Desconocido

## B Funciones incorporadas de Python



Python tiene una caja bien surtida de herramientas de programación, incluyendo un gran número de funciones y módulos que están listos para que los utilices. Estas herramientas incorporadas pueden hacer que escribir programas sea mucho más fácil.

Como leíste en el [Capítulo 7](ch07.xhtml#ch07) , los módulos deben importarse antes de poder utilizarse. Las funciones incorporadas *de* Python no necesitan importarse primero; están disponibles en cuanto se inicia el Shell de Python. En este apéndice, veremos algunas de las  funciones incorporadas más útiles, y luego nos centraremos en una: la función open , que te permite abrir archivos para leer y escribir en ellos.

### Uso de las funciones incorporadas

Veamos algunas de las funciones incorporadas que utilizan habitualmente los programadores de Python. Describiré lo que hacen y cómo utilizarlas, y luego mostraré ejemplos de cómo pueden ayudarte en tus programas.

### La función abs

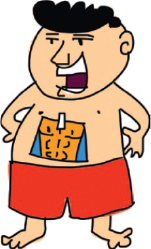
La función abs devuelve el *valor absoluto* de un número, que es el valor de un número sin su signo. Por ejemplo, el valor absoluto de 10 es 10, y el valor absoluto de -10 es 10.

Para utilizar la función abs , basta con llamarla con un número o variable como parámetro, así

>>> print(abs(10))  
10  
>>> print(abs(-10))  
10

Puedes utilizar la función abs para calcular la cantidad absoluta de movimiento de un personaje en un juego, sin importar la dirección en la que se desplace. Por ejemplo, digamos que el personaje da 3 pasos hacia su izquierda (negativo 3 o -3) y luego 10 pasos hacia su derecha (positivo 10).

Si no nos importara la dirección (positiva o negativa), el valor absoluto de estos números sería 3 y 10. Podrías utilizar esto en un juego de mesa en el que lanzas dos dados y luego mueves a tu personaje un número máximo de pasos en cualquier dirección, en función del total de los dados. Ahora, si almacenamos el número de pasos en una variable, podemos determinar si el personaje se está moviendo con el siguiente código. Puede que queramos mostrar alguna información cuando el jugador haya decidido moverse (en este caso, mostraremos "El personaje se está moviendo lejos" o "El personaje se está moviendo", según el número):



>>> steps = -3  
>>> if abs(steps) > 5:  
 print('Character is moving far')  
 elif abs(steps) != 0:  
 print('Character is moving')

Si no hubiéramos utilizado abs , la sentencia if podría tener este aspecto:

>>> steps = 10  
>>> if steps < -5 or steps > 5:  
 print('Character is moving far')  
 elif steps != 0:  
 print('Character is moving')

Como puedes ver, el uso de abs hizo que la sentencia if fuera un poco más corta y fácil de entender.

### La función all

La función all devuelve True si todos los elementos de una lista (o cualquier otro tipo de colección) se evalúan como True . En pocas palabras, esto significa que todos los valores de los elementos de la lista no son 0, None , una cadena vacía ( ’’’’ ) o el valor booleano False .

Por tanto, si todos los elementos de la lista son números distintos de cero, all devolverá True :

>>> mylist = [1,2,5,6]  
>>> all(mylist)  
True

Pero si alguno de los valores es 0 , devolverá False :

>>> mylist = [1, 2, 3, 0]  
>>> all(mylist)  
False

No sólo números: una lista mixta de valores que incluya None también devolverá False :

>>> mylist = [100, 'a', None, 'b', True, 'zzz', ' ']  
>>> all(mylist)  
False

Intentemos de nuevo el mismo ejemplo si se elimina None :

>>> mylist = [100, 'a', 'b', True, 'zzz', ' ']  
>>> all(mylist)  
True

### La función any

La función any es similar a all , salvo que si alguno de los valores se evalúa como True , devolverá True . Intentemos el mismo ejemplo con los números:

>>> mylist = [1, 2, 5, 6]  
>>> any(mylist)  
True

Nuestra lista mixta de ceros, None , cadenas vacías y False también funciona igual que all :

>>> mylist = [0, False, None, "", 0, False, '']  
>>> any(mylist)  
False

Pero si hacemos un pequeño cambio en esa lista -como añadir un número distinto de cero, como 100- obtendremos True :

>>> mylist = [0, False, None, "", 0, False, '', 100]  
>>> any(mylist)  
True

### La función bin

La función bin convierte un número en su *representación binaria .* El binario está fuera del alcance de este libro, pero, en resumen, es un sistema de numeración formado por 1s y 0s y es la base de casi todo en informática. He aquí un ejemplo sencillo, convirtiendo algunos números en binario:

>>> bin(100)  
'0b1100100'  
>>> bin(5)  
'0b101'

### La función bool

El nombre bool es la abreviatura de *Boolean* , la palabra que utilizan los programadores para describir un tipo de dato que puede tener uno de dos valores posibles: normalmente, True o False .

La función bool toma un único parámetro y devuelve True o False en función de su valor. Cuando se utiliza bool para números, 0 devuelve False pero cualquier otro número devuelve True . Así es como puedes utilizar bool con varios números:

>>> print(bool(0))  
False  
>>> print(bool(1))  
True  
>>> print(bool(1123.23))  
True  
>>> print(bool(-500))  
True

Cuando utilizas bool para otros valores, como cadenas, devuelve False si no hay valor para la cadena (en otras palabras, la palabra clave None o una cadena vacía). En caso contrario, devolverá True , como se muestra aquí:

>>> print(bool(None))  
False  
>>> print(bool('a'))  
True  
>>> print(bool(' '))  
True  
>>> print(bool('What do you call a pig doing karate? Pork Chop!'))  
True

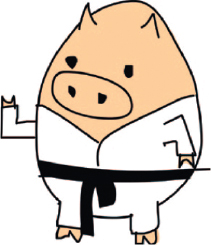
La función bool también devolverá False para listas, tuplas y mapas que no contengan ningún valor, o True cuando sí lo contengan:

>>> my\_silly\_list = []  
>>> print(bool(my\_silly\_list))  
False  
>>> my\_silly\_list = ['s', 'i', 'l', 'l', 'y']  
>>> print(bool(my\_silly\_list))  
True

Puedes utilizar bool cuando necesites decidir si un valor se ha establecido o no. Por ejemplo, si pedimos a las personas que utilizan nuestro programa que introduzcan el año en que nacieron, nuestra sentencia if podría utilizar bool para comprobar el valor que introducen:

>>> year = input('Year of birth: ')  
Year of birth:  
>>> if not bool(year):  
 print('You need to enter a value for your year of birth')  
  
You need to enter a value for your year of birth

La primera línea de este ejemplo utiliza input para almacenar lo que alguien introduce en el teclado como la variable año. Si se pulsa INTRO en la línea siguiente (sin escribir nada más) se obtiene una cadena vacía en la variable. (También utilizamos sys.stdin.readline() en [el Capítulo 7](ch07.xhtml#ch07) , que es otra forma de hacer lo mismo).



En la línea siguiente, la sentencia if comprueba el valor booleano de la variable. Como el usuario no ha introducido nada en este ejemplo, la función bool devuelve False . La sentencia if utiliza la palabra clave not , que es una forma de decir: "Haz esto si la función no devuelve True ," por lo que el código imprime You need to enter a value for your year of birth en la línea siguiente.

### La función invocable

La función callable simplemente te dice si algo es una función (en otras palabras, ¿se puede llamar?). El siguiente código devuelve False . . .

>>> callable('peas')  
False

. . . porque la cadena ’peas’ no es una función. Pero el código siguiente devuelve True . . .

>>> callable(bin)  
True

. . . porque bin es una función. El código siguiente también devolverá True :

>>> class People:  
 def run(self):  
 print('running')  
  
>>> callable(People.run)  
True

La clase People tiene una única función run . Si comprobamos si la función de la clase es invocable (lo es), obtendremos True . Además, si creamos un objeto de la clase y luego comprobamos si la función del objeto ( p.run ) es invocable, también obtendremos True :

>>> p = People()  
>>> callable(p.run)  
True

### La función chr

Cada carácter que escribes en Python tiene un código numérico subyacente que lo identifica. El carácter ’a’ , por ejemplo, tiene el valor numérico 97. Una ’A’ mayúscula tiene el valor numérico 65. La función chr toma un parámetro numérico y devuelve el carácter. Así que podemos probar los valores 97 y 65:

>>> chr(97)  
'a'  
>>> chr(65)  
'A'

Podemos probar con otros números aleatorios, como 22283, que es un carácter del juego de caracteres chino:

>>> chr(22283)

O 949, que es el carácter griego *épsilon :*

>>> chr(949)  
'ε'

O 8595, que no es un carácter, sino una flecha que apunta hacia abajo:

>>> chr(8595)  
'↓'

### La función dir

La función dir (abreviatura de *directorio* ) devuelve información sobre cualquier valor. Básicamente, te indica las funciones que se pueden utilizar con ese valor por orden alfabético.

Por ejemplo, para mostrar las funciones disponibles para un valor de la lista, introduce esto:

>>> dir(['a', 'short', 'list'])  
['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_',  
'\_\_delitem\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_',  
'\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_',  
'\_\_imul\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_',  
'\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_',  
'\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_',  
'\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'count',  
'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

La función dir funciona prácticamente con cualquier cosa, incluidas cadenas, números, funciones, módulos, objetos y clases. Pero a veces la información que devuelve puede no ser muy útil. Por ejemplo, si llamas a dir sobre el número 1, muestra una serie de funciones especiales (las que empiezan y terminan con guiones bajos) utilizadas por el propio Python, lo que no es realmente útil (normalmente puedes ignorar la mayoría de ellas):

>>> dir(1)  
['\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_bool\_\_', '\_\_ceil\_\_', '\_\_class\_\_',  
'\_\_delattr\_\_', '\_\_dir\_\_', '\_\_divmod\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_float\_\_',   
'\_\_floor\_\_', '\_\_floordiv\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_',   
'\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_index\_\_', '\_\_init\_\_',  
'\_\_init\_subclass\_\_', '\_\_int\_\_', '\_\_invert\_\_', '\_\_le\_\_', '\_\_lshift\_\_',  
'\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_neg\_\_', '\_\_new\_\_', '\_\_or\_\_',   
'\_\_pos\_\_', '\_\_pow\_\_', '\_\_radd\_\_', '\_\_rand\_\_', '\_\_rdivmod\_\_', '\_\_reduce\_\_',   
'\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rfloordiv\_\_', '\_\_rlshift\_\_', '\_\_rmod\_\_',   
'\_\_rmul\_\_', '\_\_ror\_\_', '\_\_round\_\_', '\_\_rpow\_\_', '\_\_rrshift\_\_', '\_\_rshift\_\_',   
'\_\_rsub\_\_', '\_\_rtruediv\_\_', '\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_',   
'\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_truediv\_\_', '\_\_trunc\_\_', '\_\_xor\_\_',   
'as\_integer\_ratio', 'bit\_count', 'bit\_length',  
'conjugate', 'denominator', 'from\_bytes', 'imag', 'numerator',  
'real', 'to\_bytes']

La función dir puede ser útil cuando tienes una variable y quieres averiguar rápidamente qué puedes hacer con ella. Por ejemplo, ejecuta dir utilizando la variable popcorn que contiene un valor de cadena, y obtendrás la lista de funciones que proporciona la clase string (todas las cadenas son miembros de la clase string ):

>>> popcorn = 'I love popcorn!'  
>>> dir(popcorn)  
['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_',   
'\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_',  
'\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_',   
'\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_',  
'\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_',   
'\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_',  
'\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode',   
'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index',  
'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier',   
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',   
'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',  
'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust',   
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',  
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

En este punto, podrías utilizar help para obtener una breve descripción de cualquier función de la lista. He aquí un ejemplo de ejecución de help para la función upper :

>>> help(popcorn.upper)  
Help on built-in function upper:  
  
upper() method of builtins.str instance  
 Return a copy of the string converted to uppercase.

La información devuelta puede ser un poco confusa, así que echemos un vistazo más de cerca. La primera línea te dice que upper es una función incorporada de una instancia (objeto) de una cadena. Y la segunda línea  te dice exactamente lo que hace (devuelve una copia de la cadena en mayúsculas).

### La función divmod

La función auxiliar divmod toma dos parámetros (dos números que representan un dividendo y un divisor) y devuelve el resultado de dividir los dos números, junto con el resultado de realizar la operación de módulo en los dos números. La división es la operación matemática que consiste en calcular cuántas veces se puede dividir un número entre partes de un segundo número. Por ejemplo, ¿cuántas veces podemos dividir un conjunto de seis bolas en conjuntos de dos bolas?

Image

La respuesta: podemos dividirlo tres veces.

Image

La operación modulo es casi lo mismo, salvo que el módulo devuelve el número que sobra después de hacer la división. Así que el resultado del módulo para las seis bolas divididas por dos de arriba es cero (porque no sobra nada). ¿Y si añadimos una bola más? Si dividimos siete bolas en conjuntos de dos, el resultado de la división sigue siendo tres, pero sobrará una bola.

Image

Y eso es lo que divmod devolverá como una tupla de dos números: el resultado de la división y el resultado de la operación módulo. Probemos primero con 6 y 2:

>>> divmod(6, 2)  
(3, 0)

Y luego con 7 y 2:

>>> divmod(7, 2)  
(3, 1)

### La función eval

La función eval (abreviatura de *evaluar* ) toma una cadena como parámetro y la ejecuta como si fuera una expresión de Python. Por ejemplo, eval(’print(’’wow’’)’) ejecutará en realidad la expresión print(’’wow’’) .

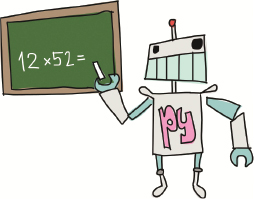
La función eval sólo funciona con expresiones sencillas, como las siguientes:

>>> eval('10\*5')  
50

Las expresiones que se dividen en más de una línea (como las sentencias if ) generalmente no se evaluarán, como en este ejemplo:

>>> eval('''if True:  
 print("this won't work at all")''')  
  
Traceback (most recent call last):  
 File "<pyshell#2>", line 1, in <module>  
 eval('''if True:  
 File "<string>", line 1  
 if True:  
 ^^  
SyntaxError: invalid syntax

La función eval se utiliza a menudo para convertir las entradas del usuario en expresiones de Python. Por ejemplo, puedes escribir un sencillo programa calculadora que lea las ecuaciones introducidas en Python y luego calcule (evalúe) las respuestas.



Como la entrada del usuario se lee como una cadena, Python necesita convertirla en números y operadores antes de realizar cualquier cálculo. La función eval facilita esa conversión:

>>> your\_calculation = input('Enter a calculation: ')  
  
Enter a calculation: 12\*52  
>>> eval(your\_calculation)  
624

En este ejemplo, utilizamos la función input para leer lo que el usuario introduce en la variable your\_calculation . En la línea siguiente, introducimos la expresión 12\*52 . Utilizamos eval para ejecutar este cálculo, y el resultado se imprime en la última línea.

### La función exec

La función exec es como eval , salvo que puedes utilizarla para ejecutar programas más complicados. Mientras que eval devuelve un valor (algo que puedes guardar en una variable), exec no lo hace. He aquí un ejemplo:

>>> my\_small\_program = '''print('ham')  
print('sandwich')'''  
>>> exec(my\_small\_program)  
ham  
sandwich

En las dos primeras líneas, creamos una variable con una cadena multilínea que contiene dos sentencias print , y luego utilizamos exec para ejecutar la cadena.

Puedes utilizar exec para ejecutar miniprogramas que tu programa Python lea de archivos: ¡programas dentro de programas! Esto puede ser muy útil al escribir aplicaciones largas y complejas. Por ejemplo, podrías crear un juego de *Duelo de Robots* , en el que dos robots se mueven por una pantalla e intentan atacarse mutuamente. Los jugadores del juego proporcionarían las instrucciones para su robot como miniprogramas Python. El juego *Duelo* de Robots leería estos scripts y utilizaría exec para ejecutarse.

### La función float

La función float convierte una cadena o un número en un número *de coma flotante* , que es un número con un decimal (también llamado *número real* o *flotante* ). Por ejemplo, el número 10 es un número entero (también llamado *número entero* ) *,* pero 10,0, 10,1 y 10,253 son números de coma flotante. Puedes utilizar números de coma flotante si escribes un programa sencillo para calcular cantidades monetarias. Los números flotantes también se utilizan en programas gráficos (como los juegos 3D) para calcular cómo y dónde dibujar cosas en la pantalla.



Puedes convertir una cadena en un flotante simplemente llamando a float :

>>> float('12')  
12.0

También puedes utilizar un decimal en una cadena:

>>> float('123.456789')  
123.456789

Podrías utilizar float para convertir los valores introducidos en tu programa en números propiamente dichos, lo que resulta especialmente útil cuando necesitas comparar el valor que introduce una persona con otros valores. Por ejemplo, para comprobar si la edad de una persona es superior a un determinado número, podríamos hacer lo siguiente:

>>> your\_age = input('Enter your age: ')  
  
Enter your age: 20  
>>> age = float(your\_age)  
>>> if age > 13:  
 print(f'You are {age - 13} years too old')  
  
You are 7.0 years too old

### La entrada Función

La función input se utiliza para leer el texto introducido por la persona que está utilizando tu programa: todo lo que escribe hasta que pulsa la tecla INTRO. Los resultados se devuelven en una cadena para que puedas utilizarla. Puedes pedir al usuario de tu programa que introduzca algo, con un mensaje como el de este ejemplo:

>>> s = input('Tell me a play on words:\n')  
Tell me a play on words:  
A hedgehog went to see a play about a plucky young girl, but left  
dis-a-pointed  
>>> print(s)  
A hedgehog went to see a play about a plucky young girl, but left  
dis-a-pointed

O sin ningún mensaje:

>>> s = input()  
A hedgehog went to see a play about a plucky young girl, but left  
dis-a-pointed

En cualquier caso, el resultado de la función input es el mismo: una cadena que contiene el texto. Consulta la sección anterior sobre la función float para ver más ejemplos de utilización del valor de retorno.

### La función int

La función int convierte una cadena o un número en un número entero (o *integer* ), lo que básicamente significa que se elimina todo lo que hay después del punto decimal. Por ejemplo, así es como se convierte un número de coma flotante en un número entero:

>>> int(123.456)  
123

Este ejemplo convierte una cadena en un número entero:

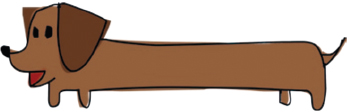
>>> int('123')  
123

Pero si intentas convertir una cadena que contiene un número en coma flotante en un número entero, obtendrás un mensaje de error. Para el ejemplo  , aquí intentamos convertir una cadena que contiene un número en coma flotante utilizando la función int :

>>> int('123.456')  
Traceback (most recent call last):  
 File "<pyshell>", line 1, in <module>  
 int('123.456')  
ValueError: invalid literal for int() with base 10: '123.456'

Como puedes ver, el resultado es un mensaje ValueError .

### La función len



La función len devuelve la longitud de un objeto o, en el caso de una cadena, el número de caracteres de la cadena. Por ejemplo, para obtener la longitud de ’this is a test string’ , harías lo siguiente:

>>> len('this is a test string')  
21

Cuando se utiliza con una lista o una tupla, len devuelve el número de elementos de esa lista o tupla:

>>> creature\_list = ['unicorn', 'cyclops', 'fairy', 'elf', 'dragon',   
 'troll']  
>>> print(len(creature\_list))  
6

Si se utiliza con un dict (o *diccionario* ), len también devuelve el número de elementos del dict :

>>> enemies = {'Batman' : 'Joker',  
 'Superman' : 'Lex Luthor',   
 'Spiderman' : 'Green Goblin'}  
>>> print(len(enemies))  
3

La función len es especialmente útil cuando trabajas con bucles. Por ejemplo, podríamos utilizarla para mostrar las posiciones de los índices de los elementos de una lista, de esta forma

>>> fruit = ['apple', 'banana', 'clementine', 'dragon fruit']  
>>> length = len(fruit)  
>>> for x in range(0, length):  
 print(f'the fruit at index {x} is {fruit[x]}')  
  
the fruit at index 0 is apple  
the fruit at index 1 is banana  
the fruit at index 2 is clementine  
the fruit at index 3 is dragon fruit

Primero, almacenamos la longitud de la lista en la variable length , y luego utilizamos esa variable en la función range para crear nuestro bucle. A medida que recorremos cada elemento de la lista, imprimimos un mensaje mostrando la posición del índice del elemento y su valor. También podrías utilizar la función len , si tuvieras una lista de cadenas y quisieras imprimir cada segundo o tercer elemento de la lista.

### La función lista

Si llamas a list sin ningún parámetro, obtendrás como respuesta un objeto lista vacío. En ese punto no hay diferencia entre list() y utilizar corchetes. Podemos comprobarlo comprobando si las dos listas son iguales ( == ):

>>> l1 = list()  
>>> l2 = []  
>>> l1 == l2  
True

Aunque no parezca especialmente útil, list también puede utilizarse para convertir ciertos tipos de objetos Python (llamados *iterables* ) en una lista. El ejemplo más sencillo sería utilizar la función range (esta función se describe en [la página 317](app02.xhtml#app02lev1sec21) ) con list :

>>> list(range(0, 10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

### Las funciones max y min

La función max devuelve el mayor elemento de una lista, tupla o cadena. Por ejemplo, aquí tienes cómo utilizarla con una lista de números:

>>> numbers = [5, 4, 10, 30, 22]  
>>> print(max(numbers))  
30

Puedes hacer exactamente lo mismo con una cadena o una lista de cadenas:



>>> strings = 'stringSTRING'  
>>> print(max(strings))  
t  
>>> strings = ['s', 't', 'r', 'i', 'n', 'g', 'S', 'T', 'R', 'I', 'N', 'G']  
>>> print(max(strings))  
t

Las letras se ordenan alfabéticamente, pero las minúsculas van después de las mayúsculas, así que t es más que T . Pero no tienes por qué utilizar listas, tuplas o cadenas. También puedes llamar directamente a la función max e introducir en los paréntesis, como parámetros, los elementos que quieras comparar:

>>> print(max(10, 300, 450, 50, 90))  
450

La función min funciona como max , salvo que devuelve el elemento más pequeño de la lista, tupla o cadena. Este es nuestro ejemplo de lista de números utilizando min en lugar de max :

>>> numbers = [5, 4, 10, 30, 22]  
>>> print(min(numbers))  
4

Supón que estás jugando a un juego de adivinanzas con un equipo de cuatro jugadores, y cada uno tiene que adivinar un número que sea menor que tu número. Si algún jugador adivina por encima de tu número, todos los jugadores pierden, pero si todos adivinan por debajo, ganan. Podríamos utilizar  max para averiguar rápidamente si alguna de las suposiciones es mayor, de este modo:

>>> guess\_this\_number = 61  
>>> player\_guesses = [12, 15, 70, 45]  
>>> if max(player\_guesses) > guess\_this\_number:  
 print('Boom! You all lose')  
 else:  
 print('You win')  
  
Boom! You all lose

En este ejemplo, almacenamos el número a adivinar utilizando la variable guess\_this\_number . Las suposiciones de los miembros del equipo se almacenan en la lista player\_guesses . La sentencia if comprueba el máximo adivinado con el número de guess\_this\_number , y si algún jugador adivina por encima del número, imprimimos el mensaje "¡Boom! Todos habéis perdido".

### La función ord

La función ord es básicamente la inversa de la función chr : mientras que chr convierte un número en un carácter, ord te dice cuál es el código numérico de un carácter. Aquí tienes algunos ejemplos:

>>> ord('a')  
97  
>>> ord('A')  
65  
>>> ord('')  
22283

### La función pow

La función pow toma dos números y calcula el valor de un número (llamémoslo x ) a la potencia del otro número (llamémoslo y ). Esencialmente, pow multiplicará x por sí mismo y veces. Por ejemplo, 2 a la potencia de 3 (en términos matemáticos es 2 3 ) sería 2 \* 2 \* 2 (o en notación matemática,  2 × 2 × 2), que es 8 (2 \* 2 es 4, 4 \* 2 es 8). Otro ejemplo: 3 a la potencia de 3 (3 3 ) es 27. Veamos cómo queda esto en código:

>>> pow(2, 3)  
8  
>>> pow(3, 3)  
27

### La función rango

La función range se utiliza principalmente en los bucles for para recorrer una sección de código un número determinado de veces. Los dos primeros parámetros que se dan a range se llaman *inicio* y *parada .* Has visto range con estos dos parámetros en el ejemplo anterior en el que se utilizaba la función len para trabajar con un bucle.



Los números que genera range comienzan con el número dado como primer parámetro y terminan con el número que sea uno menos que el segundo parámetro. Por ejemplo, a continuación se muestra lo que ocurre cuando imprimimos los números que crea range entre 0 y 5:

>>> for x in range(0, 5):  
 print(x)  
  
0  
1  
2  
3  
4

La función range en realidad devuelve un objeto especial llamado *iterador* que repite una acción un número de veces. En este caso, devuelve el siguiente número más alto cada vez que se llama.

Puedes convertir el iterador en una lista utilizando la función list . Si luego imprimes el valor devuelto al llamar a range , verás también los números que contiene:

>>> print(list(range(0, 5)))  
[0, 1, 2, 3, 4]

También puedes añadir un tercer parámetro a range , llamado step . Si no se incluye el valor step , se utilizará por defecto el número 1 como step . Pero, ¿qué ocurre si pasamos el número 2 como step ? Éste es el resultado:

>>> count\_by\_twos = list(range(0, 30, 2))  
>>> print(count\_by\_twos)  
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]

Cada número de la lista aumenta en dos respecto al número anterior, y la lista termina con el número 28, que es 2 menos que 30. También puedes utilizar pasos negativos:

>>> count\_down\_by\_twos = list(range(40, 10, -2))  
>>> print(count\_down\_by\_twos)  
[40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12]

### La función suma

La función sum suma los elementos de una lista y devuelve el total. Aquí tienes un ejemplo:

>>> my\_list\_of\_numbers = list(range(0, 500, 50))  
>>> print(my\_list\_of\_numbers)  
[0, 50, 100, 150, 200, 250, 300, 350, 400, 450]  
>>> print(sum(my\_list\_of\_numbers))  
2250

En la primera línea, creamos una lista de números entre 0 y 500, utilizando range con un paso de 50. A continuación, imprimimos la lista para ver el resultado. Por último, pasando la variable my\_list\_of\_numbers a la función sum con print(sum(my\_list\_of\_numbers)) sumamos todos los elementos de la lista, obteniendo el total de 2250.

### Abrir un archivo en Python

La función incorporada de Python open abre un archivo, para que puedas hacer algo útil con él (como mostrar su contenido). La forma de indicar a la función qué archivo debe abrir depende de tu sistema operativo. Lee el ejemplo para un archivo de Windows, y luego lee la sección específica para Mac o Ubuntu si utilizas uno de esos sistemas. Primero crea un archivo de **texto** plano en tu carpeta de inicio llamado *prueba.txt -en* Windows, puedes utilizar el Bloc de Notas; en Ubuntu Linux o Raspberry Pi, utiliza TextEditor; en macOS, utiliza TextEdit (pero en TextEdit, tendrás que seleccionar **Formato** **▸ Hacer texto plano** ). Puedes poner lo que quieras en el archivo.

#### Abrir un archivo de Windows

Si utilizas Windows, introduce el siguiente código para abrir *test.txt :*

>>> test\_file = open('c:\\Users\\<your username>\\test.txt')  
>>> text = test\_file.read()  
>>> print(text)  
There once was a boy named Marcelo  
Who dreamed he ate a marshmallow  
He awoke with a start  
As his bed fell apart  
And he found he was a much rounder fellow

En la primera línea, utilizamos open , que devuelve un objeto archivo con funciones para trabajar con archivos. El parámetro que utilizamos con la función open es una cadena que indica a Python dónde encontrar el archivo. Si utilizas Windows, has guardado prueba. *txt* en tu carpeta de usuario de la unidad *C:* , así que especifica la ubicación de tu archivo como c:\Users\<your username>\test.txt . (¡No olvides sustituir <your username> por tu nombre de usuario real!)

Las dos barras invertidas en el nombre de archivo de Windows indican a Python que la barra invertida es sólo eso, y no una especie de comando. (Como leíste en el [Capítulo 3](ch03.xhtml#ch03) , las barras invertidas por sí solas tienen un significado especial en Python, sobre todo en las cadenas). Guardamos el objeto archivo en la variable test\_file .

En la segunda línea, utilizamos la función read , proporcionada por el objeto archivo, para leer el contenido del archivo y almacenarlo en la variable text . Imprimimos la variable en la última línea para mostrar el contenido del archivo.

#### Abrir un archivo macOS

Si utilizas macOS, tendrás que introducir una ubicación diferente en la primera línea del ejemplo de Windows para abrir *prueba.txt .* Utiliza en la cadena el nombre de usuario que seleccionaste al guardar el archivo de texto. Por ejemplo, si el nombre de usuario es *sarahwinters* , el parámetro open debería tener este aspecto:

>>> test\_file = open('/Users/sarahwinters/test.txt')

#### Abrir un archivo de Ubuntu o Raspberry Pi

Si utilizas Ubuntu Linux o Raspberry Pi, tendrás que introducir una ubicación diferente en la primera línea del ejemplo de Windows para abrir *test.txt .* Utiliza el nombre de usuario que seleccionaste al guardar el archivo de texto. Por ejemplo, si el nombre de usuario es *jacob* , el parámetro open debería tener este aspecto:

>>> test\_file = open('/home/jacob/test.txt')

### Escribir en archivos

El objeto archivo devuelto por open tiene otras funciones además de read . Podemos crear un nuevo archivo vacío utilizando un segundo parámetro -la cadena ’w’ - cuando llamemos a open (este parámetro indica a Python que queremos escribir en el objeto archivo, en lugar de leer de él):

>>> test\_file = open('c:\\Users\\rachel\\myfile.txt', 'w')

Ahora podemos añadir información a este nuevo archivo utilizando la función write :

>>> test\_file = open('c:\\Users\\rachel\\myfile.txt', 'w')  
>>> test\_file.write('What is green and loud? A froghorn!')  
20

Por último, tenemos que indicar a Python cuándo hemos terminado de escribir en el archivo, utilizando la función close :

>>> test\_file = open('c:\\Users\\rachel\\myfile.txt', 'w')  
>>> test\_file.write('What is green and loud? A froghorn!')  
>>> test\_file.close()

Ahora, si abres el archivo con tu editor de texto, deberías ver que contiene el texto "¿Qué es verde y ruidoso? Un cuerno de rana". O bien, puedes utilizar Python para volver a leerlo:



>>> test\_file = open('c:\\Users\\rachel\\myfile.txt')  
>>> print(test\_file.read())  
What is green and loud? A froghorn!