Capítulo 2

8 multiplicado por 3.57 igual a...

¿Cuánto es 8 multiplicado por 3.57? Tendrías que utilizar una calculadora, ¿a que sí? Bueno, tal vez eres superlisto y puedes calcular con decimales mentalmente—bueno, no es importante. Tambien puedes hacer el cálculo con la consola de Python. Inicia la consola de nuevo (mira el Capítulo 1 para más información, si te lo hubieras saltado por alguna extraña razón), y cuando veas el prompt en pantalla, teclea 8∗3.57 y pulsa la tecla Intro:

```
Python 3.0 (r30:67503, Dec  6 2008, 23:22:48)
Type "help", "copyright", "credits" or "license" for more information.
>>> 8 * 3.57
28.559999999999999
```

La estrella (*), o tecla de asterico, se utiliza como símbolo de multiplicación, en lugar del simbolo de multiplicar tradicional (X) que utilizas en el cole (es necesario que utilices la tecla de estrella porque el ordenador no tendría forma de distinguir si lo querías era teclear la letra x o el símbolo de la multiplicación X). ¿Qué te parece si probamos un cálculo que sea un poco más útil?

Supón que haces tus obligaciones de la casa una vez a la semana, y por ello te dan 5 euros, y que haces de chico repartidor de periódicos 5 veces por semana cobrando 30 euros—¿Cuánto dinero podrías ganar en un año?

Si lo escribieramos en un papel, lo calcularíamos como sigue:

```
(5 + 30) x 52
```

¿!¿!Python está roto!?!?

Si has probado a calcular 8 x 3.57 con una calculadora el resultado que se muestra será el siguiente:

28.56

¿Porqué el resultado de Python es diferente? ¿Está roto?

En realidad no. La razón de esta diferencia está en la forma en que se hacen los cálculos en el ordenador con los números de coma flotante (Los números que tienen una coma y decimales). Es algo complejo y que confunde a los principiantes, por lo que es mejor que recuerdes únicamente que cuando estás trabajando con decimales, *a veces* el resultado no será exactamente el experado. Esto puede pasar con las multiplicaciones, divisiones, sumas y restas.

Es decir, 5 + 30 euros a la semana multiplicado por las 52 semanas del año. Desde luego, somos chicos listos por lo que sabemos que 5 + 30 es 35, por lo que la fórmula en realidad es:

```
35 x 52
```

Que es suficientemente sencilla como para hacerla con una calculadora o a mano, con lápiz y papel. Pero también podemos hacer todos los cálculos con la consola:

```
>>> (5 + 30) * 52
1820
>>> 35 * 52
1820
```

Veamos, ¿Qué pasaría si te gastases 10 euros cada semana? ¿Cuánto te quedaría a final de año? En papel podemos escribir este cálculo de varias formas, pero vamos a teclearlo en la consola de Python:

```
>>> (5 + 30 - 10) * 52
1300
```

Son 5 más 30 euros menos 10 euros multiplicado por las 52 semanas que tiene el año. Te quedarán 1300 euros a final de año. Estupendo, pero aún no es que sea muy útil lo que hemos visto. Todo se podría haber hecho con una calculadora. Pero volveremos con este asunto más tarde para mostrarte como hacer que sea mucho más útil.

En la consola de Python puedes multiplicar y sumar (obvio), y restar y dividir, además de un gran puñado de otras operaciones matemáticas que no vamos a comentar ahora. Por el momento nos quedamos con los símbolos básicos de matemáticas que se pueden usar en Python (en realidad, en este lenguaje, se les llama operadores):

+	Suma
-	Resta
*	Multiplicación
/	División

La razón por la que se usa la barra inclinada hacia adelante (/) para la división, es que sería muy difícil dibujar la línea de la división como se supone que lo haces en las fórmulas escritas (además de que ni se molestan en poner un símbolo de división ÷ en el teclado del ordenador). Por ejemplo, si tuvieras 100 huevos y 20 cajas, querrías conocer cuantos huevos deberías meter en cada caja, por lo que tendrías que dividir 100 entre 20, para lo que escribirías la siguiente fórmula:

$$\frac{100}{20}$$

O si supieras como se escribe la división larga, lo harías así:

$$\begin{array}{r|l} 100 & 20 \\ \cline{2-2} 100 & 5 \\ \cline{1-1} 0 & \\ & \\ & \end{array}$$

O incluso lo podrías escribir de esta otra forma:

$$100 \div 20$$

Sin embargo, en Python tienes que escribirlo así “100 / 20”.

Lo que es mucho más sencillo, o eso me parece a mi. Pero bueno, yo soy un libro—¿Qué voy a saber yo?

2.1. El uso de los paréntesis y el "Orden de las Operaciones"

En los lenguajes de programación se utilizan los paréntesis para controlar lo que se conoce como el "Orden de las Operaciones". Una operación es cuando se usa un operador (uno de los símbolos de la tabla anterior). Hay más operadores que esos símbolos básicos, pero para esa lista (suma, resta, multiplicación y división), es suficiente con saber que la multiplicación y la división tiene mayor orden de prioridad que la suma y la resta. Esto significa que en una fórmula, la parte de la multiplicación o la división se calcula antes que la parte de la suma y resta. En la siguiente fórmula, todos los operadores son sumas (+), en este caso los números se suman en el orden que aparecen de izquierda a derecha:

```
>>> print(5 + 30 + 20)
55
```

De igual manera, en esta otra fórmula, hay únicamente operadores de sumas y restas, por lo que de nuevo Python considera cada número en el orden en que aparece:

```
>>> print(5 + 30 - 20)
15
```

Pero en la siguiente fórmula, hay un operador de multiplicación, por lo que los números 30 y 20 se toman en primer lugar. Esta fórmula lo que está diciendo es "multiplica 30 por 20, y luego suma 5 al resultado" (se multiplica primero porque tiene mayor orden que la suma):

```
>>> print(5 + 30 * 20)
605
```

¿Pero qué es lo que sucede cuando añadimos paréntesis? La siguiente fórmula muestra el resultado:

```
>>> print((5 + 30) * 20)
700
```

¿Por qué el número es diferente? Porque los paréntesis controlan el orden de las operaciones. Con los paréntesis, Python sabe que tiene que calcular primero todos los operadores que están dentro de los paréntesis, y luego los de fuera. Por eso lo que significa esta fórmula es "suma 5 y 30 y luego multiplica el resultado por 20". El uso de los paréntesis puede volverse más complejo. Pueden existir paréntesis dentro de paréntesis:

```
>>> print(((5 + 30) * 20) / 10)
70
```

En este caso, Python calcula los paréntesis **más internos** primero, luego los exteriores, y al final el otro operador. Por eso esta fórmula significa "suma 5 y 30, luego multiplica el resultado por 20, y finalmente divide el resultado entre 10". El resultado sin los paréntesis sería ligeramente diferente:

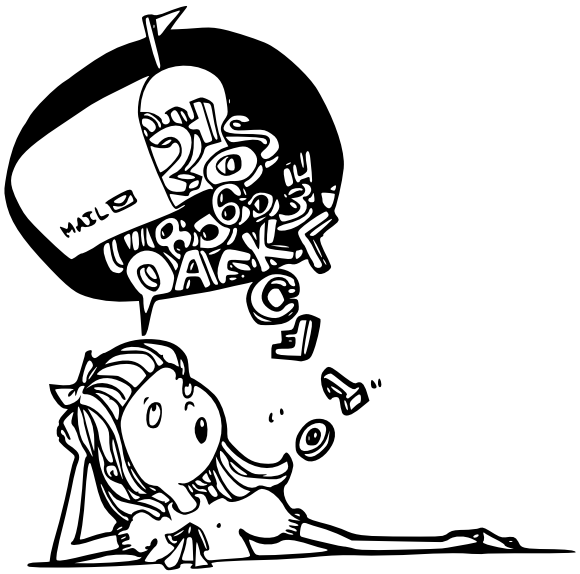
```
>>> 5 + 30 * 20 / 10
65
```

En este caso se multiplica primero 30 por 20, luego el resultado se divide entre 10, finalmente se suma 5 para obtener el resultado final.

Recuerda que la multiplicación y la división siempre van antes que la suma y la resta, a menos que se utilicen paréntesis para controlar el orden de las operaciones.

2.2. No hay nada tan voluble como una variable

Una 'variable' es un término de programación que se utiliza para describir un sitio en el que almacenar cosas. Las 'cosas' pueden ser números, o textos, o listas de números y textos—y toda clase de otros elementos demasiado numerosos para profundizar aquí. Por el momento, vamos a decir que una variable algo así como un buzón o caja.



De la misma forma en que puedes meter cosas (como una carta o un paquete) en un un buzón, puedes hacerlo con una variable, puedes meter cosas (números, textos, listas de números y textos, etc, etc, etc) en ella. Con esta idea de buzón es con la que funcionan muchos lenguajes de programación. Pero no todos.

En Python, las variables son ligeramente diferentes. En lugar de ser un buzón con cosas dentro, una variable es más parecida a una etiqueta que pegas por fuera de la caja o carta. Podemos despegar la etiqueta y pegarla en otra cosa, o incluso atar la etiqueta (con un trozo de cuerda, tal vez) a más de una cosa. En Python las variables se crean al darles un nombre, utilizando después un símbolo de igual (=), y luego diciéndole a Python a qué cosa queremos que apunte este nombre. Por ejemplo:

```
>>> fred = 100
```

Acabamos de crear una variable llamada 'fred' y le dijimos que apuntase al número 100. Lo que estamos haciendo es decirle a Python que recuerde ese número porque queremos usarlo más tarde. Para descubrir a qué está apuntando una variable, lo único que tenemos que hacer es teclear 'print' en la consola, seguido del nombre de la variable, y pulsar la tecla Intro. Por ejemplo:

```
>>> fred = 100
>>> print(fred)
100
```

También le podemos decir a Python que queremos que la variable fred apunte a otra cosa diferente:

```
>>> fred = 200
>>> print(fred)
200
```

En la primera línea decimos que ahora queremos que fred apunte al número 200. Luego, en la segunda línea, le pedimos a Python que nos diga a qué está apuntando fred para comprobar que lo hemos cambiado (ya no apunta a 100). También podemos hacer que más de un nombre apunte a la misma cosa:

```
>>> fred = 200
>>> john = fred
>>> print(john)
200
```

En el código anterior estamos diciendo que queremos que el nombre (o etiqueta) john apunte a la misma cosa a la que apunta fred. Desde luego, 'fred' no es un nombre

muy útil para una variable. No nos dice nada sobre para qué se usa. Un buzón se sabe para lo que se usa—para el correo. Pero una variable puede tener diferentes usos, y puede apuntar a muchas cosas diferentes entre sí, por lo que normalmente queremos que su nombre sea más informativo.

Supón que iniciaste la consola de Python, tecleaste 'fred = 200', y después te fuiste—pasaste 10 años escalando el Monte Everest, cruzando el desierto del Sahara, haciendo puenting desde un puente en Nueva Zelanda, y finalmente, navegando por el rio Amazonas—cuando vuelves a tu ordenador, ¿podrías acordarte de lo que el número 200 significaba (y para qué servía)?

Yo no creo que pudiera.

En realidad, acabo de usarlo y no tengo ni idea qué significa 'fred=200' (más allá de que sea un *nombre* apuntando a un número *200*). Por eso después de 10 años, no tendrás absolutamente ninguna oportunidad de acordarte.

¡Ajá! Pero si llamáramos a nuestra variable: *numero_de_estudiantes*.

```
>>> numero_de_estudiantes = 200
```

Podemos hacer esto porque los nombres de las variables pueden formarse con letras, números y guiones bajos (_)—aunque no pueden comenzar por un número. Si vuelves después de 10 años, 'numero_de_estudiantes' aún tendrá sentido para ti. Puedes teclear:

```
>>> print(numero_de_estudiantes)
200
```

E inmediatamente sabrás que estamos hablando de 200 estudiantes. No siempre es importante que los nombres de las variables tengan significado. Puedes usar cualquier cosa desde letras sueltas (como la 'a') hasta largas frases; y en ocasiones, si vas a hacer algo rápido, un nombre simple y rápido para una variable es suficientemente útil. Depende mucho de si vas a querer usar ese nombre de la variable más tarde y recordar por el nombre en qué estabas pensando cuando la tecleaste.

```
este_tambien_es_un_nombre_valido_de_variable_pero_no_muy_util
```

2.3. Utilizando variables

Ya conocemos como crear una variable, ahora ¿Cómo la usamos? ¿Recuerdas la fórmula que preparamos antes? Aquella que nos servía para conocer cuánto dinero

tendríamos al final de año, si ganabas 5 euros a la semana haciendo tareas, 30 euros a la semana repartiendo periódicos, y gastabas 10 euros por semana. Por ahora tenemos:

```
>>> print((5 + 30 - 10) * 52)
1300
```

¿Qué pasaría si convertimos los tres primeros números en variables? Intenta teclear lo siguiente:

```
>>> tareas = 5
>>> repartir_periodicos = 30
>>> gastos = 10
```

Acabamos de crear unas variables llamadas 'tareas', 'repartir_periodicos' y 'gastos'. Ahora podemos volver a teclear la fórmula de la siguiente forma:

```
>>> print((tareas + repartir_periodicos - gastos) * 52)
1300
```

Lo que nos da exactamente la misma respuesta. Pero qué sucedería si eres capaz de conseguir 2 euros más por semana, por hacer tareas extra. Solamente tienes que cambiar la variable 'tareas' a 7, luego pulsar la tecla de flecha para arriba (↑) en el teclado varias veces, hasta que vuelva a aparecer la fórmula en el prompt, y pulsar la tecla Intro:

```
>>> tareas = 7
>>> print((tareas + repartir_periodicos - gastos) * 52)
1404
```

Así hay que teclear mucho menos para descubrir que a final de año vas a tener 1404 euros. Puedes probar a cambiar las otras variables, y luego pulsar la tecla de flecha hacia arriba para que se vuelva a ejecutar el cálculo y ver qué resultado se obtiene.

```
Si gastases el doble por semana:
>>> gastos = 20
>>> print((tareas + repartir_periodicos - gastos) * 52)
884
```

Solamente te quedarán 884 euros al final de año. Por ahora esto es un poco más util, pero no mucho. No hemos llegado a nada realmente útil aún. Pero por el momento es suficiente para comprender que las variables sirven para almacenar cosas.

¡Piensa en un buzón con una etiqueta en él!

2.4. ¿Un trozo de texto?

Si has estado prestando atención, y no únicamente leyendo por encima el texto, recordarás que he mencionado que las variables se pueden utilizar para varias cosas—no únicamente para los números. En programación, la mayor parte del tiempo llamamos a los textos 'cadenas de caracteres'. Esto puede parecer un poco extraño; pero si piensas que un texto es como un 'encadenamiento de letras' (poner juntas las letras), entonces quizá tenga un poco más de sentido para ti.

Si bueno, quizás no tenga tanto sentido.

En cualquier caso, lo que tienes que saber es que una cadena es solamente un puñado de letras y números y otros símbolos que se juntan de forma que signifique algo. Todas las letras, números y símbolos de este libro podrían considerarse que forman una cadena. Tu nombre podría considerarse una cadena. Como podría serlo la dirección de tu casa. El primer programa Python que creamos en el Capítulo 1, utilizaba una cadena: 'Hola mundo'.

En Python, creamos una cadena poniendo comillas alrededor del texto. Ahora podemos tomar nuestra, hasta ahora inútil, variable fred, y asignarle una cadena de la siguiente forma:

```
>>> fred = "esto es una cadena"
```

Ahora podemos mirar lo que contiene la variable fred, tecleando print(fred):

```
>>> print(fred)
esto es una cadena
```

También podemos utilizar comillas simples para crear una cadena:

```
>>> fred = 'esto es otra cadena'
>>> print(fred)
esto es otra cadena
```

Sin embargo, si intentas teclear más de una línea de texto en tu cadena utilizando comillas simple (') o comillas dobles ("), verás un mensaje de error en la consola. Por ejemplo, teclea la siguiente línea y pulsa Intro, y saldrá en pantalla un mensaje de error similar a esto:

```
>>> fred = "esta cadena tiene dos
  File "<stdin>", line 1
    fred = "esta cadena tiene dos
                         ^
SyntaxError: EOL while scanning string literal
```

Hablaremos de los errores más tarde, pero por el momento, si quieres escribir más de una línea de texto, recuerda que tienes que usar 3 comillas simples:

```
>>> fred = '''esta cadena tiene dos
... líneas de texto'''
```

Imprime el contenido de la variable para ver si ha funcionado:

```
>>> print(fred)
esta cadena tiene dos
líneas de texto
```

Por cierto, verás que salen 3 puntos (...) siempre que tecleas algo que continua en otra línea (como sucede en una cadena que ocupa más de una línea). De hecho, lo verás más veces según avanzemos en el libro.

2.5. Trucos para las cadenas

He aquí una interesante pregunta: ¿Cuánto es 10 * 5 (10 veces 5)? La respuesta es, por supuesto, 50.

De acuerdo, para nada es una pregunta interesante.

Pero ¿Cuánto es 10 * 'a' (10 veces la letra a)? Podría parecer una pregunta sin sentido, pero hay una respuesta para ella en el Mundo de Python:

```
>>> print(10 * 'a')
aaaaaaaaaa
```

También funciona con cadenas de más de un carácter:

```
>>> print(20 * 'abcd')
abcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcd
```

Otro truco con cadenas consiste en incrustar valores. Para ello puedes usar %s, que funciona como marcador o espacio reservado para el valor que quieras incluir en la cadena. Es más fácil de explicar con un ejemplo:

```
>>> mitexto = 'Tengo %s años'
>>> print(mitexto % 12)
Tengo 12 años
```

En la primera línea, creamos la variable mitexto con una cadena que contiene algunas palabras y un marcador (%s). El %s es una especie de señal que le dice

"sustitúyeme con algo" a la consola de Python. Así que en la siguiente línea, cuando ejecutamos print(mitexto), usamos el símbolo %, para decirle a Python que reemplace el marcador con el número 12. Podemos reutilizar esa cadena y pasarle diferentes valores:

```
>>> mitexto = 'Hola %s, ¿Cómo estás hoy?'
>>> nombre1 = 'Joe'
>>> nombre2 = 'Jane'
>>> print(mitexto % nombre1)
Hola Joe, ¿Cómo estás hoy?
>>> print(mitexto % nombre2)
Hola Jane, ¿Cómo estás hoy?
```

En el ejemplo anterior, hemos creado 3 variables (mitexto, nombre1 y nombre2)—la primera almacena la cadena con el marcador. Por ello, podemos imprimir la variable 'mitexto', y utilizando el operador % pasarle el valor almacenado en las variables 'nombre1' y 'nombre2'. Puedes usar más de un marcador:

```
>>> mitexto = 'Hola %s y %s, ¿Cómo estáis hoy?'
>>> print(mitexto % (nombre1, nombre2))
Hola Joe y Jane, ¿Cómo estáis hoy?
```

Cuando utilizas más de un marcador, necesitas que los valores que se usan para reemplazar las marcas se encuentren entre paréntesis—Así que el modo correcto de pasar 2 variables es (nombre1, nombre2). En Python a un conjunto de valores rodeados de paréntesis se le llama *tupla*, y es algo parecido a una lista, de las que hablaremos a continuación.

2.6. No es la lista de la compra

Huevos, leche, queso, apio, manteca de cahauetes, y levadura. Esto no es una lista de la compra completa, pero nos sirve para nuestros propósitos. Si quisieras almacenar esto en una variable podrías pensar en crear una cadena:

```
>>> lista_de_la_compra = 'huevos, leche, queso, apio, manteca de cacahuetes, levadura'
>>> print(lista_de_la_compra)
huevos, leche, queso, apio, manteca de cacahuetes, levadura
```

Otro modo de hacerlo sería crear una 'lista', que es una clase especial de objetos de Python:

```
>>> lista_de_la_compra = [ 'huevos', 'leche', 'queso', 'apio', 'manteca de cacahuetes',
...'levadura' ]
>>> print(lista_de_la_compra)
['huevos', 'leche', 'queso', 'apio', 'manteca de cacahuetes', 'levadura']
```

Aunque supone teclear más, también es más útil. Podríamos imprimir el tercer elemento en la lista utilizando su posición (al número que refleja la posición de un elemento se le denomina índice), dentro de unos corchetes []:

```
>>> print(lista_de_la_compra[2])
queso
```

Las listas comienzan a contarse en la posición con número de índice 0—es decir, el primer elemento de una lista es el 0, el segundo es el 1, el tercero es el 2. Esto no tiene mucho sentido para la mayoría de la gente, pero lo tiene para los programadores. Muy pronto te acostumbrarás y cuando estés subiendo unas escaleras, comenzarás a contar los escalones desde cero en lugar que comenzando desde uno. Lo que despistará a tu hermano o hermana pequeña.

También podemos seleccionar todos los elementos que van desde el tercero al quinto de la lista, para ello utilizamos el símbolo de los dos puntos dentro de los corchetes:

```
>>> print(lista_de_la_compra[2:5])
['queso', 'apio', 'manteca de cacahuetes']
```

[2:5] es lo mismo que decir que estamos interesados en los elementos que van desde la posición de índice 2 hasta (pero sin incluirla) la posición de índice 5. Y desde luego, como comenzamos a contar con el cero, el tercer elemento en la lista es el número 2, y el quinto es el número 4. Las listas se pueden utilizar para almacenar toda clase de objetos. Pueden almacenar números:

```
>>> milista = [ 1, 2, 5, 10, 20 ]
```

Y cadenas:

```
>>> milista = [ 'a', 'bbb', 'ccccccc', 'ddddddddd' ]
```

Y mezcla de números y cadenas:

```
>>> milista = [1, 2, 'a', 'bbb']
>>> print(milista)
[1, 2, 'a', 'bbb']
```

E incluso listas de listas:

```
>>> lista1 = [ 'a', 'b', 'c' ]
>>> lista2 = [ 1, 2, 3 ]
>>> milista = [ lista1, lista2 ]
>>> print(milista)
[['a', 'b', 'c'], [1, 2, 3]]
```

En el ejemplo anterior se crea la variable 'lista1' para contener 3 letras, la lista 'lista2' se crea con 3 números, y 'milista' se crea utilizando a las variables lista1 y lista2. Como ves, las cosas se complican rápidamente cuando empiezas a crear listas de listas de listas de listas... pero afortunadamente normalmente no hay ninguna necesidad de complicarlo tanto en Python. De todos modos sigue siendo útil conocer que puedes almacenar toda clase de elementos en una lista de Python.

Y no únicamente para hacer la compra.

Sustituyendo elementos

Podemos reemplazar un elemento en la lista asignándole un valor de la misma forma en que se hace con una variable corriente. Por ejemplo, podríamos cambiar el apio por lechuga asignando el valor nuevo a la posición de índice 3:

```
>>> lista_de_la_compra[3] = 'lechuga'
>>> print(lista_de_la_compra)
['huevos', 'leche', 'queso', 'lechuga', 'manteca de cacahuetes', 'levadura']
```

Añadiendo más elementos...

Podemos añadir elementos a una lista utilizando el método denominado 'append' (que en inglés significa 'añadir'). Un método es una acción u orden que le dice a Python que queremos que haga algo. Más tarde hablaremos más de ellos, por el momento, recuerda que para añadir un elemento a nuestra lista de la compra, podemos hacer lo siguiente:

```
>>> lista_de_la_compra.append('chocolate')
>>> print(lista_de_la_compra)
['huevos', 'leche', 'queso', 'lechuga', 'manteca de cacahuetes', 'levadura', 'chocolate']
```

Al menos ahora sí que hemos mejorado la lista de la compra.

... y suprimiendo elementos

Podemos quitar elementos de una lista mediante el uso de la orden 'del' (que es una forma corta de escribir delete, que en inglés significa 'borrar'). Por ejemplo, para eliminar el sexto elemento de la lista (levadura) podemos escribir:

```
>>> del lista_de_la_compra[5]
>>> print(lista_de_la_compra)
['huevos', 'leche', 'queso', 'lechuga', 'manteca de cacahuetes', 'chocolate']
```

Recuerda que las posiciones comienzan a contar desde cero, por lo que lista_de_la_compra[5] se refiere en realidad al sexto elemento.

2 listas mejor que 1

Podemos unir listas mediante el operador de suma, como si estuvieramos sumando dos números:

```
>>> lista1 = [ 1, 2, 3 ]
>>> lista2 = [ 4, 5, 6 ]
>>> print(lista1 + lista2)
[1, 2, 3, 4, 5, 6]
```

También podemos juntar las dos listas y asignar la lista resultante a otra variable:

```
>>> lista1 = [ 1, 2, 3 ]
>>> lista2 = [ 4, 5, 6 ]
>>> lista3 = lista1 + lista2
>>> print(lista3)
[1, 2, 3, 4, 5, 6]
```

Y puedes multiplicar una lista de la misma forma que multiplicamos una cadena (recuerda que la multiplicación de una cadena por un número lo que hace es crear una cadena que repite tantas veces la original como el número por el que multiplicas):

```
>>> lista1 = [ 1, 2 ]
>>> print(lista1 * 5)
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

En el ejemplo anterior multiplicar lista1 por cinco lo que significa es "repite la lista1 cinco veces". Sin embargo, no tiene sentido utilizar la división (/) ni la resta (-) cuando estamos trabajando con listas, por eso Python mostrará errores cuando intentas ejecutar los siguientes ejemplos:

```
>>> lista1 / 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

o:

```
>>> lista1 - 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'type' and 'int'
```

El resultado será un mensaje de error bastante feo.

2.7. Tuplas y Listas

Una tupla (mencionada anteriormente) es una cosa parecida a una lista, la diferencia es que en lugar de utilizar corchetes, utilizas paréntesis—por ejemplo '(' y ')'. Las tuplas funcionan de forma similar a las listas:

```
>>> t = (1, 2, 3)
>>> print(t[1])
2
```

La diferencia principal es que, a diferencia de las listas, las tuplas no se pueden modificar después de haberlas creado. Por eso si intentas reemplazar un valor como hicimos antes en la lista, lo que hace Python es dar un mensaje de error.

```
>>> t[0] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'tuple' object does not support item assignment
```

Eso no significa que no puedas cambiar el contenido de la variable para que en lugar de contener a la tupla, contenga otra cosa. Este código funcionará sin problemas:

```
>>> mivar = (1, 2, 3)
>>> mivar = [ 'una', 'lista', 'de', 'cadenas' ]
```

En primer lugar hemos creado la variable mivar que apunta a una tupla compuesta por 3 números. Luego modificamos mivar para que apunte a una lista de

cadenas. Al principio te puede resultar algo confuso. Pero trata de pensar en las taquillas del gimnasio o del colegio, cada taquilla está etiquetada con un nombre. Pones cualquier cosa en la taquilla, cierras la puerta con llave y después la tiras. Entonces le arrancas la etiqueta y te vas a otra taquilla vacía, le pones la etiqueta que te llevaste y metes algo dentro (pero esta vez te quedas con la llave). Una tupla es como una taquilla cerrada. No puedes modificar lo que hay dentro. Pero puedes llevarte la etiqueta (la variable) y y pegarla en otra taquilla abierta, y entonces puedes poner cosas dentro de ella y sacar cosas de ella—así son las listas.

2.8. Cosas que puedes probar

En este capítulo hemos visto cómo hacer cálculos con fórmulas matemáticas simples utilizando la consola de Python. También hemos visto cómo los paréntesis pueden modificar el resultado de una fórmula, al controlar el orden en que se calculan las operaciones. Hemos aprendido cómo decirle a Python que recuerde valores para usarlos más tarde—utilizando variables—además de aprender a usar las 'cadenas' de Python para almacenar textos, y cómo usar las listas y las tuplas para almacenar más de un elemento.

Ejercicio 1

Crea una lista de tus juguetes favoritos y llámala juguetes. Crea una lista de tus comidas favoritas y llámala comidas. Une las dos listas y llama al resultado favoritos. Finalmente imprime la variable favoritos.

Ejercicio 2

Si tienes 3 cajas que contienen 25 chocolates cada una, y 10 bolsas que contienen 32 caramelos cada una, ¿cuántos dulces y chocolates tienes en total? (Nota: puedes calcularlo con una fórmula en la consola de Python)

Ejercicio 3

Crea variables para almacenar tu nombre y tus apellidos. Después crea una cadena con un texto que quieras e incluye marcadores para añadir tu nombre y apellidos. Imprime el resultado para verlo en la consola de Python.