Capítulo 2.2
Funciones y módulos

1. Las funciones

1.1 ¿Por qué utilizar funciones?

Cuando se desarrolla, se utilizan muchas funciones, como por ejemplo **`print`** o **`input`**. Estas últimas son bastante simples de manipular por varios motivos:

– poseen un nombre sencillo que indica con claridad para qué sirven;

– reciben parámetros que permiten variar la manera en la que se utilizan;

– no necesitamos saber cómo están escritas, simplemente qué van a hacer.

Cuando escribe su propio código, debe diseñar algoritmos más o menos complejos, y cuando no está organizado, puede producir lo que hemos hecho hasta el momento: un código perfectamente lineal.

El principal inconveniente es el siguiente: el código es una larga prosa, sin descansos particulares. Es difícil aislar una parte y saber qué línea se invoca y en cada momento. Habitualmente, consideramos que una función bien hecha debe tener unas diez líneas de media, y 20 o 25 como máximo.

Estas métricas no son, realmente, obligaciones, sino un orden de magnitud para tener en mente e intentar respetar y obtener así un código legible y comprensible por todos, incluido usted algunos meses más tarde, pues lo que ha escrito solo lo tendrá fresco en el momento de su escritura.

En efecto, un código demasiado largo es difícil de leer, de aprender y de mantener.

La realidad es clara, hay que organizar el propio código para que esté formado por pequeños bloques simples y fáciles de identificar. La verdadera dificultad es saber cómo delimitar estos bloques, cómo construir bellas funciones que sean lo suficientemente precisas como para hacer lo que deseamos con detalle, pero también lo suficientemente genéricas como para no tener que construir dos funciones que sean casi idénticas y que solo se distingan por un pequeño detalle.

Un buen punto para comenzar consiste en mirar el código producido hasta el momento (la solución al ejercicio del final del capítulo anterior) e identificar duplicados en el código:

```
print("Introduzca el número a adivinar")
while True:
    numero = input("Introduzca un número entre 0 y 99: ")
    try:
        numero = int(numero)
    except:
        pass
    else:
        if 0 <= numero <= 99:
            break

# PARTE 2
print("Intente adivinar el número")
while True:  # BUCLE 1
    while True:  # BUCLE 2
        intento = input("Introduzca un número entre 0 y 99: ")
        try:
            intento = int(intento)
        except:
            pass
        else:
            if 0 <= intento <= 99:
                break  # Bucle 2

    if intento < numero:
        print("Demasiado pequeño")
    elif intento > numero:
        print("Demasiado grande")
    else:
        print("¡Ha ganado!")
        break  # Bucle 1
```

Observación

Encontrará este ejemplo en la carpeta Guía de los archivos para descargar. Se llama 11_JUEGO_guess_the_number.py.

Vemos que en este extracto de código, pedir la información del número que hay que encontrar y la del número que se debe adivinar es casi lo mismo: cambia la primera visualización, que indica al usuario lo que se pide de lo que cambia.

Observe que eliminar los duplicados del código es algo muy importante, pues cuando tenga que mantener un código, si tiene que cambiar cualquier cosa, tendrá que repetirlo sobre todos los duplicados y resulta bastante fácil olvidarse de alguno durante la operación.

Este objetivo de eliminación de duplicados es nuestro hilo conductor y donde vamos a empezar definiendo nuestra primera función útil.

Sin embargo, no olvide que en la vida real tendrá que reflexionar primero y codificar después y que, en consecuencia, tendrá que definir aquellos bloques que quiera crear antes de crearlos realmente y no escribir primero un código y reflexionar cómo hacerlo legible más adelante.

1.2 Introducción a las funciones

1.2.1 Cómo declarar una función

En Python, los principios sintácticos son siempre los mismos. El código de una función, al ser un bloque, exige que la sintaxis de una función sea la de un bloque.
Vamos a escribir la palabra clave **`def`**, que permite indicar que se está definiendo una función, a continuación el nombre de esta función, seguido de un paréntesis (veremos más adelante qué poner dentro) y los famosos dos puntos. Esta primera línea se llama la **firma de la función**.
Lo que hay a continuación, y que se escribe indentado, es el **cuerpo de la función**. Veamos lo que se obtiene:

```
def pedir_numero():
    while True:
        entrada = input("Introduzca un número entre 0 y 99: ")
        try:
            entrada = int(entrada)
        except:
            pass
        else:
            if 9 <= entrada <= 99:
                break
    return entrada
```

A excepción de la primera y última líneas, el conjunto de este extracto de código es completamente idéntico a la parte que estaba duplicada en nuestra primera versión del juego.
La única diferencia es el nombre de la variable, que era **`numero`**, y luego **`intento`**, y que ahora se llama **`entrada`**.
En efecto, el nombre de las variables correspondía, en el programa de partida, respectivamente con el número que se debe adivinar y luego con los intentos del jugador.
Aquí tenemos una función cuyo objetivo es simplemente pedir al usuario que introduzca un número cualquiera. A nivel de la función, no se sabe para qué va a servir este número, no se conoce su nombre y tampoco hace falta saberlo. Se sabe únicamente que se trata de una entrada, de modo que se decide llamarlo así.
Las cosas se ponen interesantes cuando utilizamos nuestra función:

```
# PARTE 1
print("Introduzca el número a adivinar")
numero = pedir_numero()

# PARTE 2
print("Intente adivinar el número")
while True:
```

```
intento = pedir_numero()
if intento < numero:
    print("Demasiado pequeño")
elif intento > numero:
    print("Demasiado grande")
else:
    print "¡Ha ganado!"
    break
```

Vemos que el código es considerablemente más corto y que encontramos nuestras variables **`numero`** e **`intento`**, como resultado de la función.
Gracias a que la función devuelve la entrada, es posible realizar esta asignación: comprende ahora el sentido de la instrucción **`return`** al final de la función.

Observación

Encontrará este ejemplo en la carpeta Guía de los archivos para descargar. Se llama 12_Funciones.py.

Hemos escrito nuestra primera función y el programa se comporta, desde el punto de vista del usuario, exactamente igual.

1.2.2 Gestión de un parámetro

Podemos mejorar fácilmente nuestra función. En efecto, en lugar de mostrar información antes de invocar nuestra función, podemos cambiar la invitación a introducir un número.
Para ello, hay que pasar un parámetro, es decir, que cuando se invoque la función tendremos que darle los elementos para que pueda comportarse como deseamos.
En nuestro ejemplo, queremos definir los valores mínimo y máximo una única vez: vamos a utilizar constantes.
Esta noción es muy importante, pues utilizando esta constante en lugar de un literal, dispondremos de los medios para cambiar este valor de manera sencilla: basta con modificar la constante sin tener que recorrer todo el código para buscar un literal y modificarlo.
De este modo, una constante se define exactamente como una variable, salvo que se escribe en mayúsculas:

```
MIN = 0
MAX = 99
```

En Python, una constante es una variable como cualquier otra. La única convención consiste en escribirla en mayúsculas, aunque puede modificarlas; nada se lo impide.

Observación

Python funciona bastante con convenciones: le da las herramientas para hacer las cosas de manera correcta, pero no le impone restricciones. Python parte del principio de que el desarrollador sabe lo que hace y confía plenamente en que hará lo correcto: si por cualquier motivo no respeta la convención, es porque posee una razón de peso para no hacerlo y Python lo respetará.

El uso de estas constantes mejora la legibilidad y la comprensión del código pues, tras su declaración, se comprende inmediatamente de qué se trata y si vemos **MIN** o **MAX** más adelante en el código, sabremos a qué se refiere, mejor que usando literales.
He aquí la función ligeramente modificada:

```
def pedir_numero(invitacion):
    # Se completa la invitacion:
    invitacion += " entre " + str(MIN) + " y " + str(MAX) + ": "

    while True:
        entrada = input(invitacion)
        try:
            entrada  = int(entrada)
        except:
            pass
        else:
            if MIN <= entrada <= MAX:
                break
    return entrada
```

Damos la posibilidad de invocar nuestra función diciendo lo que hay que introducir, y se completa esta información precisando los límites del número que se debe indicar.
Observe que las constantes **MIN** y **MAX** se definen fuera de la función. Por lo tanto, son accesibles. Ocurre así también con todas las variables que se definen, en el momento en que se invoque la función.

Observación

Salvo casos excepcionales que llegaremos a dominar, evitaremos el uso en una función de una variable que pueda no estar definida en el momento en que se invoque esta función.

Si reflexionamos con calma, las funciones **int** e **input** se definen también fuera de la función, y si habíamos importado un módulo al inicio del archivo, este será accesible desde el interior de la función.
Si queremos ir más allá acerca de estas cuestiones, tendremos que dirigirnos al capítulo Declaraciones - sección Visibilidad que aborda el **ámbito de una variable**.
He aquí el código que hace uso de esta función:

```
# PARTE 1
numero = pedir_numero("Introduzca el número a adivinar")

# PARTE 2
while True:
    intento = pedir_numero("Adivine el número")
    if intento < numero:
        print("Demasiado pequeño")
    elif intento > numero:
        print("Demasiado grande")
    else:
```

```
        print("¡Ha ganado!")
        break
```

El código escrito es mucho más legible: se sabe enseguida por qué se utiliza nuestra función y lo que va a producir.

Observación

Encontrará este ejemplo en la carpeta Guía de los archivos para descargar. Se llama 13_Funciones_genéricas_1.py.

Veremos ahora cómo puede utilizarse esta función de una manera todavía más inteligente.

1.2.3 Cómo hacer la función más genérica

La función, tal y como la hemos escrito, depende de **MIN** y de **MAX**. Si queremos que estos dos valores puedan variar, tendremos que dejar de utilizar las constantes. Pero la propia función no sabe cómo pueden variar estos dos valores.
Estos valores deben convertirse en parámetros:

```
def pedir_numero(invitacion, minimo, maximo):
    invitacion += " entre " + str(minimo) + " y " +
                  str(maximo) + " : "

    while True:
        entrada = input(invitacion)
        try:
            entrada = int(entrada)
        except:
            pass
        else:
             if minimo <= entrada <= maximo:
                break
    return entrada
```

Vemos aparecer dos nuevos parámetros, **minimo** y **maximo** y, respecto al ejemplo anterior, hemos reemplazado **MIN** por minimo y **MAX** por **maximo**, simplemente.

Truco

Observe el salto de línea entre las líneas 2 y 3: como la línea 2 termina con un +, Python sabe que la línea 3 es la continuación de la instrucción que empieza en la línea 2. Por convención, como esta instrucción es una asignación, se alinea la línea 3 con el principio del operando derecho.