# Variables

## Qué es una variable

### Variables en Matemáticas

El concepto de "variable" proviene de las Matemáticas. En Matemáticas, una variable es un símbolo que forma parte de una **expresión** o de una **fórmula**. Normalmente las variables se representan mediante letras del alfabeto latino (x, y, z, n, i, j, etc.). Dependiendo del **contexto**, las variables significan cosas distintas. Por ejemplo:

#### Álgebra

En el caso del **Álgebra**, una variable representa una cantidad desconocida que se relaciona con otras y que en algunos casos podemos averiguar.

Consideremos por ejemplo la ecuación:

```
x + 3 = 5
```

En este caso, la variable **x** representa una cantidad desconocida pero de la que se sabe que si se le suma 3 se obtiene 5. Resolviendo la ecuación, obtenemos inmediatamente que la variable x estaba representando realmente el número 2.

#### Análisis matemático

En el caso del Análisis matemático, una variable no representa una cantidad determinada, sino que representa todo un conjunto de valores.

Consideremos por ejemplo la ecuación de la recta:

```
y = x + 1
```

En este caso, la variable **x** no representa ningún valor concreto, sino que puede tomar cualquier valor numérico positivo o negativo. Para cada valor de x podemos calcular el valor correspondiente de la variable y. Si interpretamos x e y como las coordenadas de puntos en un plano y dibujamos varios puntos, podríamos ver que todos los puntos se encuentran situados en una misma recta.

### Variables en Programación

En Programación también existe el concepto de "variable", parecido pero no idéntico al concepto matemático.

En Programación, las variables están asociadas a valores concretos. Además, cada lenguaje de programación tiene su forma de implementar el concepto de variable, por lo que lo que se explica a continuación es válido para muchos lenguajes de programación, aunque otros lenguajes de programación permiten otras posibilidades.

- En muchos lenguajes de programación, una variable se puede entender como una especie de caja en la que se puede guardar un valor (por ejemplo, un valor numérico). Esa caja suele corresponder a una **posición de memoria** en la memoria del ordenador.

- Las variables se representan también mediante letras o palabras completas: *x, y, a, b, nombre, apellidos, edad, etc.*
- Cuando en esos lenguajes de programación escribimos, por ejemplo:

```
a = 2
```

lo que estamos pidiendo al programa es que guarde el valor 2 en una "caja" y que a la caja le llame **a**.

En esos lenguajes, el símbolo igualdad `=` hay que entenderlo como una asignación, no como una igualdad matemática. Al escribir una igualdad, le estamos pidiendo al programa que **calcule** lo que hay a la **derecha** de la igualdad y que lo **guarde** en la variable que hay a la **izquierda** de la igualdad.

En esos lenguajes, no estaría permitido escribir ...

```
2 = a
```

... porque 2 no es un nombre válido de variable. Tampoco estaría permitido escribir ...

```
x + 3 = 5
```

... porque en el lado izquierdo no pueden aparecer operadores, sólo el nombre de la variable.

- Una vez hemos guardado un valor en una variable, podemos hacer referencia a él a lo largo del programa. Por ejemplo, el siguiente programa ...

```
a = 2
b = 3
c = a + b
```

- guarda el valor 2 en la variable a
- guarda el valor 3 en la variable b
- coge los valores guardados en las variables a y b (2 y 3), los suma (2+3) y el resultado (5) lo guarda en la variable c.

La información que se guarda en una variable puede ser de muchos tipos:

- números (enteros, decimales, imaginarios, en notación científica, con precisión arbitraria, en base decimal o en otras bases, etc.),
- cadenas de texto (una sola letra o más letras, del juego de caracteres ASCII occidental o del juego de caracteres Unicode, etc),
- conjuntos de números o texto (matrices, listas, tuplas, etc.)
- estructuras más complicadas (punteros, diccionarios, etc.)

Cada tipo de información se codifica de manera distinta al almacenarse en memoria, por lo que existen diferentes tipos de variables para cada tipo de información.

- Algunos lenguajes de programación (C, C++, Java, etc) exigen que antes de utilizar una variable se defina el tipo de información que se va a guardar en esa variable. Otros lenguajes de programación (Python, PHP, etc.) no lo exigen y es el intérprete del lenguaje el que decide el tipo de variable a utilizar en el momento que se guarda la información. Los llenguajes que requieren definir los tipos de las variables se denominan lenguajes tipados o tipificados y los que no se denominan lenguajes no tipados o no tipificados. Python es un

lenguaje no tipado, aunque en Python 3.5 se ha introducido la posibilidad de indicar los tipos de datos de los argumentos de las funciones y de los valores devueltos.

# Variables en Python

En algunos lenguajes de programación, las variables se pueden entender como "cajas" en las que se guardan los datos, pero en Python las variables son "etiquetas" que permiten hacer referencia a los datos (que se guardan en unas "cajas" llamadas **objetos**).

Python es un lenguaje de programación orientado a objetos y su modelo de datos también está basado en objetos. Para cada dato que aparece en un programa, Python **crea un objeto** que lo contiene. Cada objeto tiene:

- un identificador único (un número entero, distinto para cada objeto). El identificador permite a Python referirse al objeto sin ambigüedades.
- un tipo de datos (entero, real, cadena de caracteres, etc.). El tipo de datos permite saber a Python qué operaciones pueden hacerse con el dato.
- un valor (el propio dato).

Así, las variables en Python no guardan los datos, sino que son simples nombres para poder hacer
referencia a esos objetos. Si escribimos la instrucción ...

```
a = 2
```

... Python:

- crea el objeto "2". Ese objeto tendrá un **identificador único** que se asigna en el momento de la creación y se conserva a lo largo del programa. En este caso, el objeto creado será de **tipo número entero** y guardará el **valor 2**.
- asocia el nombre **a** al objeto número entero 2 creado.

Así, al describir la instrucción anterior no habría que decir 'la variable a almacena el número entero 2', sino que habría que decir 'podemos llamar **a** al objeto número entero 2'. La variable a es como una etiqueta que nos permite hacer referencia al objeto "2", más cómoda de recordar y utilizar que el identificador del objeto.

> Durante el curso usaremos expresiones como "guardar un valor en una variable" en vez de "guardar el valor en un objeto y asociar una variable al objeto", pero debemos tener presente lo que ocurre en realidad en Python.

Más adelante se profundiza en la idea de variable de Python.

# Definir una variable

Las variables en Python se crean cuando se definen por primera vez, es decir, cuando se les asigna un valor por primera vez. Para asignar un valor a una variable se utiliza el operador de igualdad (=). A la izquierda de la igualdad se escribe el nombre de la variable y a la derecha el valor que se quiere dar a la variable.

En el ejemplo siguiente se almacena el número real 2,5 en una variable de nombre x (como se ha comentado en el apartado anterior, realmente habría que decir que se crea la etiqueta x para hacer referencia al objeto número real 2.5). Fíjese en que los números decimales se escriben con punto (.) y no con coma (,).

```
>>> x = 2.5
```

La variable se escribe siempre a la izquierda de la igualdad. Si se escribe al revés, Python genera un mensaje de error:

```
>>> 2.5 = x
SyntaxError: can't assign to literal
```

Para que intérprete muestre el valor de una variable, basta con escribir su nombre:

```
>>> x = 2.5
>>> x
2.5
```

Si una variable no se ha definido previamente, escribir su nombre genera un mensaje de error:

```
>>> x = -10
>>> y
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    y
NameError: name 'y' is not defined
```

Una variable puede almacenar números, texto o estructuras más complicadas (que se verán más adelante). Si se va a almacenar texto, el texto debe escribirse entre comillas simples (') o dobles ("), que son equivalentes. A las variables que almacenan texto se les suele llamar **cadenas** (de texto).

```
>>> nombre = "Pepito Conejo"
>>> nombre
'Pepito Conejo'
```

Si no se escriben comillas, Python supone que estamos haciendo referencia a otra variable (que, si no está definida, genera un mensaje de error):

```
>>> nombre = Pepe
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    nombre = Pepe
NameError: name 'Pepe' is not defined
>>> nombre = Pepito Conejo
SyntaxError: invalid syntax
```

# Borrar una variable

La instrucción `del` borra completamente una variable.

```
>>> nombre = "Pepito Conejo"
>>> del nombre
>>> nombre
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    nombre
NameError: name 'nombre' is not defined
```

# Nombres de variables

Aunque no es obligatorio, se recomienda que el nombre de la variable esté relacionado con la información que se almacena en ella, para que sea más fácil entender el programa. Si el programa es trivial o mientras se está escribiendo un programa, esto no parece muy importante, pero si se consulta un programa escrito por otra persona o escrito por uno mismo pero hace tiempo, resultará mucho más fácil entender el programa si los nombres están bien elegidos.

También se acostumbra a utilizar determinados nombres de variables en algunas ocasiones, como se verá más adelante, pero esto tampoco es obligatorio.

El nombre de una variable debe empezar por una letra o por un guión bajo (_) y puede seguir con más letras, números o guiones bajos.

```
>>> _x = 3.8
>>> _x
3.8
>>> x1 = 100
>>> x1
100
>>> fecha_de_nacimiento = "27 de octubre de 1997"
>>> fecha_de_nacimiento
'27 de octubre de 1997'
>>> 1y = 200
  File "<input>", line 1
    1y = 200
     ^
SyntaxError: invalid syntax
```

Los nombres de variables pueden contener cualquier carácter alfabético (las del alfabeto inglés, pero también ñ, ç o vocales acentuadas) y guíones bajos. Lo que no pueden incluir es espacios en blanco

```
>>> fecha de nacimiento = "27 de octubre de 1997"
SyntaxError: invalid syntax
```

> **Nota**: En Python 2 los nombres de variables no podían contener caracteres no ingleses.

Los nombres de las variables pueden contener mayúsculas, pero tenga en cuenta que Python **distingue** entre mayúsculas y minúsculas (en inglés se dice que Python es "case-sensitive").

```
>>> nombre = "Pepito Conejo"
>>> Nombre = "Numa Nigerio"
>>> nomBre = "Fulanito Mengánez"
>>> nombre
'Pepito Conejo'
>>> Nombre
'Numa Nigerio'
>>> nomBre
'Fulanito Mengánez'
```

Cuando el nombre de una variable contiene varias palabras, se aconseja separarlas con guiones bajos para facilitar la legibilidad, aunque también se utiliza la notación **camelCase**, en las que las palabras no se separan pero empiezan con mayúsculas (salvo la primera palabra).

```
>>> fecha_de_nacimiento = "27 de octubre de 1997"
>>> fechaDeNacimiento = "27 de octubre de 1997"
```

Las palabras reservadas del lenguaje (las que intérprete escribe en naranja) están prohibidas como nombres de variables:

```
>>> lambda = 3
SyntaxError: invalid syntax
```

Los nombres de las funciones integradas sí que se pueden utilizar como nombres de variables, pero más vale no hacerlo porque a continuación ya no se puede utilizar la función como tal:

```
>>> print = 3
>>> print("Hola")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print("Hola")
TypeError: 'int' object is not callable
```

Borrando con `del` la variable con nombre de función, se recupera la función.

```
>>> print = 3
>>> print("Hola")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print("Hola")
TypeError: 'int' object is not callable
>>> del print
>>> print("Hola")
Hola
```

# Tipos de variables

En el apartado anterior hay definiciones de algunos de los tipos de variables que hay en Python: números decimales, números enteros y cadenas (una o más letras).

Aunque se definan de forma similar, para Python no es lo mismo un número entero, un número decimal o una cadena ya que, por ejemplo, dos números se pueden multiplicar pero dos cadenas no (curiosamente, una cadena sí que se puede multiplicar por un número).

Por tanto, estas tres definiciones de variables no son equivalentes:

```
>>> fecha = 1997
>>> fecha = 1997.0
>>> fecha = "1997"
>>> fecha = [27, "octubre", 1997]
```

En el primer caso la variable fecha está almacenando un número entero, en el segundo fecha está almacenando un número real, en el tercero fecha está almacenando una cadena de cuatro letras. En el cuarto, fecha está almacenando una **lista** (un tipo de variable que puede contener varios elementos ordenados).

Este ejemplo demuestra también que se puede volver a definir una variable. Python modifica el tipo de la variable automáticamente.

### Función type()

La función predefinida `type(x)` nos permite obtener el tipo de una variable

```
>>> s = 5
>>> type(s)
<class 'int'>
>>> s = 5.1
>>> type(s)
<class 'float'>
>>> s = "5.1"
>>> type(s)
<class 'str'>
>>> s = [27, "octubre", 1997]
>>> type(s)
<class 'list'>
```

## Mostrar el valor de las variables en intérprete

Como hemos visto, para que intérprete muestre el valor de una variable, basta con escribir su nombre.

```
>>> a = 2
>>> a
2
```

También se puede conocer el valor de varias variables a la vez escribiéndolas entre comas intérprete las mostrará entre paréntesis porque Python las considera como un conjunto ordenado llamado tupla), como muestra el siguiente ejemplo:

```
>>> a = b = 2
>>> c = "pepe"
>>> a
2
>>> c, b
('pepe', 2)
```

## Utilizar o modificar variables ya definidas

Una vez se ha definido una variable, se puede utilizar para hacer cálculos o para definir nuevas variables, como muestran los siguientes ejemplos:

```
>>> a = 2
>>> a + 3
5
>>> horas = 5
>>> minutos = 60 * horas
>>> segundos = 60 * minutos
>>> segundos
18000
>>> horas = 1
>>> minutos = 2
>>> segundos = 3
>>> segundos + 60 * minutos + 3600 * horas
3723
```

En caso de utilizar una variable no definida anteriormente, Python genera un mensaje de error.

```
>>> dias = 7 * semanas
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    dias = 7 * semanas
NameError: name 'semanas' is not defined
```

Se puede redefinir una variable a partir de su propio valor. Por ejemplo

```
>>> a = 10
>>> a = a + 5
>>> a
15
```

Esto es posible porque el símbolo de igualdad significa siempre asignación. Lo que hace Python es calcular la expresión situada a la derecha de la igualdad y después guardar el resultado en la variable situada a la izquiereda de la igualdad. En el ejemplo, Python coge el valor almacenado en la variable a (10), le suma 5, y el resultado (15) lo guarda en la misma variable a.

Es importante tener presente que las igualdades son siempre asignaciones, nunca ecuaciones. En el ejemplo, la expresión a = a + 5 no tendría mucho sentido como ecuación (no tendría solución), pero como asignación lo que hace es aumentar en 5 el valor de la variable a.

> Cuando se modifica una variable, el valor anterior se pierde y no se puede recuperar (salvo si se realiza la operación contraria o hay otra variable que conserva el valor anterior).

## Asignaciones aumentadas

Cuando una variabe se modifica a partir de su propio valor, se puede utilizar la denominada "asignación aumentada", una notación compacta que existe también en otros lenguajes de programación.

Por ejemplo:

```
>>> a = 10
>>> a += 5
>>> a
15
```

es equivalente a:

```
>>> a = 10
>>> a = a + 5
>>> a
15
```

En general:

| Asignación aumentada | es equivalente a |
|---|---|
| a += b | a = a + b |
| a -= b | a = a - b |
| a *= b | a = a * b |
| a /= b | a = a / b |
| a **= b | a = a ** b |
| a //= b | a = a // b |
| a %= b | a = a % b |

> Nota: En general, las dos notaciones son completamente equivalentes. Veremos que en función del tipo de las variables hay una pequeña diferencia.

Lo que no se permite en Python son los operadores incremento (++) o decremento (--) que sí existen en otros lenguajes de programación:

```
>>> a = 5
>>> a++
SyntaxError: invalid syntax
```

# Definir y modificar varias variables a la vez

En una misma línea se pueden definir simultáneamente varias variables, con el mismo valor o con valores distintos, como muestra el siguiente ejemplo:

```
>>> a = b = 99
>>> c, d, e = "Mediterráneo", 10, ["pan", "queso"]
```

En el primer caso las dos variables tendrán el mismo valor. En el segundo caso la primera variable tomará el primer valor y así sucesivamente.

Si el número de variables no coincide con el de valores, Python genera un mensaje de error.

```
>>> a, b, c = 1, 2
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a, b, c = 1, 2
ValueError: need more than 2 values to unpack
>>> a, b, c = 1, 2, 3, 4
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a, b, c = 1, 2, 3, 4
ValueError: too many values to unpack
```

Se pueden modificar varias variables en una sola instrucción y la modificación se realiza en un solo paso.

```
>>> a, b = 5, 10
>>> a, b
(5, 10)
>>> a, b = b, a
>>> a, b
(10, 5)
```

Obsérvese que si este procedimiento lo hubiéramos hecho paso a paso, el resultado hubiera sido distinto:

```
>>> a, b = 5, 10
>>> a, b
(5, 10)
>>> a = b
>>> b = a
>>> a, b
(10, 10)
```

El motivo de este resultado se entiende fácilmente mostrando los valores tras cada operación:

```
>>> a, b = 5, 10
>>> a, b
(5, 10)
>>> a = b
>>> a, b
(10, 10)
>>> b = a
>>> a, b
(10, 10)
```

# Referencias

Apuntes generados a partir del curso Introducción a la programación con Python que se distribuye bajo una [Licencia Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional (CC BY-SA 4.0).](