Python

[Ir a la navegación](https://es.wikipedia.org/wiki/Python#mw-head)[Ir a la búsqueda](https://es.wikipedia.org/wiki/Python#p-search)

*Este artículo trata sobre el lenguaje de programación. Para el grupo de humoristas, véase*[*Monty Python*](https://es.wikipedia.org/wiki/Monty_Python)*.*

*Para el revólver, véase*[*Colt Python*](https://es.wikipedia.org/wiki/Colt_Python)*.*

*Para otros usos de este término, véase*[*Pitón*](https://es.wikipedia.org/wiki/Pit%C3%B3n)*.*

|  |  |
| --- | --- |
| **Python** | |
| [Python-logo-notext.svg](https://commons.wikimedia.org/wiki/File:Python-logo-notext.svg) | |
| **Desarrollador(es)** | |
| [Python Software Foundation](https://es.wikipedia.org/wiki/Python_Software_Foundation) [*http://www.python.org/*](http://www.python.org/) | |
| **Información general** | |
| [**Extensiones comunes**](https://es.wikipedia.org/wiki/Extensi%C3%B3n_de_archivo) | .py, .pyc, .pyd, .pyo, .pyw |
| [**Paradigma**](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) | [Multiparadigma](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_multiparadigma): [orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), [imperativo](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa), [funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional), [reflexivo](https://es.wikipedia.org/wiki/Reflexi%C3%B3n_(inform%C3%A1tica)) |
| **Apareció en** | 1991 |
| **Diseñado por** | [Guido van Rossum](https://es.wikipedia.org/wiki/Guido_van_Rossum) |
| **Última versión estable** | 3.7.0[1](https://es.wikipedia.org/wiki/Python#cite_note-1)​ / 2.7.15[2](https://es.wikipedia.org/wiki/Python#cite_note-2)​ ([27 de junio](https://es.wikipedia.org/wiki/27_de_junio)de [2018](https://es.wikipedia.org/wiki/2018) (1 mes y 26 días)) |
| **Última versión en pruebas** | ([6 de diciembre](https://es.wikipedia.org/wiki/6_de_diciembre) de [2017](https://es.wikipedia.org/wiki/2017) (8 meses y 17 días)) |
| [**Sistema de tipos**](https://es.wikipedia.org/wiki/Sistema_de_tipos) | [Fuertemente tipado](https://es.wikipedia.org/wiki/Tipado_fuerte), [dinámico](https://es.wikipedia.org/wiki/Tipado_din%C3%A1mico) |
| **Implementaciones** | [CPython](https://es.wikipedia.org/wiki/CPython), [IronPython](https://es.wikipedia.org/wiki/IronPython), [Jython](https://es.wikipedia.org/wiki/Jython), Python for S60, [PyPy](https://es.wikipedia.org/wiki/PyPy), [Pygame](https://es.wikipedia.org/wiki/Pygame), [ActivePython](https://es.wikipedia.org/w/index.php?title=ActivePython&action=edit&redlink=1), Unladen Swallow |
| **Dialectos** | [Stackless Python](https://es.wikipedia.org/w/index.php?title=Stackless_Python&action=edit&redlink=1), [RPython](https://es.wikipedia.org/w/index.php?title=RPython&action=edit&redlink=1) |
| **Influido por** | [ABC](https://es.wikipedia.org/wiki/ABC_(lenguaje_de_programaci%C3%B3n)), ALGOL 68, [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)), [Haskell](https://es.wikipedia.org/wiki/Haskell), [Icon](https://es.wikipedia.org/wiki/Icon), [Lisp](https://es.wikipedia.org/wiki/Lisp), [Modula-3](https://es.wikipedia.org/wiki/Modula-3), [Perl](https://es.wikipedia.org/wiki/Perl), [Smalltalk](https://es.wikipedia.org/wiki/Smalltalk), [Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)) |
| **Ha influido a** | [Boo](https://es.wikipedia.org/wiki/Boo_(lenguaje_de_programaci%C3%B3n)), Cobra, [D](https://es.wikipedia.org/wiki/D_(lenguaje_de_programaci%C3%B3n)), [Falcon](https://es.wikipedia.org/wiki/Falcon_(lenguaje_de_programaci%C3%B3n)), [Genie](https://es.wikipedia.org/wiki/Genie_(lenguaje_de_programaci%C3%B3n)), [Groovy](https://es.wikipedia.org/wiki/Groovy_(lenguaje_de_programaci%C3%B3n)), [Ruby](https://es.wikipedia.org/wiki/Ruby), [JavaScript](https://es.wikipedia.org/wiki/JavaScript), [Cython](https://es.wikipedia.org/wiki/Cython), [Go](https://es.wikipedia.org/wiki/Go_(lenguaje_de_programaci%C3%B3n)) |
| [**Sistema operativo**](https://es.wikipedia.org/wiki/Sistema_operativo) | [Multiplataforma](https://es.wikipedia.org/wiki/Multiplataforma) |
| [**Licencia**](https://es.wikipedia.org/wiki/Licencia_de_software) | [Python Software Foundation License](https://es.wikipedia.org/wiki/Python_Software_Foundation_License) |
| [[editar datos en Wikidata](https://www.wikidata.org/wiki/Q28865)] | |

**Python** es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) [interpretado](https://es.wikipedia.org/wiki/Int%C3%A9rprete_(inform%C3%A1tica)) cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación [multiparadigma](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_multiparadigma), ya que soporta [orientación a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), [programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa) y, en menor medida, [programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional). Es un [lenguaje interpretado](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_interpretado), usa [tipado dinámico](https://es.wikipedia.org/wiki/Tipado_din%C3%A1mico) y es [multiplataforma](https://es.wikipedia.org/wiki/Multiplataforma).

Es administrado por la [Python Software Foundation](https://es.wikipedia.org/wiki/Python_Software_Foundation). Posee una licencia de [código abierto](https://es.wikipedia.org/wiki/C%C3%B3digo_abierto), denominada [Python Software Foundation License](https://es.wikipedia.org/wiki/Python_Software_Foundation_License),[3](https://es.wikipedia.org/wiki/Python#cite_note-lib-history-3)​ que es compatible con la [Licencia pública general de GNU](https://es.wikipedia.org/wiki/GNU_General_Public_License) a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.



**Índice**

* [1Historia](https://es.wikipedia.org/wiki/Python#Historia)
* [2Características y paradigmas](https://es.wikipedia.org/wiki/Python#Caracter%C3%ADsticas_y_paradigmas)
* [3Filosofía](https://es.wikipedia.org/wiki/Python#Filosof%C3%ADa)
* [4Modo interactivo](https://es.wikipedia.org/wiki/Python#Modo_interactivo)
* [5Elementos del lenguaje](https://es.wikipedia.org/wiki/Python#Elementos_del_lenguaje)
  + [5.1Comentarios](https://es.wikipedia.org/wiki/Python#Comentarios)
  + [5.2Variables](https://es.wikipedia.org/wiki/Python#Variables)
  + [5.3Tipos de datos](https://es.wikipedia.org/wiki/Python#Tipos_de_datos)
  + [5.4Condicionales](https://es.wikipedia.org/wiki/Python#Condicionales)
  + [5.5Bucle for](https://es.wikipedia.org/wiki/Python#Bucle_for)
  + [5.6Bucle while](https://es.wikipedia.org/wiki/Python#Bucle_while)
  + [5.7Listas y Tuplas](https://es.wikipedia.org/wiki/Python#Listas_y_Tuplas)
  + [5.8Diccionarios](https://es.wikipedia.org/wiki/Python#Diccionarios)
  + [5.9Sentencia Switch Case](https://es.wikipedia.org/wiki/Python#Sentencia_Switch_Case)
    - [5.9.1Usando if, elif, else](https://es.wikipedia.org/wiki/Python#Usando_if,_elif,_else)
    - [5.9.2Usando diccionario](https://es.wikipedia.org/wiki/Python#Usando_diccionario)
  + [5.10Conjuntos](https://es.wikipedia.org/wiki/Python#Conjuntos)
  + [5.11Listas por comprensión](https://es.wikipedia.org/wiki/Python#Listas_por_comprensi%C3%B3n)
  + [5.12Funciones](https://es.wikipedia.org/wiki/Python#Funciones)
  + [5.13Clases](https://es.wikipedia.org/wiki/Python#Clases)
  + [5.14Módulos](https://es.wikipedia.org/wiki/Python#M%C3%B3dulos)
  + [5.15Interfaz al sistema operativo](https://es.wikipedia.org/wiki/Python#Interfaz_al_sistema_operativo)
  + [5.16Comodines de archivos](https://es.wikipedia.org/wiki/Python#Comodines_de_archivos)
  + [5.17Argumentos de línea de órdenes](https://es.wikipedia.org/wiki/Python#Argumentos_de_l%C3%ADnea_de_%C3%B3rdenes)
  + [5.18Matemática](https://es.wikipedia.org/wiki/Python#Matem%C3%A1tica)
  + [5.19Fechas y Tiempos](https://es.wikipedia.org/wiki/Python#Fechas_y_Tiempos)
* [6Sistema de objetos](https://es.wikipedia.org/wiki/Python#Sistema_de_objetos)
* [7Biblioteca estándar](https://es.wikipedia.org/wiki/Python#Biblioteca_est%C3%A1ndar)
* [8Implementaciones](https://es.wikipedia.org/wiki/Python#Implementaciones)
* [9Diferencias entre Python 2.x y Python 3.x](https://es.wikipedia.org/wiki/Python#Diferencias_entre_Python_2.x_y_Python_3.x)
* [10Véase también](https://es.wikipedia.org/wiki/Python#V%C3%A9ase_tambi%C3%A9n)
* [11Referencias](https://es.wikipedia.org/wiki/Python#Referencias)
* [12Bibliografía](https://es.wikipedia.org/wiki/Python#Bibliograf%C3%ADa)
* [13Enlaces externos](https://es.wikipedia.org/wiki/Python#Enlaces_externos)

Historia[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=1)]

*Artículo principal:*[Historia de Python](https://es.wikipedia.org/wiki/Historia_de_Python)

[](https://commons.wikimedia.org/wiki/File:Guido_van_Rossum_OSCON_2006.jpg)

[Guido van Rossum](https://es.wikipedia.org/wiki/Guido_van_Rossum), creador de **Python**, en la convención OSCON 2006

Python fue creado a finales de los ochenta[4](https://es.wikipedia.org/wiki/Python#cite_note-venners-interview-pt-1-4)​ por [Guido van Rossum](https://es.wikipedia.org/wiki/Guido_van_Rossum) en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los [Países Bajos](https://es.wikipedia.org/wiki/Pa%C3%ADses_Bajos), como un sucesor del [lenguaje de programación ABC](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_ABC), capaz de [manejar excepciones](https://es.wikipedia.org/wiki/Manejo_de_excepciones) e interactuar con el [sistema operativo Amoeba](https://es.wikipedia.org/wiki/Amoeba_(Inform%C3%A1tica)).[5](https://es.wikipedia.org/wiki/Python#cite_note-whycreated-5)​

El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos [Monty Python](https://es.wikipedia.org/wiki/Monty_Python).[6](https://es.wikipedia.org/wiki/Python#cite_note-6)​

Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como [*Benevolente Dictador Vitalicio*](https://es.wikipedia.org/wiki/Benevolent_Dictator_for_Life) (en inglés: *Benevolent Dictator for Life*, BDFL); sin embargo el 12 de julio de 2018 declinó de dicha situación de honor sin dejar un sucesor o sucesora y con una declaración altisonante:[7](https://es.wikipedia.org/wiki/Python#cite_note-7)​

Entonces, ¿qué van a hacer todos ustedes? ¿Crear una democracia? ¿Anarquía? ¿Una dictadura? ¿Una federación?

Guido van Rossum[8](https://es.wikipedia.org/wiki/Python#cite_note-8)​

En 1991, van Rossum publicó el código de la versión 0.9.0 en [alt.sources](news:alt.sources).[9](https://es.wikipedia.org/wiki/Python#cite_note-9)​ En esta etapa del desarrollo ya estaban presentes clases con [herencia](https://es.wikipedia.org/wiki/Herencia_(inform%C3%A1tica)), manejo de excepciones, [funciones](https://es.wikipedia.org/wiki/Subrutina) y los tipos modulares, como: str, list, dict, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de [Modula-3](https://es.wikipedia.org/wiki/Modula-3); van Rossum describe el módulo como “una de las mayores unidades de programación de Python”.[4](https://es.wikipedia.org/wiki/Python#cite_note-venners-interview-pt-1-4)​ El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula else.[5](https://es.wikipedia.org/wiki/Python#cite_note-whycreated-5)​ En el año 1994 se formó [comp.lang.python](news:comp.lang.python), el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios de este lenguaje.

Python alcanzó la versión 1.0 en enero de 1994. Una característica de este lanzamiento fueron las herramientas de la [programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional): [lambda](https://es.wikipedia.org/wiki/C%C3%A1lculo_lambda), reduce, filter y map. Van Rossum explicó que “hace 12 años, Python adquirió lambda, reduce(), filter() y map(), cortesía de un hacker informático de [Lisp](https://es.wikipedia.org/wiki/Lisp) que las extrañaba y que envió parches”.[10](https://es.wikipedia.org/wiki/Python#cite_note-10)​ El donante fue Amrit Prem; no se hace ninguna mención específica de cualquier herencia de Lisp en las notas de lanzamiento.

La última versión liberada proveniente de CWI fue Python 1.2. En 1995, van Rossum continuó su trabajo en Python en la [Corporation for National Research Initiatives](https://en.wikipedia.org/wiki/Corporation_for_National_Research_Initiatives) (CNRI) en Reston, [Virginia](https://es.wikipedia.org/wiki/Virginia), donde lanzó varias versiones del [software](https://es.wikipedia.org/wiki/Software).

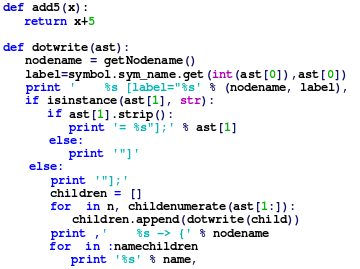
Durante su estancia en CNRI, van Rossum lanzó la iniciativa *Computer Programming for Everybody* (CP4E), con el fin de hacer la programación más accesible a más gente, con un nivel de 'alfabetización' básico en lenguajes de programación, similar a la alfabetización básica en inglés y habilidades matemáticas necesarias por muchos trabajadores. Python tuvo un papel crucial en este proceso: debido a su orientación hacia una sintaxis limpia, ya era idóneo, y las metas de CP4E presentaban similitudes con su predecesor, ABC. El proyecto fue patrocinado por [DARPA](https://es.wikipedia.org/wiki/DARPA).[11](https://es.wikipedia.org/wiki/Python#cite_note-11)​ En el año 2007, el proyecto CP4E está inactivo, y mientras Python intenta ser fácil de aprender y no muy arcano en su sintaxis y semántica, alcanzando a los no-programadores, no es una preocupación activa.[12](https://es.wikipedia.org/wiki/Python#cite_note-12)​

En el año 2000, el equipo principal de desarrolladores de Python se cambió a [BeOpen.com](https://es.wikipedia.org/w/index.php?title=BeOpen.com&action=edit&redlink=1) para formar el equipo BeOpen [PythonLabs](https://es.wikipedia.org/w/index.php?title=PythonLabs&action=edit&redlink=1). CNRI pidió que la versión 1.6 fuera pública, continuando su desarrollo hasta que el equipo de desarrollo abandonó CNRI; su programa de lanzamiento y el de la versión 2.0 tenían una significativa cantidad de traslapo.[13](https://es.wikipedia.org/wiki/Python#cite_note-newin-2.0-13)​ Python 2.0 fue el primer y único lanzamiento de BeOpen.com. Después que Python 2.0 fuera publicado por BeOpen.com, Guido van Rossum y los otros desarrolladores de PythonLabs se unieron en [Digital Creations](https://es.wikipedia.org/w/index.php?title=Digital_Creations&action=edit&redlink=1).

Python 2.0 tomó una característica mayor del lenguaje de [programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional) [Haskell](https://es.wikipedia.org/wiki/Haskell): listas por comprensión. La sintaxis de Python para esta construcción es muy similar a la de Haskell, salvo por la preferencia de los caracteres de puntuación en Haskell, y la preferencia de Python por palabras claves alfabéticas. Python 2.0 introdujo además un sistema de [recolección de basura](https://es.wikipedia.org/wiki/Recolecci%C3%B3n_de_basura) capaz de recolectar referencias cíclicas.[13](https://es.wikipedia.org/wiki/Python#cite_note-newin-2.0-13)​

Posterior a este doble lanzamiento, y después que van Rossum dejó CNRI para trabajar con desarrolladores de software comercial, quedó claro que la opción de usar Python con software disponible bajo [GNU GPL](https://es.wikipedia.org/wiki/GNU_General_Public_License) era muy deseable. La licencia usada entonces, la [Python License](https://es.wikipedia.org/wiki/Python_License), incluía una cláusula estipulando que la licencia estaba gobernada por el estado de [Virginia](https://es.wikipedia.org/wiki/Virginia), por lo que, bajo la óptica de los abogados de [Free Software Foundation](https://es.wikipedia.org/wiki/Free_Software_Foundation) (FSF), se hacía incompatible con GPL. CNRI y FSF se relacionaron para cambiar la licencia de software libre de Python para hacerla compatible con GPL. En el año 2001, van Rossum fue premiado con [FSF Award for the Advancement of Free Software](https://es.wikipedia.org/wiki/FSF_Award_for_the_Advancement_of_Free_Software).

Python 1.6.1 es esencialmente el mismo que Python 1.6, con unos pocos arreglos de [bugs](https://es.wikipedia.org/wiki/Error_de_software), y con una nueva licencia compatible con GPL.[3](https://es.wikipedia.org/wiki/Python#cite_note-lib-history-3)​

[](https://commons.wikimedia.org/wiki/File:Python_add5_syntax.svg)

Código Python con [coloreado de sintaxis](https://es.wikipedia.org/wiki/Coloreado_de_sintaxis).

Python 2.1 fue un trabajo derivado de Python 1.6.1, así como también de Python 2.0. Su licencia fue renombrada a: [Python Software Foundation License](https://es.wikipedia.org/wiki/Python_Software_Foundation_License). Todo el código, documentación y especificaciones añadidas, desde la fecha del lanzamiento de la versión alfa de Python 2.1, tiene como dueño a [Python Software Foundation](https://es.wikipedia.org/wiki/Python_Software_Foundation) (PSF), una organización sin ánimo de lucro fundada en el año 2001, tomando como modelo la [Apache Software Foundation](https://es.wikipedia.org/wiki/Apache_Software_Foundation).[3](https://es.wikipedia.org/wiki/Python#cite_note-lib-history-3)​ Incluido en este lanzamiento fue una implementación del scoping más parecida a las reglas de [static scoping](https://es.wikipedia.org/w/index.php?title=Static_scoping&action=edit&redlink=1) (del cual [Scheme](https://es.wikipedia.org/wiki/Scheme) es el originador).[14](https://es.wikipedia.org/wiki/Python#cite_note-14)​

Una innovación mayor en Python 2.2 fue la unificación de los tipos en Python (tipos escritos en C), y clases (tipos escritos en Python) dentro de una jerarquía. Esa unificación logró un modelo de objetos de Python puro y consistente.[15](https://es.wikipedia.org/wiki/Python#cite_note-15)​ También fueron agregados los [generadores](https://es.wikipedia.org/wiki/Generador_(inform%C3%A1tica)) que fueron inspirados por el lenguaje [Icon](https://es.wikipedia.org/wiki/Icon).[16](https://es.wikipedia.org/wiki/Python#cite_note-16)​

Las adiciones a la biblioteca estándar de Python y las decisiones sintácticas fueron influenciadas fuertemente por [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) en algunos casos: el package logging,[17](https://es.wikipedia.org/wiki/Python#cite_note-17)​ introducido en la versión 2.3, está basado en [log4j](https://es.wikipedia.org/wiki/Log4j); el parser [SAX](https://es.wikipedia.org/wiki/Simple_API_for_XML), introducido en 2.0; el package threading,[18](https://es.wikipedia.org/wiki/Python#cite_note-18)​ cuya clase *Thread* expone un subconjunto de la interfaz de la clase homónima en Java.

Características y paradigmas[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=2)]

Python es un lenguaje de programación [multiparadigma](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_multiparadigma). Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: [programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), [programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa) y [programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional). Otros paradigmas están soportados mediante el uso de extensiones.

Python usa [tipado dinámico](https://es.wikipedia.org/wiki/Tipado_din%C3%A1mico) y [conteo de referencias](https://es.wikipedia.org/wiki/Conteo_de_referencias) para la administración de memoria.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) o [C++](https://es.wikipedia.org/wiki/C%2B%2B). Python puede incluirse en aplicaciones que necesitan una interfaz programable.

Aunque la programación en Python podría considerarse en algunas situaciones hostil a la programación funcional tradicional del [Lisp](https://es.wikipedia.org/wiki/Lisp), existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser [Scheme](https://es.wikipedia.org/wiki/Scheme).

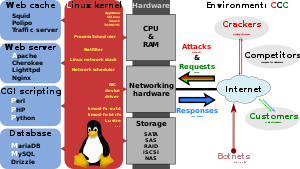
Filosofía[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=3)]

Los usuarios de Python se refieren a menudo a la **Filosofía Python** que es bastante análoga a la filosofía de [Unix](https://es.wikipedia.org/wiki/Unix). El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el código opaco u ofuscado es bautizado como "no pythonico" ("unpythonic" en inglés). Estos principios fueron descritos por el desarrollador de Python [Tim Peters](https://es.wikipedia.org/w/index.php?title=Tim_Peters&action=edit&redlink=1) en [El Zen de Python](https://es.wikipedia.org/wiki/Zen_de_Python)

* Bello es mejor que feo.
* Explícito es mejor que implícito.
* Simple es mejor que complejo.
* Complejo es mejor que complicado.
* Plano es mejor que anidado.
* Disperso es mejor que denso.
* La legibilidad cuenta.
* Los casos especiales no son tan especiales como para quebrantar las reglas.
* Lo práctico gana a lo puro.
* Los errores nunca deberían dejarse pasar silenciosamente.
* A menos que hayan sido silenciados explícitamente.
* Frente a la ambigüedad, rechaza la tentación de adivinar.
* Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
* Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.[19](https://es.wikipedia.org/wiki/Python#cite_note-19)​
* Ahora es mejor que nunca.
* Aunque *nunca* es a menudo mejor que *ya mismo*.
* Si la implementación es difícil de explicar, es una mala idea.
* Si la implementación es fácil de explicar, puede que sea una buena idea.
* Los espacios de nombres (*namespaces*) son una gran idea ¡Hagamos más de esas cosas!

[Tim Peters](https://es.wikipedia.org/w/index.php?title=Tim_Peters&action=edit&redlink=1), [El Zen de Python](https://es.wikipedia.org/wiki/Zen_de_Python)

Desde la versión 2.1.2, Python incluye estos puntos (en su versión original en inglés) como un [huevo de pascua](https://es.wikipedia.org/wiki/Huevo_de_pascua_(virtual)) que se muestra al ejecutar import this.[20](https://es.wikipedia.org/wiki/Python#cite_note-20)​

[](https://commons.wikimedia.org/wiki/File:LAMP_software_bundle.svg)

La [LAMP](https://es.wikipedia.org/wiki/LAMP) comprende Python (aquí con [Squid](https://es.wikipedia.org/wiki/Squid_(programa)))

Modo interactivo[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=4)]

El intérprete de Python estándar incluye un *modo interactivo* en el cual se escriben las instrucciones en una especie de [intérprete de comandos](https://es.wikipedia.org/wiki/Int%C3%A9rprete_de_comandos): las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente, lo que da la posibilidad de probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa. Esto resulta útil tanto para las personas que se están familiarizando con el lenguaje como para los programadores más avanzados.

Existen otros programas, tales como [IDLE](http://docs.python.org/library/idle.html), [bpython](http://bpython-interpreter.org/) o [IPython](https://es.wikipedia.org/wiki/IPython),[21](https://es.wikipedia.org/wiki/Python#cite_note-21)​ que añaden funcionalidades extra al modo interactivo, como el autocompletado de código y el [coloreado de la sintaxis](https://es.wikipedia.org/wiki/Coloreado_de_sintaxis) del lenguaje.

Ejemplo del modo interactivo:

>>> 1 + 1

2

>>> a = range(10)

>>> **print**( list(a) )

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Elementos del lenguaje[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=5)]

Python fue diseñado para ser leído con facilidad. Una de sus características es el uso de palabras donde otros lenguajes utilizarían símbolos. Por ejemplo, los operadores lógicos !, || y && en Python se escriben not, or y and, respectivamente. Curiosamente el lenguaje [Pascal](https://es.wikipedia.org/wiki/Pascal_(lenguaje_de_programaci%C3%B3n)) es junto con [COBOL](https://es.wikipedia.org/wiki/COBOL) uno de los lenguajes con muy clara sintaxis y ambos son de la década del 70. La idea del código claro y legible no es algo nuevo.

El contenido de los bloques de código (bucles, funciones, clases, etc.) es delimitado mediante espacios o tabuladores, conocidos como [indentación](https://es.wikipedia.org/wiki/Indentaci%C3%B3n), antes de cada línea de órdenes pertenecientes al bloque.[22](https://es.wikipedia.org/wiki/Python#cite_note-22)​ Python se diferencia así de otros lenguajes de programación que mantienen como costumbre declarar los bloques mediante un conjunto de caracteres, normalmente entre llaves {}.[23](https://es.wikipedia.org/wiki/Python#cite_note-23)​[24](https://es.wikipedia.org/wiki/Python#cite_note-24)​ Se pueden utilizar tanto espacios como tabuladores para indentar el código, pero se recomienda no mezclarlos.[25](https://es.wikipedia.org/wiki/Python#cite_note-25)​

|  |  |
| --- | --- |
| [**Función factorial**](https://es.wikipedia.org/wiki/Factorial)**en**[**C**](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C)**(indentación opcional)** | [**Función factorial**](https://es.wikipedia.org/wiki/Factorial)**en Python (indentación obligatoria)** |
| int factorial(int x)  {  **if** (x < 0 || x % 1 != 0) {  printf("x debe ser un numero entero mayor o igual a 0");  **return** -1; *//Error*  }  **if** (x == 0) {  **return** 1;  }  **return** x \* factorial(x - 1);  } | **def** factorial(x):  **assert** x >= 0 **and** x % 1 == 0, "x debe ser un entero mayor o igual a 0."  **if** x == 0:  **return** 1  **else**:  **return** x \* factorial(x - 1) |

Debido al significado sintáctico de la indentación, cada instrucción debe estar contenida en una sola línea. No obstante, si por legibilidad se quiere dividir la instrucción en varias líneas, añadiendo una barra invertida \ al final de una línea, se indica que la instrucción continúa en la siguiente.

Estas instrucciones son equivalentes:

|  |  |
| --- | --- |
| lista=['valor 1','valor 2','valor 3']  cadena='Esto es una cadena bastante larga' | lista=['valor 1','valor 2' \  ,'valor 3']  cadena='Esto es una cadena ' \  'bastante larga' |

**Comentarios**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=6)]

Los *comentarios* se pueden poner de dos formas. La primera y más apropiada para comentarios largos es utilizando la notación ''' comentario ''', tres apóstrofos de apertura y tres de cierre. La segunda notación utiliza el símbolo #, y se extienden hasta el final de la línea.

El intérprete no tiene en cuenta los *comentarios*, lo cual es útil si deseamos poner información adicional en el código. Por ejemplo, una explicación sobre el comportamiento de una sección del programa.

*'''*

*Comentario más largo en una línea en Python*

*'''*

**print** "Hola mundo" *# También es posible añadir un comentario al final de una línea de código*

**Variables**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=7)]

Las *variables* se definen de forma [dinámica](https://es.wikipedia.org/wiki/Tipado_din%C3%A1mico), lo que significa que no se tiene que especificar cuál es su tipo de antemano y puede tomar distintos valores en otro momento, incluso de un tipo diferente al que tenía previamente. Se usa el símbolo = para asignar valores.

x = 1

x = "texto" *# Esto es posible porque los tipos son asignados dinámicamente*

**Tipos de datos**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=8)]

Los *tipos de datos* se pueden resumir en esta tabla:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo** | **Clase** | **Notas** | **Ejemplo** |
| str | Cadena | Inmutable | 'Cadena' |
| unicode | Cadena | Versión [Unicode](https://es.wikipedia.org/wiki/Unicode) de str | u'Cadena' |
| list | Secuencia | Mutable, puede contener objetos de diversos tipos | [4.0, 'Cadena', True] |
| tuple | Secuencia | Inmutable, puede contener objetos de diversos tipos | (4.0, 'Cadena', True) |
| set | Conjunto | Mutable, sin orden, no contiene duplicados | set([4.0, 'Cadena', True]) |
| frozenset | Conjunto | Inmutable, sin orden, no contiene duplicados | frozenset([4.0, 'Cadena', True]) |
| dict | Mapping | Grupo de pares clave:valor | {'key1': 1.0, 'key2': False} |
| int | Número entero | Precisión fija, convertido en *long* en caso de overflow. | 42 |
| long | Número entero | Precisión arbitraria | 42L ó 456966786151987643L |
| float | Número decimal | Coma flotante de doble precisión | 3.1415927 |
| complex | Número complejo | Parte real y parte imaginaria *j*. | (4.5 + 3j) |
| bool | Booleano | Valor booleano verdadero o falso | True o False |

* Mutable: si su contenido (o dicho valor) puede cambiarse en tiempo de ejecución.
* Inmutable: si su contenido (o dicho valor) no puede cambiarse en tiempo de ejecución.

**Condicionales**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=9)]

Una [sentencia condicional](https://es.wikipedia.org/wiki/Sentencia_condicional) (*if*) ejecuta su [bloque de código](https://es.wikipedia.org/wiki/Bloque_de_c%C3%B3digo) interno sólo si se cumple cierta condición. Se define usando la palabra clave if seguida de la condición, y el bloque de código. Condiciones adicionales, si las hay, se introducen usando elif seguida de la condición y su bloque de código. Todas las condiciones se evalúan secuencialmente hasta encontrar la primera que sea verdadera, y su bloque de código asociado es el único que se ejecuta. Opcionalmente, puede haber un bloque final (la palabra clave else seguida de un bloque de código) que se ejecuta sólo cuando todas las condiciones fueron falsas.

>>> verdadero = True

>>> **if** verdadero: *# No es necesario poner "verdadero == True"*

... **print** "Verdadero"

... **else**:

... **print** "Falso"

...

Verdadero

>>> lenguaje = "Python"

>>> **if** lenguaje == "C": *# lenguaje no es "C", por lo que este bloque se obviará y evaluará la siguiente condición*

... **print** "Lenguaje de programación: C"

... **elif** lenguaje == "Python": *# Se pueden añadir tantos bloques "elif" como se quiera*

... **print** "Lenguaje de programación: Python"

... **else**: *# En caso de que ninguna de las anteriores condiciones fuera cierta, se ejecutaría este bloque*

... **print** "Lenguaje de programación: indefinido"

...

Lenguaje de programación: Python

>>> **if** verdadero **and** lenguaje == "Python": *# Uso de "and" para comprobar que ambas condiciones son verdaderas*

... **print** "Verdadero y Lenguaje de programación: Python"

...

Verdadero y Lenguaje de programación: Python

**Bucle for**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=10)]

El bucle *for* es similar a [*foreach*](https://es.wikipedia.org/wiki/Bucle_for#Bucle_Por_Cada_(For_Each)) en otros lenguajes. Recorre un objeto iterable, como una *lista*, una *tupla* o un generador, y por cada elemento del iterable ejecuta el bloque de código interno. Se define con la palabra clave for seguida de un nombre de variable, seguido de in, seguido del iterable, y finalmente el bloque de código interno. En cada iteración, el elemento siguiente del iterable se asigna al nombre de variable especificado:

>>> lista = ["a", "b", "c"]

>>> **for** i **in** lista: *# Iteramos sobre una lista, que es iterable*

... **print** i

...

a

b

c

>>> cadena = "abcdef"

>>> **for** i **in** cadena: *# Iteramos sobre una cadena, que también es iterable*

... **print** i, *# Añadiendo una coma al final hacemos que no introduzca un salto de línea, sino un espacio*

...

a b c d e f

**Bucle while**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=11)]

El bucle [*while*](https://es.wikipedia.org/wiki/Bucle_while) evalúa una condición y, si es verdadera, ejecuta el bloque de código interno. Continúa evaluando y ejecutando mientras la condición sea verdadera. Se define con la palabra clave while seguida de la condición, y a continuación el bloque de código interno:

>>> numero = 0

>>> **while** numero < 10:

... **print** numero,

... numero += 1 *#un buen programador modificará las variables de control al finalizar el ciclo while*

...

0 1 2 3 4 5 6 7 8 9

**Listas y Tuplas**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=12)]

* Para declarar una *lista* se usan los corchetes [], en cambio, para declarar una *tupla* se usan los paréntesis (). En ambas los elementos se separan por comas, y en el caso de las *tuplas* es necesario que tengan como mínimo una coma.
* Tanto las *listas* como las *tuplas* pueden contener elementos de diferentes tipos. No obstante las *listas* suelen usarse para elementos del mismo tipo en cantidad variable mientras que las *tuplas* se reservan para elementos distintos en cantidad fija.
* Para acceder a los elementos de una *lista* o *tupla* se utiliza un índice entero (empezando por "0", no por "1"). Se pueden utilizar índices negativos para acceder elementos a partir del final.
* Las *listas* se caracterizan por ser mutables, es decir, se puede cambiar su contenido en tiempo de ejecución, mientras que las *tuplas* son inmutables ya que no es posible modificar el contenido una vez creada.

**Listas**

>>> lista = ["abc", 42, 3.1415]

>>> lista[0] *# Acceder a un elemento por su índice*

'abc'

>>> lista[-1] *# Acceder a un elemento usando un índice negativo*

3.1415

>>> lista.append(True) *# Añadir un elemento al final de la lista*

>>> lista

['abc', 42, 3.1415, True]

>>> **del** lista[3] *# Borra un elemento de la lista usando un índice (en este caso: True)*

>>> lista[0] = "xyz" *# Re-asignar el valor del primer elemento de la lista*

>>> lista[0:2] *# Mostrar los elementos de la lista del índice "0" al "2" (sin incluir este último)*

['xyz', 42]

>>> lista\_anidada = [lista, [True, 42L]] *# Es posible anidar listas*

>>> lista\_anidada

[['xyz', 42, 3.1415], [True, 42L]]

>>> lista\_anidada[1][0] *# Acceder a un elemento de una lista dentro de otra lista (del segundo elemento, mostrar el primer elemento)*

True

**Tuplas**

>>> tupla = ("abc", 42, 3.1415)

>>> tupla[0] *# Acceder a un elemento por su índice*

'abc'

>>> **del** tupla[0] *# No es posible borrar (ni añadir) un elemento en una tupla, lo que provocará una excepción*

( Excepción )

>>> tupla[0] = "xyz" *# Tampoco es posible re-asignar el valor de un elemento en una tupla, lo que también provocará una excepción*

( Excepción )

>>> tupla[0:2] *# Mostrar los elementos de la tupla del índice "0" al "2" (sin incluir este último)*

('abc', 42)

>>> tupla\_anidada = (tupla, (True, 3.1415)) *# También es posible anidar tuplas*

>>> 1, 2, 3, "abc" *# Esto también es una tupla, aunque es recomendable ponerla entre paréntesis (recuerda que requiere, al menos, una coma)*

(1, 2, 3, 'abc')

>>> (1) *# Aunque entre paréntesis, esto no es una tupla, ya que no posee al menos una coma, por lo que únicamente aparecerá el valor*

1

>>> (1,) *# En cambio, en este otro caso, sí es una tupla*

(1,)

>>> (1, 2) *# Con más de un elemento no es necesaria la coma final*

(1, 2)

>>> (1, 2,) *# Aunque agregarla no modifica el resultado*

(1, 2)

**Diccionarios**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=13)]

* Para declarar un *diccionario* se usan las llaves {}. Contienen elementos separados por comas, donde cada elemento está formado por un par clave:valor (el símbolo :separa la clave de su valor correspondiente).
* Los *diccionarios* son mutables, es decir, se puede cambiar el contenido de un valor en tiempo de ejecución.
* En cambio, las claves de un *diccionario* deben ser inmutables. Esto quiere decir, por ejemplo, que no podremos usar ni *listas* ni *diccionarios* como claves.
* El valor asociado a una clave puede ser de cualquier *tipo de dato*, incluso un *diccionario*.

>>> diccionario = {"cadena": "abc", "numero": 42, "lista": [True, 42L]} *# Diccionario que tiene diferentes valores por cada clave, incluso una lista*

>>> diccionario["cadena"] *# Usando una clave, se accede a su valor*

'abc'

>>> diccionario["lista"][0] *# Acceder a un elemento de una lista dentro de un valor (del valor de la clave "lista", mostrar el primer elemento)*

True

>>> diccionario["cadena"] = "xyz" *# Re-asignar el valor de una clave*

>>> diccionario["cadena"]

'xyz'

>>> diccionario["decimal"] = 3.1415927 *# Insertar un nuevo elemento clave:valor*

>>> diccionario["decimal"]

3.1415927

>>> diccionario\_mixto = {"tupla": (True, 3.1415), "diccionario": diccionario} *# También es posible que un valor sea un diccionario*

>>> diccionario\_mixto["diccionario"]["lista"][1] *# Acceder a un elemento dentro de una lista, que se encuentra dentro de un diccionario*

42L

>>> diccionario = {("abc",): 42} *# Sí es posible que una clave sea una tupla, pues es inmutable*

>>> diccionario = {["abc"]: 42} *# No es posible que una clave sea una lista, pues es mutable, lo que provocará una excepción*

( Excepción )

**Sentencia Switch Case**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=14)]

Si bien Python no tiene la estructura Switch, hay varias formas de realizar la operación típica que realizaríamos con una sentencia switch case

**Usando if, elif, else**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=15)]

Podemos usar la estructura de la siguiente manera

>>> **if** condicion1:

... hacer1

>>> **elif** condicion2:

... hacer2

>>> **elif** condicion3:

... hacer3

>>> **else**:

... hacer

En esa estructura se ejecutara controlando la condicion1, si no se cumple pasara a la siguiente y así sucesivamente hasta entrar en el else. Un ejemplo practico seria:

>>> **def** calculo(op,a,b):

... **if** 'sum' == op:

... **return** a + b

... **elif** 'rest' == op:

... **return** a - b

... **elif** 'mult' == op:

... **return** a \* b

... **elif** 'div' == op:

... **return** a/b

... **else**:

... **return** None

>>>

>>> **print**(calculo('sum',3,4))

7

Podriamos decir que el lado negativo de la sentencia armada con if, elif y else es que si la lista de posibles operaciones es muy larga, las tiene que recorrer una por una hasta llegar a la correcta.

**Usando diccionario**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=16)]

Podemos usar un diccionario para el mismo ejemplo:

>>> **def** calculo(op,a,b):

... **return** {

... 'sum': **lambda**: a + b,

... 'rest': **lambda**: a - b,

... 'mult': **lambda**: a \* b,

... 'div': **lambda**: a/b

... }.get(op, **lambda**: None)()

>>>

>>> **print**(calculo('sum',3,4))

7

De esta manera si las opciones fueran muchas no recorrería todas. Solo iría directamente a la operación buscada. en la ultima linea *.get(op, lambda: None)()* estamos dando la opción por defecto.

**Conjuntos**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=17)]

* Los *conjuntos* se construyen mediante set(items) donde *items* es cualquier objeto *iterable*, como *listas* o *tuplas*. Los *conjuntos* no mantienen el orden ni contienen elementos duplicados.
* Se suelen utilizar para eliminar duplicados de una secuencia, o para operaciones matemáticas como [intersección](https://es.wikipedia.org/wiki/Intersecci%C3%B3n_de_conjuntos), [unión](https://es.wikipedia.org/wiki/Uni%C3%B3n_de_conjuntos), [diferencia](https://es.wikipedia.org/wiki/Diferencia_de_conjuntos) y [diferencia simétrica](https://es.wikipedia.org/wiki/Diferencia_sim%C3%A9trica).

>>> conjunto\_inmutable = frozenset(["a", "b", "a"]) *# Se utiliza una lista como objeto iterable*

>>> conjunto\_inmutable

frozenset(['a', 'b'])

>>> conjunto1 = set(["a", "b", "a"]) *# Primer conjunto mutable*

>>> conjunto1

set(['a', 'b'])

>>> conjunto2 = set(["a", "b", "c", "d"]) *# Segundo conjunto mutable*

>>> conjunto2

set(['a', 'c', 'b', 'd']) *# Recuerda, no mantienen el orden, como los diccionarios*

>>> conjunto1 & conjunto2 *# Intersección*

set(['a', 'b'])

>>> conjunto1 | conjunto2 *# Unión*

set(['a', 'c', 'b', 'd'])

>>> conjunto1 - conjunto2 *# Diferencia (1)*

set([])

>>> conjunto2 - conjunto1 *# Diferencia (2)*

set(['c', 'd'])

>>> conjunto1 ^ conjunto2 *# Diferencia simétrica*

set(['c', 'd'])

**Listas por comprensión**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=18)]

Una *lista por comprensión* (en inglés: list comprehension) es una expresión compacta para definir *listas*. Al igual que lambda, aparece en lenguajes funcionales. Ejemplos:

>>> range(5) *# La función "range" devuelve una lista, empezando en 0 y terminando con el número indicado menos uno*

[0, 1, 2, 3, 4]

>>> [i\*i **for** i **in** range(5)] *# Por cada elemento del rango, lo multiplica por sí mismo y lo agrega al resultado*

[0, 1, 4, 9, 16]

>>> lista = [(i, i + 2) **for** i **in** range(5)]

>>> lista

[(0, 2), (1, 3), (2, 4), (3, 5), (4, 6)]

**Funciones**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=19)]

* Las *funciones* se definen con la palabra clave def, seguida del nombre de la *función* y sus parámetros. Otra forma de escribir *funciones*, aunque menos utilizada, es con la palabra clave lambda (que aparece en lenguajes funcionales como [Lisp](https://es.wikipedia.org/wiki/Lisp)).
* El valor devuelto en las *funciones* con def será el dado con la instrucción return.

def:

>>> **def** suma(x, y = 2):

... **return** x + y *# Retornar la suma del valor de la variable "x" y el valor de "y"*

...

>>> suma(4) *# La variable "y" no se modifica, siendo su valor: 2*

6

>>> suma(4, 10) *# La variable "y" sí se modifica, siendo su nuevo valor: 10*

14

lambda:

>>> suma = **lambda** x, y = 2: x + y

>>> suma(4) *# La variable "y" no se modifica, siendo su valor: 2*

6

>>> suma(4, 10) *# La variable "y" sí se modifica, siendo su nuevo valor: 10*

14

**Clases**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=20)]

* Las *clases* se definen con la palabra clave class, seguida del nombre de la *clase* y, si [hereda](https://es.wikipedia.org/wiki/Herencia_m%C3%BAltiple) de otra *clase*, el nombre de esta.
* En Python 2.x es recomendable que una clase herede de "object", en Python 3.x esto ya no hará falta.
* En una [*clase*](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)) un "método" equivale a una "función", y un "atributo" equivale a una "variable".
* "\_\_init\_\_" es un método especial que se ejecuta al instanciar la clase, se usa generalmente para inicializar atributos y ejecutar métodos necesarios. Al igual que todos los métodos en Python, debe tener al menos un parámetro, generalmente se utiliza self. El resto de parámetros serán los que se indiquen al instanciar la *clase*.
* Los atributos que se desee que sean accesibles desde fuera de la *clase* se deben declarar usando self. delante del nombre.
* En python no existe el concepto de encapsulación,[26](https://es.wikipedia.org/wiki/Python#cite_note-26)​ por lo que el programador debe ser responsable de asignar los valores a los atributos

>>> **class** **Persona**(object):

... **def** \_\_init\_\_(self, nombre, edad):

... self.nombre = nombre *# Un atributo cualquiera*

... self.edad = edad *# Otro atributo cualquiera*

... **def** mostrar\_edad(self): *# Es necesario que, al menos, tenga un parámetro, generalmente: "self"*

... **print** self.edad *# mostrando un atributo*

... **def** modificar\_edad(self, edad): *# Modificando Edad*

... **if** edad < 0 **or** edad > 150: *# Se comprueba que la edad no sea menor de 0 (algo imposible), ni mayor de 150 (algo realmente difícil)*

... **return** False

... **else**: *# Si está en el rango 0-150, entonces se modifica la variable*

... self.edad = edad *# Se modifica la edad*

...

>>> p = Persona("Alicia", 20) *# Instanciar la clase, como se puede ver, no se especifica el valor de "self"*

>>> p.nombre *# La variable "nombre" del objeto sí es accesible desde fuera*

'Alicia'

>>> p.nombre = "Andrea" *# Y por tanto, se puede cambiar su contenido*

>>> p.nombre

'Andrea'

>>> p.mostrar\_edad() *# Se llama a un método de la clase*

20

>>> p.modificar\_edad(21) *# Es posible cambiar la edad usando el método específico que hemos hecho para hacerlo de forma controlada*

>>> p.mostrar\_edad()

21

**Módulos**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=21)]

Existen muchas propiedades que se pueden agregar al lenguaje importando módulos, que son "minicódigos" (la mayoría escritos también en Python) que proveen de ciertas funciones y clases para realizar determinadas tareas. Un ejemplo es el módulo [Tkinter](https://es.wikipedia.org/wiki/Tkinter)[27](https://es.wikipedia.org/wiki/Python#cite_note-27)​, que permite crear interfaces gráficas basadas en la biblioteca [Tk](https://es.wikipedia.org/wiki/Tk). Otro ejemplo es el módulo *os*, que provee acceso a muchas funciones del sistema operativo. Los módulos se agregan a los códigos escribiendo import seguida del nombre del módulo que queramos usar.[28](https://es.wikipedia.org/wiki/Python#cite_note-28)​

**Interfaz al sistema operativo**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=22)]

El módulo **os** provee funciones para interactuar con el sistema operativo:

>>> **import** **os** *# Módulo que provee funciones del sistema operativo*

>>> os.name *# Devuelve el nombre del sistema operativo*

'posix'

>>> os.mkdir("/tmp/ejemplo") *# Crea un directorio en la ruta especificada*

>>> **import** **time** *# Módulo para trabajar con fechas y horas*

>>> time.strftime("%Y-%m-**%d** %H:%M:%S") *# Dándole un cierto formato, devuelve la fecha y/u hora actual*

'2010-08-10 18:01:17'

Para tareas de administración de archivos, el módulo **shutil** provee una interfaz de más alto nivel:

>>> **import** **shutil**

>>> shutil.copyfile('datos.db', 'informacion.db')

'informacion.db'

>>> shutil.move('/build/programas', 'dir\_progs')

'dir\_progs'

**Comodines de archivos**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=23)]

El módulo **glob** provee una función para crear listas de archivos a partir de búsquedas con comodines en carpetas:

>>> **import** **glob**

>>> glob.glob('\*.py')

['numeros.py', 'ejemplo.py', 'ejemplo2.py']

**Argumentos de línea de órdenes**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=24)]

Los argumentos de línea de órdenes se almacenan en el atributo argv del módulo **sys** como una lista.

>>> **import** **sys**

>>> **print**(sys.argv)

['demostracion.py', 'uno', 'dos', 'tres']

**Matemática**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=25)]

El módulo **math** permite acceder a las funciones de matemática de punto flotante:

>>> **import** **math**

>>> math.cos(math.pi / 3)

0,494888338963

>>> math.log(1024, 2)

10.0

El módulo **random** se utiliza para realizar selecciones al azar:

>>> **import** **random**

>>> random.choice(['durazno', 'manzana', 'frutilla'])

'durazno'

>>> random.sample(range(100), 10) *# elección sin reemplazo*

[30, 23, 17, 24, 8, 81, 41, 80, 28, 13]

>>> random.random() *# un float al azar*

0.23370387692726126

>>> random.randrange(6) *# un entero al azar tomado de range(6)*

3

El módulo **statistics** se utiliza para estadística básica, por ejemplo: media, mediana, varianza, etc:

>>> **import** **statistics**

>>> datos = [1.75, 2.75, 1.25, 0.5, 0.25, 1.25, 3.5]

>>> statistics.mean(datos)

1.6071428571428572

>>> statistics.median(datos)

1.25

>>> statistics.variance(datos)

1.3720238095238095

**Fechas y Tiempos**[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=26)]

El módulo **datetime** permite manejar fechas y tiempos:

>>> **from** **datetime** **import** date

>>> hoy = date.today()

>>> hoy

datetime.date(2017, 8, 16)

Sistema de objetos[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=27)]

En **Python** todo es un *objeto* (incluso las [clases](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica))). Las *clases*, al ser *objetos*, son instancias de una [metaclase](https://es.wikipedia.org/wiki/Metaclase). **Python** además soporta [herencia múltiple](https://es.wikipedia.org/wiki/Herencia_m%C3%BAltiple) y [polimorfismo](https://es.wikipedia.org/wiki/Polimorfismo_(inform%C3%A1tica)).

>>> cadena = "abc" *# Una cadena es un objeto de "str"*

>>> cadena.upper() *# Al ser un objeto, posee sus propios métodos*

'ABC'

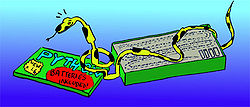
>>> lista = [True, 3.1415] *# Una lista es un objeto de "list"*

>>> lista.append(42L) *# Una lista también (al igual que todo) es un objeto, y también posee sus propios métodos*

>>> lista

[True, 3.1415, 42L]

Biblioteca estándar[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=28)]

[](https://commons.wikimedia.org/wiki/File:Python_batteries_included.jpg)

Python viene con "pilas incluidas"

Python tiene una gran biblioteca estándar, usada para una diversidad de tareas. Esto viene de la filosofía "pilas incluidas" (*"batteries included"*) en referencia a los módulos de Python. Los módulos de la biblioteca estándar pueden mejorarse por módulos personalizados escritos tanto en C como en Python. Debido a la gran variedad de herramientas incluidas en la biblioteca estándar, combinada con la habilidad de usar lenguajes de bajo nivel como C y C++, los cuales son capaces de interactuar con otras bibliotecas, Python es un lenguaje que combina su clara sintaxis con el inmenso poder de lenguajes menos elegantes.

Implementaciones[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=29)]

Existen diversas implementaciones del lenguaje:

* [CPython](https://es.wikipedia.org/wiki/CPython) es la implementación original, disponible para varias plataformas en el sitio oficial de Python.
* [IronPython](https://es.wikipedia.org/wiki/IronPython) es la implementación para .NET
* [Stackless Python](https://es.wikipedia.org/w/index.php?title=Stackless_Python&action=edit&redlink=1) es la variante de CPython que trata de no usar el *stack* de [C](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C) ([www.stackless.com](http://www.stackless.com/))
* [Jython](https://es.wikipedia.org/wiki/Jython) es la implementación hecha en [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java)
* [Pippy](https://es.wikipedia.org/w/index.php?title=Pippy&action=edit&redlink=1) es la implementación realizada para Palm ([pippy.sourceforge.net](http://pippy.sourceforge.net/))
* [PyPy](https://es.wikipedia.org/wiki/PyPy) es una implementación de Python escrita en Python y optimizada mediante [JIT](https://es.wikipedia.org/wiki/Compilaci%C3%B3n_en_tiempo_de_ejecuci%C3%B3n) ([pypy.org](http://pypy.org/))
* [ActivePython](https://es.wikipedia.org/w/index.php?title=ActivePython&action=edit&redlink=1) es una implementación privativa de Python con extensiones, para servidores en producción y aplicaciones de misión crítica desarrollado por ActiveState Software.

Diferencias entre Python 2.x y Python 3.x[[editar](https://es.wikipedia.org/w/index.php?title=Python&action=edit&section=30)]

El 13 de febrero de 2009[29](https://es.wikipedia.org/wiki/Python#cite_note-29)​ se lanzó una nueva versión de Python bajo el nombre clave *"Python 3000"* o, abreviado, *"Py3K"*.[30](https://es.wikipedia.org/wiki/Python#cite_note-30)​ Esta nueva versión incluye toda una serie de cambios que requieren reescribir el código de versiones anteriores. Para facilitar este proceso junto con Python 3 se ha publicado una herramienta automática llamada **2to3**.[31](https://es.wikipedia.org/wiki/Python#cite_note-31)​ Una lista completa de los cambios puede encontrarse en [Novedades de Python 3.0](http://docs.python.org/3.1/whatsnew/3.0.html).