2.4. Variables y asignaciones

En ocasiones deseamos que el ordenador recuerde ciertos valores para usarlos más adelante. Por ejemplo, supongamos que deseamos efectuar el cálculo del perímetro y el área de un círculo de radio 1.298373 m. La fórmula del perímetro es $2\pi r$, donde r es el radio, y la fórmula del área es πr^2. (Aproximaremos el valor de π con 3.14159265359.) Podemos realizar ambos cálculos del siguiente modo:

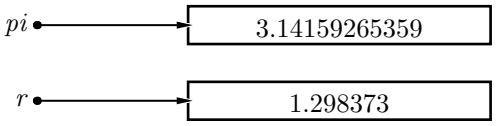
```
>>> 2 * 3.14159265359 * 1.298373 ↲
8.1579181568392176
>>> 3.14159265359 * 1.298373 ** 2 ↲
5.2960103355249037
```

Observa que hemos tenido que introducir dos veces los valores de π y r por lo que, al tener tantos decimales, es muy fácil cometer errores. Para paliar este problema podemos utilizar *variables*:

```
>>> pi = 3.14159265359 ↲
>>> r = 1.298373 ↲
>>> 2 * pi * r ↲
8.1579181568392176
>>> pi * r ** 2 ↲
5.2960103355249037
```

En la primera línea hemos creado una variable de nombre *pi* y valor 3.14159265359. A continuación, hemos creado otra variable, *r*, y le hemos dado el valor 1.298373. El acto de dar valor a una variable se denomina *asignación*. Al asignar un valor a una variable

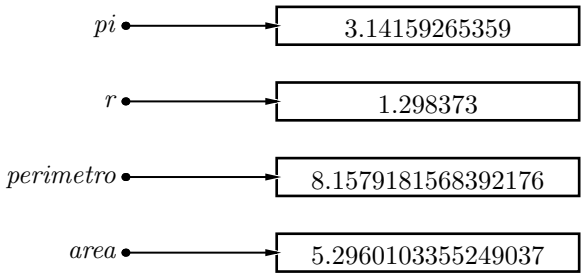
que no existía, Python reserva un espacio en la memoria, almacena el valor en él y crea una asociación entre el nombre de la variable y la dirección de memoria de dicho espacio. Podemos representar gráficamente el resultado de estas acciones así:



A partir de ese instante, escribir *pi* es equivalente a escribir 3.14159265359, y escribir *r* es equivalente a escribir 1.298373.

Podemos almacenar el resultado de calcular el perímetro y el área en sendas variables:

```
>>> pi = 3.14159265359 ↲
>>> r = 1.298373 ↲
>>> perimetro = 2 * pi * r ↲
>>> area = pi * r**2 ↲
```



La memoria se ha reservado correctamente, en ella se ha almacenado el valor correspondiente y la asociación entre la memoria y el nombre de la variable se ha establecido, pero no obtenemos respuesta alguna por pantalla. Debes tener en cuenta que las asignaciones son «mudas», es decir, no provocan salida por pantalla. Si deseamos ver cuánto vale una variable, podemos evaluar una expresión que sólo contiene a dicha variable:

```
>>> area ↲
5.2960103355249037
```

Así pues, para asignar valor a una variable basta ejecutar una sentencia como ésta:

variable = *expresión*

Ten cuidado: el orden es importante. Hacer «*expresión* = *variable*» no es equivalente. *Una asignación no es una ecuación matemática*, sino una acción consistente en (por este orden):

1. evaluar la expresión *a la derecha* del símbolo igual (=), y
2. guardar el valor resultante en la variable indicada *a la izquierda* del símbolo igual.

Se puede asignar valor a una misma variable cuantas veces se quiera. El efecto es que la variable, en cada instante, sólo «recuerda» el último valor asignado... hasta que se le asigne otro.

```
>>> a = 1 ↲
>>> 2 * a ↲
2
>>> a + 2 ↲
3
>>> a = 2 ↲
>>> a * a ↲
4
```

== no es = (comparar no es asignar)

Al aprender a programar, muchas personas confunden el operador de asignación, **=**, con el operador de comparación, **==**. El primero se usa exclusivamente para asignar un valor a una variable. El segundo, para comparar valores.

Observa la diferente respuesta que obtienes al usar **=** y **==** en el entorno interactivo:

```
>>> a = 10 ↲
>>> a ↲
10
>>> a == 1 ↲
False
>>> a ↲
10
```

Una asignación no es una ecuación

Hemos de insistir en que las asignaciones no son ecuaciones matemáticas, por mucho que su aspecto nos recuerde a éstas. Fíjate en este ejemplo, que suele sorprender a aquellos que empiezan a programar:

```
>>> x = 3 ↲
>>> x = x + 1 ↲
>>> x ↲
4
```

La primera línea asigna a la variable x el valor 3. La segunda línea parece más complicada. Si la interpretas como una ecuación, no tiene sentido, pues de ella se concluye absurdamente que $3 = 4$ o, sustrayendo la x a ambos lados del igual, que $0 = 1$. Pero si seguimos paso a paso las acciones que ejecuta Python al hacer una asignación, la cosa cambia:

1. Se evalúa la parte derecha del igual (sin tener en cuenta para nada la parte izquierda). El valor de x es 3, que sumado a 1 da 4.
2. El resultado (el 4), se almacena en la variable que aparece en la parte izquierda del igual, es decir, en x.

Así pues, el resultado de ejecutar las dos primeras líneas es que x vale 4.

El nombre de una variable es su *identificador*. Hay unas reglas precisas para construir identificadores. Si no se siguen, diremos que el identificador no es válido. Un identificador debe estar formado por letras[5] minúsculas, mayúsculas, dígitos y/o el carácter de subrayado (_), con una restricción: que el primer carácter no sea un dígito.

Hay una norma más: un identificador no puede coincidir con una *palabra reservada* o *palabra clave*. Una palabra reservada es una palabra que tiene un significado predefinido y es necesaria para expresar ciertas construcciones del lenguaje. Aquí tienes una lista con todas las palabras reservadas de Python: **and**, **assert**, **break**, **class**, **continue**, **def**, **del**, **elif**, **else**, **except**, **exec**, **finally**, **for**, **from**, **global**, **if**, **import**, **in**, **is**, **lambda**, **not**, **or**, **pass**, **print**, **raise**, **return**, **try**, **while** y **yield**.

Por ejemplo, los siguientes identificadores son válidos: *h*, *x*, *Z*, *velocidad*, *aceleracion*, *x*, *fuerza*1, *masa_* 2, *_a*, *a_* , *prueba_* 123, *desviacion_tipica*. Debes tener presente que Python distingue entre mayúsculas y minúsculas, así que *area*, *Area* y *AREA* son tres

[5] Exceptuando los símbolos que no son propios del alfabeto inglés, como las vocales acentuadas, la letra 'ñ', la letra 'ç', etc..

identificadores válidos y diferentes.

Cualquier carácter diferente de una letra, un dígito o el subrayado es inválido en un identificador, incluyendo el espacio en blanco. Por ejemplo, *edad media* (con un espacio en medio) son *dos* identificadores (*edad* y *media*), no uno. Cuando un identificador se forma con dos palabras, es costumbre de muchos programadores usar el subrayado para separarlas: *edad_media*; otros programadores utilizan una letra mayúscula para la inicial de la segunda: *edadMedia*. Escoge el estilo que más te guste para nombrar variables, pero permanece fiel al que escojas.

Dado que eres libre de llamar a una variable con el identificador que quieras, hazlo con clase: *escoge siempre nombres que guarden relación con los datos del problema*. Si, por ejemplo, vas a utilizar una variable para almacenar una distancia, llama a la variable *distancia* y evita nombres que no signifiquen nada; de este modo, los programas serán más legibles.

EJERCICIOS

▶ 18 ¿Son válidos los siguientes identificadores?

a) *Identificador*
b) *Indice\dos*
c) *Dos palabras*
d) `__`
e) 12*horas*
f) *hora*12
g) *desviación*
h) *año*
i) *from*
j) *var*!
k) `'var'`
l) *import_from*
m) *UnaVariable*
n) *a*`(`*b*`)`
ñ) 12
o) *uno*`.`*dos*
p) *x*
q) π
r) *área*
s) *area-rect*
t) *x*`_______`1
u) `________`1
v) `_`*x*`_`
w) *x_x*

▶ 19 ¿Qué resulta de ejecutar estas tres líneas?

```
>>> x = 10 ↲
>>> x = x * 10 ↲
>>> x ↲
```

▶ 20 Evalúa el polinomio $x^4 + x^3 + 2x^2 - x$ en $x = 1.1$. Utiliza variables para evitar teclear varias veces el valor de x. (El resultado es 4.1151.)

▶ 21 Evalúa el polinomio $x^4 + x^3 + \frac{1}{2}x^2 - x$ en $x = 10$. Asegúrate de que el resultado sea un número flotante. (El resultado es 11040.0.)

2.4.1. Asignaciones con operador

Fíjate en la sentencia `i = i + 1`: aplica un incremento unitario al contenido de la variable *i*. Incrementar el valor de una variable en una cantidad cualquiera es tan frecuente que existe una forma compacta en Python. El incremento de *i* puede denotarse así:

```
>>> i += 1 ↲
```

(No puede haber espacio alguno entre el `+` y el `=`.) Puedes incrementar una variable con cualquier cantidad, incluso con una que resulte de evaluar una expresión:

```
>>> a = 3 ↲
>>> b = 2 ↲
>>> a += 4 * b ↲
>>> a ↲
11
```

Todos los operadores aritméticos tienen su asignación con operador asociada.

```
z += 2
z *= 2
z /= 2
z -= 2
z %= 2
z **= 2
```

Hemos de decirte que estas formas compactas no aportan nada nuevo... salvo comodidad, así que no te preocupes por tener que aprender tantas cosas. Si te vas a sentir incómodo por tener que tomar decisiones y siempre estás pensando «¿uso ahora la forma normal o la compacta?», es mejor que ignores de momento las formas compactas.

EJERCICIOS

▶ 22 ¿Qué resultará de ejecutar las siguientes sentencias?

```
>>> z = 2 ↲
>>> z += 2 ↲
>>> z += 2 - 2 ↲
>>> z *= 2 ↲
>>> z *= 1 + 1 ↲
>>> z /= 2 ↲
>>> z %= 3 ↲
>>> z /= 3 - 1 ↲
>>> z -= 2 + 1 ↲
>>> z -= 2 ↲
>>> z **= 3 ↲
>>> z ↲
```

2.4.2. Variables no inicializadas

En Python, la primera operación sobre una variable debe ser la asignación de un valor. No se puede usar una variable a la que no se ha asignado previamente un valor:

```
>>> a + 2 ↲
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'a' is not defined
```

Como puedes ver, se genera una excepción `NameError`, es decir, de «error de nombre». El texto explicativo precisa aún más lo sucedido: «`name 'a' is not defined`», es decir, «el nombre *a* no está definido».

La asignación de un valor inicial a una variable se denomina *inicialización* de la variable. Decimos, pues, que en Python no es posible usar variables no inicializadas.