
 No description has been provided for this image

 No description has been provided for this image

Python Basics I: Variables

Bienvenido a tu primer asalto con Python. Empezamos nuestra aventura declarando variables.

Las variables son una manera de etiquetar y conservar los datos del programa para luego poder emplearlos en diferentes sitios.

Contenidos

- [Variables numéricas](#)
- [Cadenas de texto](#)
- [Escribir mal el nombre](#)
- [Cerrar bien los strings](#)
- [Prohibido espacios en los nombres de las variables](#)
- [Números en el nombre de la variable](#)
- [Sensible a mayúsculas](#)
- [Palabras reservadas](#)
- [Resumen variables](#)

Variables numéricas

[al índice](#)

```
In [1]: ingresos = 1000  
gastos = 400
```

Esto es una asignación. A la palabra *ingresos*, le asignamos un valor mediante `=`. Si queremos ver el valor de la variable, simplemente escribimos su nombre

en una celda.

```
In [2]: ingresos
```

```
Out[2]: 1000
```

```
In [3]: ingresos  
gastos
```

```
Out[3]: 400
```

```
In [4]: print(ingresos)  
print(gastos)
```

```
1000
```

```
400
```

```
In [7]: margen = ingresos - gastos
```

```
In [8]: margen
```

```
Out[8]: 600
```

```
In [9]: print(margen)
```

```
600
```

Cadenas de texto

[al índice](#)

Las cadenas de texto se declaran con comillas simples o dobles

```
In [10]: ingresos_text = "Los ingresos han sido altos"  
ingresos_alt = 'Los ingresos han sido altos'  
print(ingresos_text)  
print(ingresos_alt)
```

```
Los ingresos han sido altos
```

```
Los ingresos han sido altos
```

```
In [11]: ingresos_text = "Los 'ingresos' han sido altos"
```

```
In [12]: ingresos_text
```

```
Out[12]: "Los 'ingresos' han sido altos"
```

```
In [14]: ingresos=1000  
ingresos = 1000  
ingresos text = "Los ingresos has sido altos, pero esto da error"
```

```
Cell In[14], line 3
    ingresos text = "Los ingresos has sido altos, pero esto da error"
    ^
SyntaxError: invalid syntax
```

```
In [13]: ingresos_text = "Los "iIngresos" has sido altos"
```

```
Cell In[13], line 1
    ingresos_text = "Los "ingresos" has sido altos"
    ^
SyntaxError: invalid syntax
```

Python es un lenguaje dinámico por lo que siempre podremos actualizar los valores de las variables. **Se recomienda usar nombres descriptivos para declarar las variables, pero no excesivamente largos.** Así evitamos sobreescribirlas sin querer.

Por otro lado, cuidado con los caracteres ele y uno (`1` vs `1`), así como con cero y o (`0` vs `0`). Se suelen confundir.

fatal vs fata1

clarO vs clar0

Reasignamos valor a gastos

```
In [15]: gastos = 200
```

```
In [16]: print(gastos)
```

200

Ahora la variable gastos vale 200. Si la última línea de una celda es una variable, su valor será el *output* de la celda: 200.

Vale, ¿y de qué nos sirve guardar variables?

Podremos usar estos datos posteriormente, en otro lugar del programa. Por ejemplo, si ahora en una celda nueva queremos obtener el beneficio, simplemente restamos los nombres de las variables

```
In [17]: gastos = 100
```

```
In [18]: beneficios = ingresos - gastos
```

```
In [19]: print(beneficios)
```

900



No
description
has been
provided for
this image

ERRORES en variables

Escribir mal el nombre

[al índice](#)

Un error típico cuando declaramos variables es **escribir su nombre mal, o llamar después a la variable de forma errónea**. En tal caso, aparece un

```
NameError: name 'variable_que_no_existe' is not defined
```

In [20]:

```
gstos
```

```
-----  
---  
NameError                                Traceback (most recent call la  
st)  
Cell In[20], line 1  
----> 1 gstos  
  
NameError: name 'gstos' is not defined
```

Fíjate que te indica la línea donde se produce el error, el tipo de error (`NameError`), y una breve descripción del error.

Cerrar bien los strings

[al índice](#)

Cuidado también cuando definamos una cadena de texto, y se nos olvide cerrarla con sus correspondientes comillas. Nos dará un `SyntaxError: EOL while scanning string literal` (End Of Line)

In [21]:

```
texto = "Error sin comillas
```

```
Cell In[21], line 1  
    texto = "Error sin comillas  
              ^  
SyntaxError: EOL while scanning string literal
```

In []:

Prohibido espacios en los nombres de las variables

[al índice](#)

También dará error si tenemos un espacio en la declaración de la variable. Se recomienda minúsculas y guiones bajos para simular los espacios.

Los espacios que encuentres alrededor del igual se pueden usar perfectamente. Es pura estética a la hora de leer el código.

In []:

In []:

Números en el nombre de la variable

[al índice](#)

Ojo con los números cuando declaremos variables. En ocasiones es determinante describir nuestra variable con algún número. En tal caso, ponlo siempre al final del nombre de la variable, ya que sino saltará un error.

In [22]: 2009_ingresos

```
Cell In[22], line 1
    2009_ingresos
        ^
SyntaxError: invalid decimal literal
```

In [23]: ingresos_2009 = 250000

Sensible a mayúsculas

[al índice](#)

Mucho cuidado con las mayúsculas y minúsculas porque Python no las ignora. Si todas las letras de una variable están en mayúsculas, tendremos que usarla en mayúsculas, sino dará un error de que no encuentra la variable.

In [24]: print(Ingresos_2009)

```
-----
---
NameError                                Traceback (most recent call last)
Cell In[24], line 1
----> 1 print(Ingresos_2009)

NameError: name 'Ingresos_2009' is not defined
```

In []:

Palabras reservadas

[al índice](#)

En Python, como en otros lenguajes, hay una serie de palabras reservadas que tienen un significado para el intérprete de Python y por lo tanto no podemos usar para ponerle nombre a nuestras variables.

Por ejemplo, `def` se usa para definir funciones en Python (lo veremos en otros notebooks), por lo que no podemos emplear `def` para nombrar a nuestras variables

In [25]: `def = 10`

```
Cell In[25], line 1
    def = 10
      ^
SyntaxError: invalid syntax
```

Consulta la lista de palabras reservadas de Python

In [26]: `import keyword`
`print(keyword.kwlist)`

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

In [27]: `help("keyword")`

Help on module keyword:

NAME

keyword - Keywords (from "Grammar/Grammar")

MODULE REFERENCE

<https://docs.python.org/3.8/library/keyword>

The following documentation is automatically generated from the Python source files. It may be incomplete, incorrect or include features that are considered implementation detail and may vary between Python implementations. When in doubt, consult the module reference at the location listed above.

DESCRIPTION

This file is automatically generated; please don't muck it up!

To update the symbols in this file, 'cd' to the top directory of the python source tree and run:

```
python3 -m Parser.pgen.keywordgen Grammar/Grammar
Grammar/Tokens                               Lib/keyword.py
```

Alternatively, you can run 'make regen-keyword'.

FUNCTIONS

iskeyword = __contains__(...) method of builtins.frozenset instance
x.__contains__(y) <==> y in x.

DATA

```
__all__ = ['iskeyword', 'kwlist']
kwlist = ['False', 'None', 'True', 'and', 'as', 'assert', 'async',
'aw...
```

FILE

/usr/lib/python3.8/keyword.py

Resumen variables

[al índice](#)

En resumen:

- Usar minúsculas. Sensible a mayúsculas/minúsculas
- No usar espacios
- No usar palabras reservadas
- No usar números al principio de la variable
- Cuidado con los caracteres `1`, `l`, `0`, `o`

In []:

In []:

In []: