## 2.6. Funciones predefinidas

Hemos estudiado los operadores aritméticos básicos. Python también proporciona funciones que podemos utilizar en las expresiones. Estas funciones se dice que están *predefinidas*.[6]

La función *abs*, por ejemplo, calcula el valor absoluto de un número. Podemos usarla como en estas expresiones:

```
>>> abs(-3) ↲
3
>>> abs(3) ↲
3
```

El número sobre el que se aplica la función se denomina *argumento*. Observa que el argumento de la función debe ir encerrado entre paréntesis:

```
>>> abs(0) ↲
0
>>> abs 0 ↲
  File "<stdin>", line 1
    abs 0
        ^
SyntaxError: invalid syntax
```

Existen muchas funciones predefinidas, pero es pronto para aprenderlas todas. Te resumimos algunas que ya puedes utilizar:

- *float*: conversión a flotante. Si recibe un número entero como argumento, devuelve el mismo número convertido en un flotante equivalente.

  ```
  >>> float(3) ↲
  3.0
  ```

  La función *float* también acepta argumentos de tipo cadena. Cuando se le pasa una cadena, *float* la convierte en el número flotante que ésta representa:

[6] *Pre*definidas porque nosotros también podemos definir nuestras propias funciones. Ya llegaremos.

```
>>> float('3.2') ↲
3.2
>>> float('3.2e10') ↲
32000000000.0
```

Pero si la cadena no representa un flotante, se produce un error de tipo *ValueError*, es decir, «error de valor»:

```
>>> float('un texto') ↲
Traceback (innermost last):
  File "<stdin>", line 1, in ?
ValueError: invalid literal for float(): un texto
```

Si *float* recibe un argumento flotante, devuelve el mismo valor que se suministra como argumento.

- *int*: conversión a entero. Si recibe un número flotante como argumento, devuelve el entero que se obtiene eliminando la parte fraccionaria.[7]

```
>>> int(2.1) ↲
2
>>> int(-2.9) ↲
-2
```

También la función *int* acepta como argumento una cadena:

```
>>> int('2') ↲
2
```

Si *int* recibe un argumento entero, devuelve el argumento tal cual.

- *str*: conversión a cadena. Recibe un número y devuelve una representación de éste como cadena.

```
>>> str(2.1) ↲
'2.1'
>>> str(234E47) ↲
'2.34e+49'
```

La función *str* también puede recibir como argumento una cadena, pero en ese caso devuelve como resultado la misma cadena.

- *round*: redondeo. Puede usarse con uno o dos argumentos. Si se usa con un sólo argumento, redondea el número al flotante más próximo cuya parte decimal sea nula.

```
>>> round(2.1) ↲
2.0
>>> round(2.9) ↲
3.0
>>> round(-2.9) ↲
-3.0
>>> round(2) ↲
2.0
```

[7] El redondeo de *int* puede ser al alza o a la baja según el ordenador en que lo ejecutes. Esto es así porque *int* se apoya en el comportamiento del redondeo automático de C (el intérprete de Python que usamos está escrito en C) y su comportamiento está indefinido. Si quieres un comportamiento homogéneo del redondeo, pues usar las funciones *round*, *floor* o *ceil*, que se explican más adelante.

(¡Observa que el resultado siempre es de tipo flotante!) Si *round* recibe dos argumentos, *éstos deben ir separados por una coma* y el segundo indica el número de decimales que deseamos conservar tras el redondeo.

```
>>> round(2.1451, 2) ↲
2.15
>>> round(2.1451, 3) ↲
2.145
>>> round(2.1451, 0) ↲
2.0
```

Estas funciones (y las que estudiaremos más adelante) pueden formar parte de expresiones y sus argumentos pueden, a su vez, ser expresiones. Observa los siguientes ejemplos:

```
>>> abs(-23) % int(7.3) ↲
2
>>> abs(round(-34.2765,1)) ↲
34.3
>>> str(float(str(2) * 3 + '.123')) + '321' ↲
222.123321
```

EJERCICIOS

▶ 27 Calcula con una única expresión el valor absoluto del redondeo de −3.2. (El resultado es 3.0.)

▶ 28 Convierte (en una única expresión) a una cadena el resultado de la división 5011/10000 redondeado con 3 decimales.

▶ 29 ¿Qué resulta de evaluar estas expresiones?

```
>>> str(2.1) + str(1.2) ↲
>>> int(str(2) + str(3)) ↲
>>> str(int(12.3)) + '0' ↲
>>> int('2'+'3') ↲
>>> str(2 + 3) ↲
>>> str(int(2.1) + float(3)) ↲
```