto. ¿Conoces el *bloc de notas* de Windows? pues ese sería un ejemplo.

Pero el *bloc de notas* es muy malo, así que nosotros vamos a usar otro editor.

Editores para Linux

En Linux tenemos magníficos editores como *Gedit* (GNOME) y *Kate* (KDE).
Si eres un enamorado de la terminal también puedes usar *Pico*.

Asumo que los usuarios de Linux saben encontrar estos programas en su sistema o, en caso de no tenerlos, saben instalarlos.

Editores para Mac OS

Los usuarios de Mac también pueden disfrutar de *Gedit*:

- Para Tiger (10.4, Intel): descargar
- Para Leopard (10.5): descargar
- Para Snow Leopard (10.6) y posterior: descargar

Como ya dije antes, no estoy metido en el mundo de Apple. Seguro de que los maqueros conocen editores mucho mejores para su sistema.

Editores para Windows

Para Windows hay gran variedad, aunque yo me decanto por estos dos:

- Gedit: descargar
- Notepad++: descargar

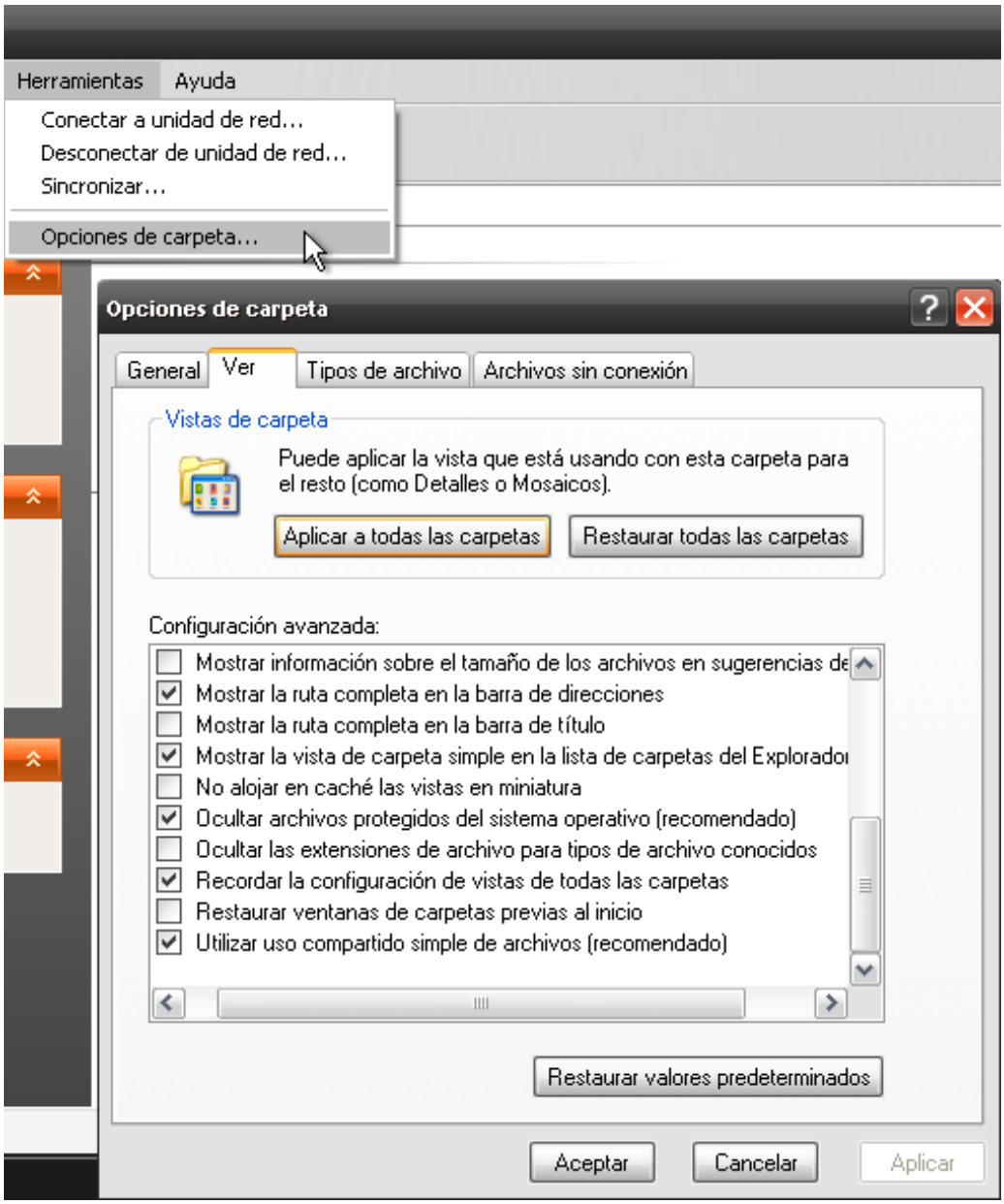
Mi recomendación es que vayáis probando y os quedéis con el que os resulte más cómodo. Si ninguno os convence, entonces podéis buscar alternativas en Internet.

Con esto ya tenemos lo necesario para empezar a aprender.

Comenzamos

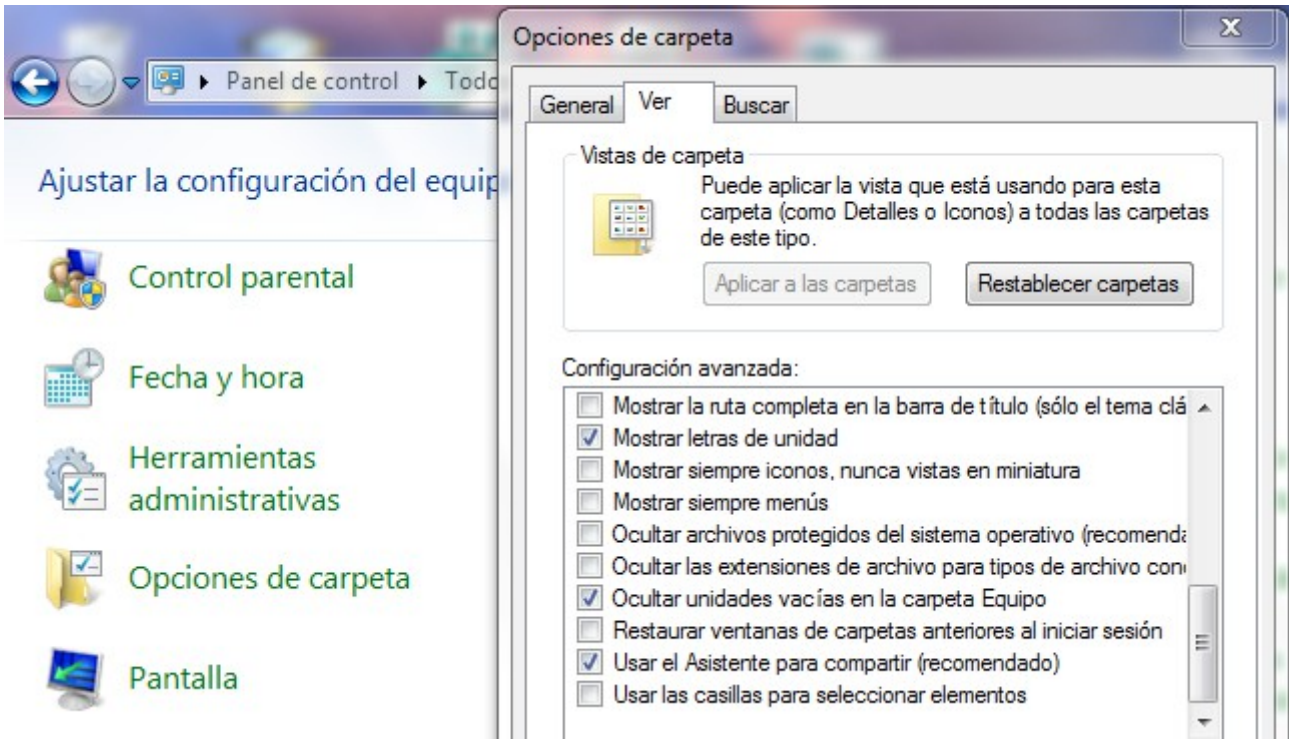
Lo primero que haremos será crear un nuevo archivo llamado “hola.py”.
En Windows puede surgir alguna complicación, ya que por defecto el sistema no permite ver/cambiar la extensión a los archivos.

Cómo cambiar esto en Windows XP:



Abre el navegador de archivos y, como se muestra en la imagen, accede a “Herramientas → Opciones de carpeta...”. Ve a la pestaña “Ver” y desmarca la casilla que dice “Ocultar las extensiones de archivo para tipos de archivo conocido”.

Cómo cambiar esto en Windows 7:



Ve a “Inicio → Panel de control”, ahí cambia la vista a “Iconos grandes” y entra en “Opciones de carpeta”. En la nueva ventana ve a la pestaña “Ver” y ahí desmarca la casilla de “Ocultar las extensiones de archivo para tipos de archivo conocidos”.

Hecho esto, abre el archivo “hola.py” con tu editor de texto favorito.

Los doce mandamientos

He aquí unas sencillas normas que deben respetarse para un correcto estilo de programación en Python. Realmente serían doce, pero yo he decidido suprimir algunas para no confundir al lector.

Al principio es probable que no entiendas nada, pero conforme vayas avanzando en el libro todo cobrará sentido. Ahora es bueno que, por lo menos, te vaya sonando:

1. Los nombres de los ficheros deben escribirse en minúscula. Ejemplo:

   hola.py calculadora.py

2. Las llamadas a funciones se escriben en minúscula. Ejemplo:

   print() input()

3. Las variables se escriben en minúscula y, de estar formadas por varias palabras, éstas van unidas por guiones bajos. Ejemplo:

   balones piezas_de_repuesto

4. Los tipos de dato se escriben en minúscula. Ejemplo:

   str int

5. Pon un espacio después de cada coma. Ejemplo:

   ```
   print('Tienes', num_zapatos, 'zapatos y', num_blusas, 'blusas.')
   ```

6. Pon un espacio antes y después de cada operador. Ejemplo:

   ```
   8 + 2  total += 6
   ```

7. Indenta con 4 espacios; **nunca** uses el tabulador. Ejemplo:

   ```
   if nombre == 'Ángel':
       print('Hola', nombre)
   1234
   input()
   ```

8. Escribe abundantes comentarios en tu código, describiendo cada detalle, para hacer que sea lo más claro y legible posible.

Salida de información

Texto

Todo programa hace una serie de acciones básicas. Una de estas acciones es la de mostrar información: texto, números, resultados... es algo imprescindible.

Antes dijimos que un programa se componía de pequeñas líneas de código, órdenes simples. En adelante llamaremos a estas órdenes *funciones*.

Vamos a hacer nuestro primer programa. Este consistirá en mostrar un texto.

Para mostrar texto en Python usamos la función *print*, cuya sintaxis es:

print('texto')

Ejemplo en lenguaje humano:

Orden: Dime “Me llamo Sergio y tengo 16 años”
Respuesta: Me llamo Sergio y tengo 16 años

Lenguaje Python:

Orden: print('Me llamo Sergio y tengo 16 años')
Respuesta: Me llamo Sergio y tengo 16 años

Recuerda: *print('el texto que quieres mostrar')*

Comentarios

Los comentarios son anotaciones que hace el programador en su código fuente. Sirven para que al leer el código, otros programadores puedan entenderlo fácilmente (o incluso el autor original, al volver a leerlo pasado un tiempo).

En Python los comentarios se hacen con una almohadilla (#). Ejemplo:

```
# Solo afecta al texto que va a continuación de la almohadilla.
print('Hola') # Te estoy saludando.
```

Es un buen hábito escribir comentarios y te conviene acostumbrarte a hacerlo.

Números

¿Y si en lugar de texto quiero mostrar una operación?

Es muy fácil, solo debes tener en cuenta una regla: cuando vas a mostrar texto, el interior de los paréntesis de *print* va con comillas simples:

print('texto')

Pero cuando quieres mostrar números u operaciones, va sin comillas:

print(1 + 2)

Se aprecia la diferencia, ¿verdad?. Ahora vamos a poner un ejemplo:

Ejemplo en lenguaje humano:

Orden: Dime, ¿cuánto son 10 menos 2 y medio?
Respuesta: Son 7 y medio.

Lenguaje Python:

Orden: print(10 – 2.5)
Respuesta: 7.5

Aquí solo hemos mostrado números pero, ¿podríamos mostrar texto y números al mismo tiempo en nuestro programa? La respuesta es que sí.

Orden: print('10 por 3 son', 10 * 3)
Respuesta: 10 por 3 son 30

Para combinar texto con operaciones debes respetar la regla que mencioné antes: el texto lleva comillas, los números no. La frase *'10 por 3 son'* aunque contiene números, a estos se los considera texto y por eso llevan comillas.
*10 * 3* es una operación; no es texto y por eso no lleva comillas.

Importante: el texto va separado de los números y operaciones por coma.

Orden: print('Mi Windows tiene', 60 + 40, 'virus.')
Respuesta: Mi Windows tiene 100 virus.

Pongo una coma a cada lado de la operación para separarlo de los dos textos.

Seguro que te habrás desconcertado un poco al ver que la multiplicación en Python se hace con asterisco (*) y no con equis (x). También te habrá sorprendido ver que en los números con parte fraccionaria no hay separación por coma, sino por punto; de tal manera que el número *6,30* en Python es *6.30*.

Te conviene conocer los tipos de número, así como las operaciones aritméticas que puedes hacer en este lenguaje (suma, resta, multiplicación…) y otras más potentes.

Tipos de número y operaciones:

En Python, dentro del sistema númerico decimal, podemos operar con números enteros y números reales. Los números enteros son aquellos que no tienen parte fraccionaria; y los números reales son los que sí la tienen. Ejemplo:

```
Número entero:   100
Número real:     20.30
```

Hay algo importante a tener en cuenta, y es que el resultado de una operación será entero o real dependiendo del tipo de números con los que se opere.
De esta manera, si todos los números son enteros el resultado también lo será; pero si al menos uno de los números fuera real, el resultado será real. Ejemplo:

```
20 + 30 = 50        # El resultado es entero, porque todos los números lo son.
5 * 2 = 10          # El resultado también es entero.

20.0 + 30 = 50.0    # El resultado es real, porque hay un número real.
5 * 2.5 = 12.5      # El resultado también es real.
2.2 – 0.2 = 2.0     # El resultado también se muestra como real.
```

He aquí una tabla que muestra algunas de las operaciones básicas que pueden hacerse en Python:

Operación	Operador	Ejemplo
Suma	+	2 + 4 = 6
Resta	–	2 – 4 = -2
Multiplicación	*	3 * 3 = 9
Potencia	**	3 ** 3 = 27
División	/	50 / 6 = 8.3
Cociente	//	50 // 6 = 8
Resto	%	50 % 6 = 2

Nota: asumo que el lector tiene conocimientos de matématicas suficientes como para entender lo que se expresa en la anterior tabla.

Variables

Ya sabemos cómo mostrar información y operar con números, pero hasta ahora nuestra información era estática y no se guardaba en ningún lado.

Una *variable* es un elemento que permite almacenar información. Para comprender su funcionamiento pondré como ejemplo una comparación.

Imagina que tienes 2 cajones y cada uno lleva una etiqueta puesta. Dentro de cada cajón tendremos guardado algo:

nombre = 'Almudena'	edad = 19

De esta manera nuestro programa tendrá control en todo momento sobre los datos que manejamos. Si Almudena cumpliera un año más, su edad se incrementaría:

nombre = 'Almudena'	edad = 20

Pongamos un ejemplo en Python:

```
nombre = 'Almudena' # Guardo el nombre de la chica.
edad = 19 # Guardo su edad.

# Muestro su nombre:
print('Te llamas', nombre)

# A continuación muestro cuantos años tiene:
print('Tienes', edad, 'años.')

edad = edad + 1 # Le pongo un año más.

# Vuelvo a mostrar su edad:
print('Cumples', edad, 'años ¡felicidades!')

input() # Hago una pausa hasta que el usuario pulse la tecla INTRO.
```

El programa mostraría lo siguiente:

```
Te llamas Almudena
Tienes 19 años.
Cumples 20 años ¡felicidades!
```

Te habrás percatado de que la función *print* también sirve para mostrar variables. Las variables también deben respetar la norma que vimos antes: van sin comillas y deben estar separadas del texto por coma.

La variable debe su nombre a que la información que guarda puede variar.
Para manejar la información podemos usar los operadores que vimos en la anterior tabla; y además, operar con varias variables a la vez. Ejemplo:

```
comida_niños = 50 # Tengo 50 bandejas de comida para niños.
comida_adultos = 30 # Tengo 30 bandejas de comida para adultos.

# Muestro las bandejas que tengo:
print('Tienes', comida_niños, 'bandejas de comida para niños.')
print('Tienes', comida_adultos, 'bandejas de comida para adultos.')

# Simulo que los clientes han hecho consumiciones:
comida_niños = comida_niños – 40
comida_adultos = comida_adultos – 20

# Sumo las bandejas que tengo y las guardo en otra variable:
total_comida = comida_niños + comida_adultos

print('Ha habido venta. En total quedan', total_comida, 'bandejas de comida.')

input() # Hago una pausa hasta que el usuario pulse la tecla INTRO.
```

El programa mostraría lo siguiente:

```
Tienes 50 bandejas de comida para niños.
Tienes 30 bandejas de comida para adultos.
Ha habido venta. En total quedan 20 bandejas de comida.
```

Importante: en el caso de *"comida_niños = comida_niños – 40"* estamos haciendo un decremento en el valor de la variable, pero de esta manera queda demasiado largo. Podría hacerse lo mismo de una manera más corta, pero sólo es válido cuando interviene una variable y un número; o dos variables. Ejemplo:

```
comida_niños = comida_niños – 40 # Método largo.
comida_niños –= 40 # Método corto. Hace lo mismo que el largo.
```

Estaríamos haciendo lo mismo, pero ahorrando espacio y tiempo.
Aquí otro ejemplo, esta vez con dos variables:

```
total = 10
sobrante = 20
total += sobrante # Le sumo el contenido de "sobrante" a la variable "total".

print('El total mas el sobrante es', total)

input() # Hago una pausa hasta que el usuario pulse la tecla INTRO.
```

El programa mostraría lo siguiente: *El total mas el sobrante es 30*

Ahora te enseñaré lo que **nunca** debes hacer:

```
cds_reggaeton = 2
cds_metal = 7
total = cds_reggaeton += cds_metal # Sólo pueden intervenir dos variables.

# Esto daría un error sintáctico. No se puede mezclar el metal con reggaeton...
```

He aquí una tabla en la que podrás ver cada operación con su correspondiente ejemplo simplificado:

Operación	Ejemplo	Ejemplo simplificado
Asignación	a = b	a = b
Suma	a = a + b	a += b
Resta	a = a – b	a -= b
Multiplicación	a = a * b	a *= b
Potencia	a = a ** b	a **= b
División	a = a / b	a /= b
Cociente	a = a // b	a //= b
Resto	a = a % b	a %= b

Entrada de información

Otra acción imprescindible que realiza un programa es la de tomar información. Esto le permite interactuar con el usuario, adaptándose a las necesidades de éste. Si un programa no toma información los resultados siempre serán los mismos, por lo que resultará de poca utilidad.

La función input()

Ya habrás visto esta función en ejemplos de código anteriores. Nosotros la usaremos para dos cosas: hacer pausas en nuestro programa y permitir que el usuario introduzca información.

Veamos un ejemplo de pausa:

input()

Aquí el programa se quedaría esperando hasta que el usuario pulse INTRO.

También podemos hacer que se muestre un mensaje junto con la función. Ejemplo:

input('Pulsa INTRO para continuar...')

El programa mostraría lo siguiente:

Pulsa INTRO para continuar...

Tras pulsar INTRO el programa continuaría con su ejecución o, en caso de ser *input()* la última línea de código, el programa finalizaría de forma normal.

Ahora que ya sabemos cómo hacer pausas con esta función, vamos a aprender a pedirle información al usuario y guardarla en variables. Para ello conoceremos algunos tipos de datos, los cuales se muestran en la siguiente tabla:

Tipo	Clase	Ejemplo
str	Texto	'Esto es un texto'
int	Número entero	23
float	Número real	3.14

La información de la tabla anterior puede parecer algo confusa al principio, pero ahora te lo explicaré detalladamente para que lo entiendas:

```
nombre = str (input('Introduce tu nombre'))
print('Te llamas', nombre)
```

Usamos *str* cuando queremos que el usuario introduzca cualquier tipo de texto como palabras, frases, etc. Cabe destacar que también podemos operar con texto (sumarlo, restarlo, multiplicarlo…), por increíble que te pueda parecer. Ejemplo:

```
# Este programa se inventa palabras compuestas:
palabra_1 = str (input('Introduce la primera palabra'))
palabra_2 = str (input('Introduce la segunda palabra'))

compuesta = palabra_1 + palabra_2 # Hago la unión de ambos textos.

print('Me he inventado esta palabra:', compuesta)
```

El tipo *int* se utiliza cuando queremos que el usuario introduzca números enteros. Este tipo tambíen es válido para números muy grandes. Ejemplo:

```
edad = int (input('Introduce tu edad'))
print('Tienes', edad, 'años.')
```

Otro ejemplo:

```
dist_pluto = int (input('¿A cuántos kilómetros está Plutón de la Tierra?'))
print('Está a', dist_pluto, 'kilómetros.')
```

Usaremos el tipo *float* para que el usuario introduzca números reales. Ejemplo:

```
pi = float (input('¿Cuál es el número PI?')) # Ya sabemos que es 3.14
print('El número PI es', pi)
```

Y con esto tenemos lo necesario para crear una interacción básica con el usuario.

Ejercicios

Con lo que se ha visto hasta ahora has acumulado una cantidad considerable de información y ya va siendo hora de ir asimilándola. ¿Qué mejor que unos cuantos ejercicios para ponerte a prueba?

Tranquilo, son ejercicios resueltos. Si te quedas bloqueado podrás ver la solución.

Nota: se entiende que todos los ejercicios deberán estar comentados y hacer una pausa hasta que el usuario pulse INTRO, antes de finalizar.

Haz un programa que…

1. Muestre el texto “Hola mundo”.

2. Pida al usuario su nombre y edad, y los muestre.

3. Pida al usuario dos números enteros, los sume y muestre el resultado.

4. Pida al usuario un número real y calcule su raíz cuadrada.
Para esto puedes usar la propiedad de las potencias ($\sqrt{n} = n^{0.5}$).

5. Calcule el área de un círculo. Será el usuario quien introduzca el radio.
La fórmula es *PI* multiplicado por *radio* al cuadrado ($3.14 \cdot r^2$).

1. Solución:

```
print('Hola mundo.') # Muestro el texto.
input('Pulse INTRO para finalizar...') # Hago una pausa.
```

2. Solución:

```
nombre = str (input('Introduzca su nombre')) # Le pido su nombre.
edad = int (input('Introduzca su edad')) # Le pido su edad.

print('Te llamas', nombre, 'y tienes', edad, 'años.') # Muestro nombre y edad.

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

3. Solución:

```
num_1 = int (input('Introduzca el primer número')) # Pido un número.
num_2 = int (input('Introduzca el segundo número')) # Pido otro número.
Resultado = num_1 + num_2 # Hago la operación y almaceno el resultado.

print('El resultado es', resultado) # Muestro el resultado.

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

4. Solución:

```
radicando = float (input('Introduzca el radicando')) # Pido un número.
resultado = radicando ** 0.5 # Calculo la raíz y almaceno el resultado.

print('La raíz es', resultado) # Muestro el resultado.

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

5. Solución:

```
pi = 3.14 # PI son 3.14

radio = float (input('Introduzca el radio en cm')) # Pido al usuario el radio.
resultado = pi * radio ** 2 # Hago la operación y guardo el resultado.

print('El área es', resultado, 'cm') # Muestro el resultado.

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

Condiciones

Las condiciones dotan de inteligencia a nuestros programas. Usando una condición puedes plantear una igualdad que, si se cumple, conlleva a una determinada acción.

Para comprobar una igualdad usamos los siguientes operadores relacionales:

Operación	Operador	Ejemplo	Descripción
Igual	==	a == b	Comprueba si *a* y *b* son iguales.
Desigual	!=	a != b	Comprueba si *a* y *b* sin distintos.
Mayor que	>	a > b	Comprueba si *a* es mayor que *b*.
Menor que	<	a < b	Comprueba si *a* es menor que *b*.
Mayor o igual a	>=	a >= b	Comprueba si *a* es mayor o igual a *b*.
Menor o igual a	<=	a <= b	Comprueba si *a* es menor o igual a *b*.

A continuación se explicará cómo usar condiciones.

if/else

Bien, *if* y *else* son dos funciones que pueden funcionar en conjunto o de manera independiente para crear condiciones. Pongamos un ejemplo con *if*:

```
nombre = str (input('¿Cómo te llamas?')) # Pido al usuario su nombre.

# Establezco una condición: si el usuario se llama Cristian, le saludo.
if nombre == 'Cristian':
    print('Hola', nombre)
1234

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

He establecido que, si el usuario se llama Cristian, el programa le saludará. En caso de no cumplirse esto, el programa terminaría sin mostrar nada.

Ahora atento, pues lo que voy a decir es muy importante: para que una serie de acciones estén sujetas a una condición deberás indentarlas con 4 espacios, **nunca** con tabulador. Todo lo que no esté sujeto a una condición va sin indentado.

Como ya dije antes, puedes usar *if* de forma independiente. En cambio, para usar *else* **siempre** debe haber antes de éste un *if*.

El efecto de *else* es que si no se cumple la condición de *if*, entonces el bloque de acciones de *else* se ejecuta. Pongamos otro ejemplo para verlo con más claridad:

```
# Le pido al usuario que introduzca su nombre.
nombre = str (input('¿Cómo te llamas?'))

# Compruebo si el usuario se llama Laura.
# Siempre que se cumpla el if, el else nunca se ejecutará.
if nombre == 'Laura':
    print('Hola', nombre)
```

1234

```
# Si no se cumple la condición de if, entonces sí se ejecuta el else.
else:
    print('No te conozco.')
```

1234

```
# El input está afuera de ambas condiciones, así que siempre se ejecutará.
# Para dejar afuera un bloque, éste debe estar sin indentar.
input('Pulse INTRO para finalizar...') # Hago una pausa.
```

Cabe destacar que después de cada llamada a *if* y *else* van dos puntos (:).

Ahora supongamos que queremos comprobar varias condiciones, ¿podríamos poner varios *if* uno detrás de otro? la respuesta es que sí. Ejemplo:

```
# Este programa comprueba si cumplimos los requisitos de Ubuntu 12.04.
# Pido información sobre el procesador:
procesador = float (input('¿Cuántos megaherzios tiene su procesador?'))
# Pido información sobre la RAM:
ram = int (input('¿Cuánta memoria RAM tiene su ordenador?'))

if procesador >= 1000: # Compruebo la potencia del procesador.
    print('Tu procesador cumple los requisitos.')
```

1234

```
else:
    print('Tu procesador no cumple los requisitos.')
```

1234

```
if ram >= 1024: # Compruebo la cantidad de RAM.
    print('Tienes suficiente memoria RAM.')
```

1234

```
else:
    print('No tienes suficiente memoria RAM.')
```

1234

```
input('Pulse INTRO para finalizar...') # Hago una pausa.
```

elif

Hay otro método para hacer varias comprobaciones, aunque el efecto es distinto. Si queremos hacer varias comprobaciones y que varios bloques de acciones se ejecuten, entonces usaremos varios *if*; pero si queremos hacer varias comprobaciones y que sólo se ejecute un bloque de acciones, usaremos *elif*.

```
# Este programa comprueba si un número es positivo o negativo:
num = int (input('Introduzca un número')) # Pido un número al usuario.

if num < 0:
   print(num, 'es un número negativo.')
elif num > 0:
   print(num, 'es un número positivo.')
elif num == 0:
   print(num, 'no pertenece a ningún grupo.')

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

También podemos añadir un *else* después del útimo *elif* para contemplar otros posibles sucesos. Ejemplo:

```
nombre = str (input('¿Cómo se llama usted?')) # Pido su nombre al usuario.

# El programa comprobará si te llamas Alberto o Regina.
# De cumplirse te saludará, pero si tienes otro nombre dirá que no te conoce.
if nombre == 'Alberto':
   print('Hola', nombre)
elif nombre == 'Regina'
   print('Hola', nombre)
else:
   print('No te conozco.')

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

Importante: esto no es más que una cuestión de gustos. Dependiendo de tu estilo de programación (y del propósito de tu programa) te sentirás más cómodo con un método u otro, pero lo cierto es que *elif* ofrece muchas facilidades.

Para poder usar *elif*, antes debe existir un *if*. Puedes combinar *if-elif-else*, pero *else* sólo podrá intervenir una vez, después del último *elif*, como muestra el ejemplo.

Ejercicios

Ahora que has aprendido a usar condiciones es momento de ponerlo en práctica para terminar de asimilarlo. En la página siguiente encontrarás las soluciones.

Nota: se entiende que todos los ejercicios deberán estar comentados y hacer una pausa hasta que el usuario pulse INTRO, antes de finalizar.

Haz un programa que…

1. Pida dos números reales, compruebe cuál es mayor y lo muestre.

2. Pida tu edad y compruebe si tienes o no la mayoría. En caso de introducir un número menor que cero, el programa devolverá un mensaje de error.

3. Pida un número entero, la nota final de una asignatura. Si es menor que 0, devolverá un mensaje de error; si es menor que 5, mostará por pantalla "suspenso"; si es igual a 5, "suficiente"; si es igual a 6, "aprobado"; si es igual a 7, "notable"; y si es mayor o igual a 8, "sobresaliente".

1. Solución:

```
num_1 = float (input('Introduzca el primer número')) # Pido un número.
num_2 = float (input('Introduzca el segundo número')) # Pido otro número.

# Compruebo cuál de los dos es mayor, o si son iguales.
if num_1 == num_2:
   print('Son iguales.')
elif num_1 > num_2:
   print(num_1, 'es mayor que', num_2)
elif num_1 < num_2:
   print(num_2, 'es mayor que', num_1)

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

2. Solución:

```
edad = int (input('¿Cuántos años tienes?')) # Pido la edad al usuario.

# Hago la comprobación.
if edad < 0:
   print('Error.')
elif edad < 18:
   print('Eres menor de edad.')
elif edad >= 18:
   print('Eres mayor de edad.')

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

3. Solución:

```
nota = int (input('Introduzca su calificación')) # Pido la nota.

# Hago la comprobación.
If nota < 0:
   print('Error.')
elif nota < 5:
   print('Suspenso.')
elif nota == 5:
   print('Suficiente.')
elif nota == 6:
   print('Aprobado.')
elif nota == 7:
   print('Notable.')
elif nota >= 8:
   print('Sobresaliente.')

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

Bucles

El bucle es otro tipo de condición. Con él también podemos plantear una igualdad, con la diferencia de que mientras ésta se cumpla, el bloque de acciones sujeto al bucle se irá repitiendo una y otra vez. El bucle sólo finalizará cuando la igualdad deje de cumplirse.

A continuación se explicará cómo usar el bucle *while*.

while

El bucle *while* se puede utilizar cuando desconocemos el número de veces que se repetirá un bloque y para el *manejo de excepciones*, entre otras cosas.

Veamos un ejemplo de repetición de un bloque:

```
# Este programa nos servirá para comprar un número dado de cómics:
comics_usuario = 0 # Número de cómics que tiene actualmente el usuario.
# Le pregunto al usuario cuántos cómics le gustaría tener.
num_comics = int (input('¿Cuántos cómics te gustaría tener?'))

# Mientras tengas menos cómics de los que te gustaría tener...
while comics_usuario < num_comics:
    print('Tienes', comics_usuario, 'comics.')
    print('Has comprado un cómic nuevo.')
    comics_usuario += 1
    input('Pulse INTRO para continuar...') # Hago una pausa.
```

```
1234
```

```
input('Pulse INTRO para finalizar...') # Hago una pausa.
```

El *manejo de excepciones* sirve para impedir que ocurran imprevistos en la ejecución de un programa. Por ejemplo, cuando quiero que el usuario introduzca sólo números mayores que cero.

```
# Le pregunto al usuario cuántos cómics le gustaría tener.
num_comics = int (input('¿Cuántos cómics te gustaría tener?'))

while num_comics <= 0: # Manejo de excepciones.
    print('Debes introducir un número mayor que cero.') # Advertencia.
    # Hago que el usuario introduzca otro número.
    # Si sigue siendo menor o igual a cero, el bloque volverá a repetirse.
    num_comics = int (input('¿Cuántos cómics te gustaría tener?'))
```

```
1234
```

```
input('Pulse INTRO para finalizar...') # Hago una pausa.
```

Ejercicios

Ahora que sabes usar bucles es el momento de ponerlo en práctica. Éste será el último ejercicio del libro, y con él termina tu aprendizaje.

Nota: el ejercicio deberá estar comentado; tener manejo de excepciones y hacer una pausa hasta que el usuario pulse INTRO, antes de finalizar.

Haz un programa que…

1. Pida al usuario un número entero, impidiendo que éste sea menor que cero. El programa mostrará la tabla de multiplicar de dicho número.

1. Solución:

```
num = int (input('Introduzca un número')) # Pido un número al usuario.
multiplicador = 0 # Número que irá multiplicando al introducido.

while num < 0: # Manejo de excepciones.
    print('El número introducido no puede ser negativo.') # Advertencia.
    num = int (input('Introduzca otro número')) # Pido un número al usuario.

# Mientras el multiplicador sea menor o igual a 10...
while multiplicador <= 10:
    resultado = num * multiplicador # Opero y almaceno el resultado.
    print(num, 'x', multiplicador, '=', resultado) # Muestro el resultado.
    multiplicador += 1 # Incremento el multiplicador.

input('Pulse INTRO para finalizar...') # Hago una pausa.
```

La despedida

Si lees estas líneas caben dos posibilidades: o eres un curioso y te has ido directo a mirar qué hay en la última página del libro, o estás a punto de terminarlo.
De ser el segundo caso, te felicito. Has aprendido lo básico y ya estás preparado para seguir tu propio camino, pero debes saber que todavía te faltan muchísimas cosas por ver. Dominar el arte de la programación requiere años.

Éste es un camino difícil y más cuando se recorre en solitario. Por eso, ¿qué mejor que ir acompañado por otras personas que comparten tus mismos intereses?. Existe una comunidad de usuarios apasionados por la libertad y la tecnología llamada Free Development. Allí todos hacen un magnífico trabajo compartiendo sus conocimientos y llevando a cabo proyectos que apuestan por el bien común.

Cualquier persona puede entrar a formar parte de la comunidad sin importar su nivel de conocimientos. Los miembros estarán encantados de conocer a nuevos compañeros y sin duda será una gran experiencia para ti.

Puedes contactar con la comunidad Free Development desde las redes sociales:

- Diaspora
- Facebook
- Tuenti
- Twitter

También puedes escribirles un correo a free.development.7db@gmail.com

Por último, debo confesar que he disfrutado mucho invirtiendo mi tiempo en esta obra y espero que tú hayas sentido lo mismo. Sin más demora me despido.

José Miguel Ruiz Torres