

Python

Python	
	
Desarrollador(es)	
Python Software Foundation http://www.python.org/	
Información general	
Extensiones comunes	.py, .pyc, .pyd, .pyo, .pyw
Paradigma	multiparadigma: orientado a objetos, imperativo, funcional, reflexivo
Apareció en	1991
Diseñado por	Guido van Rossum
Última versión estable	3.3.2 / 2.7.5 (15 de mayo de 2013 / 15 de mayo de 2013)
Última versión en pruebas	3.4.0 alpha4 (20 de octubre de 2013)
Tipo de dato	débilmente tipado, dinámico
Implementaciones	CPython, IronPython, Jython, Python for S60, PyPy, PyGame , Unladen Swallow
Dialectos	Stackless Python, RPython
Influído por	ABC, ALGOL 68, C, Haskell, Icon, Lisp, Modula-3, Perl, Smalltalk, Java
Ha influido a	Boo, Cobra, D, Falcon, Genie, Groovy, Ruby, JavaScript, Cython, Go
Sistema operativo	Multiplataforma
Licencia	Python Software Foundation License

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License,^[1] que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Historia

Python fue creado a finales de los ochenta^[2] por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python.^[3]

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* (en inglés: *Benevolent Dictator for Life*, BDFL).

En 1991, van Rossum publicó el código de la versión 0.9.0 en `lt.sources.alt.sources`^{[4],[5]}. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: `str`, `list`, `dict`, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de Modula-3; van Rossum describe el módulo como “una de las mayores unidades de programación de Python”. El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula `else`. En el año 1994 se formó `[news:comp.lang.python 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: `lambda`, `reduce`, `filter` y `map`. Van Rossum explicó que “hace 12 años, Python adquirió `lambda`, `reduce()`, `filter()` y `map()`, cortesía de un hacker de Lisp que las extrañaba y que envió parches”.^[6] 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 (CNRI) en Reston, Virginia, donde lanzó varias versiones del 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.^[7] 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.^[8]

En el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo BeOpen PythonLabs. 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.^[9] 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.

Python 2.0 tomó una característica mayor del lenguaje de programación funcional 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 capaz de recolectar referencias cíclicas.



Guido van Rossum, creador de Python, en la convención OSCON 2006

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 era muy deseable. La licencia usada entonces, la Python License, incluía una cláusula estipulando que la licencia estaba gobernada por el estado de Virginia, por lo que, bajo la óptica de los abogados de 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.

Python 1.6.1 es esencialmente el mismo que Python 1.6, con unos pocos arreglos de bugs, y con una nueva licencia compatible con GPL.

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. 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 (PSF), una organización sin ánimo de lucro fundada en el año 2001, tomando como modelo la Apache Software Foundation. Incluido en este lanzamiento fue una implementación del scoping más parecida a las reglas de static scoping (del cual Scheme es el originador).^[10]

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodeName()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s";' % ast[1]
        else:
            print '"]'
    else:
        print '["';
        children = []
        for n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ', ' %s -> {' % nodename
        for n, namechildren:
            print '%s' % name,
```

Código Python con coloreado de sintaxis.

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.^[11] También fueron agregados los generadores que fueron inspirados por el lenguaje Icon.^[12]

Las adiciones a la biblioteca estándar de Python y las decisiones sintácticas fueron influenciadas fuertemente por Java en algunos casos: el package logging,^[13] introducido en la versión 2.3, está basado en log4j; el parser SAX, introducido en 2.0; el package threading,^[14] cuya clase *Thread* expone un subconjunto de la interfaz de la clase homónima en Java.

Características y paradigmas

Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional. Otros paradigmas están soportados mediante el uso de extensiones.

Python usa tipado dinámico y 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 o C++. 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, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme.

Filosofía

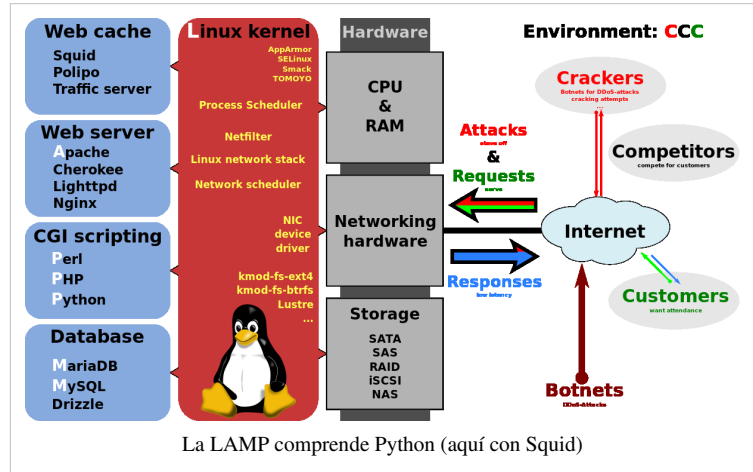
Los usuarios de Python se refieren a menudo a la **Filosofía Python** que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el código opaco u ofuscado es bautizado como "no pythonico" ("unpythonic" en inglés). Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters en *El 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.
- Aunque lo práctico gana a la pureza.
- 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.^[15]
- 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, *El Zen de Python*

Desde la versión 2.1.2, Python incluye estos puntos (en su versión original en inglés) como un huevo de pascua que se muestra al ejecutar `import this`.^[16]

Modo interactivo

El intérprete de Python estándar incluye un *modo interactivo* en el cual se escriben las instrucciones en una especie de intérprete 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 ^[17], bpython ^[18] o IPython, ^[19] que añaden funcionalidades extra al modo interactivo, como el autocompletado de código y el coloreado de la sintaxis del lenguaje.

Ejemplo del modo interactivo:

```
>>> 1 + 1
2
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Elementos del lenguaje

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.

El contenido de los bloques de código (bucles, funciones, clases, etc.) es delimitado mediante espacios o tabuladores, conocidos como indentación, antes de cada línea de órdenes pertenecientes al bloque.^[20] Python se diferencia así de otros lenguajes de programación que mantienen como costumbre declarar los bloques mediante un conjunto de caracteres, normalmente entre llaves `{}`.^{[21][22]} Se pueden utilizar tanto espacios como tabuladores para indentar el código, pero se recomienda no mezclarlos.^[23]

Función factorial en C (indentación opcional)

```
int factorial(int x)
{
    if (x == 0)
        return 1;
    else
        return x * factorial(x - 1);
}
```

Función factorial en Python (indentación obligatoria)

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

Debido al significado sintáctico de la indentación, una instrucción debe estar contenida en 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

Los *comentarios* se pueden poner de dos formas. La primera y más apropiada para comentarios largos es utilizando la notación `""" comentario """`, tres apóstrofes 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 nuestro código como, 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

Las *variables* se definen de forma dinámica, 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

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

Tipo	Clase	Notas	Ejemplo
str	Cadena	Inmutable	'Cadena'
unicode	Cadena	Versión 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.

Listas y Tuplas

- 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

- 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 )
```

Conjuntos

- 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, unión, diferencia y diferencia simétrica.

```
>>> 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

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

- 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).
- 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

- Las *clases* se definen con la palabra clave `class`, seguida del nombre de la *clase* y, si hereda 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* un "método" equivale a una "función", y una "propiedad" equivale a una "variable".^[24]
- "`__init__`" es un método especial que se ejecuta al instanciar la clase, se usa generalmente para inicializar propiedades 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*.
- Las propiedades 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,^[25] por lo que el programador debe ser responsable de asignar los valores a las propiedades

```
>>> class Persona(object):
...     def __init__(self, nombre, edad):
...         self.nombre = nombre # Una Propiedad cualquiera
...         self.edad = edad # Otra propiedad cualquiera
...     def mostrar_edad(self): # Es necesario que, al menos, tenga un
parámetro, generalmente: "self"
...         print self.edad # mostrando una propiedad
...     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) # Instanciamos 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() # Podemos llamar a un método de la clase
20
>>> p.modificar_edad(21) # Y podemos cambiar la edad usando el método
específico que hemos hecho para hacerlo de forma controlada
>>> p.mostrar_edad()
21
```

Condicionales

Una sentencia condicional (*if*) ejecuta su bloque de código 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

El bucle *for* es similar a *foreach* 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

El 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:
...     numero += 1
...     print numero,
...
1 2 3 4 5 6 7 8 9
```

Módulos

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`, que permite crear interfaces gráficas basadas en la biblioteca `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.

```
>>> 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/o hora actual
'2010-08-10 18:01:17'
```

Sistema de objetos

En **Python** todo es un *objeto* (incluso las clases). Las *clases*, al ser *objetos*, son instancias de una metaclasses. **Python** además soporta herencia múltiple y polimorfismo.

```
>>> 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

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.



Python viene con "pilas incluidas"

Implementaciones

Existen diversas implementaciones del lenguaje:

- CPython es la implementación original, disponible para varias plataformas en el sitio oficial de Python.
- IronPython es la implementación para .NET
- Stackless Python es la variante de CPython que trata de no usar el *stack* de C (www.stackless.com ^[26])
- Jython es la implementación hecha en Java
- Pippy es la implementación realizada para Palm (pippy.sourceforge.net ^[27])
- PyPy es una implementación de Python escrita en Python y optimizada mediante JIT (pypy.org ^[28])

Diferencias entre Python 2.x y Python 3.x

El 13 de febrero de 2009^[29] se lanzó una nueva versión de Python bajo el nombre clave "*Python 3000*" o, abreviado, "*Py3K*".^[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] Una lista completa de los cambios puede encontrarse en Novedades de Python 3.0 ^[32].

Referencias



- [1] History and License (<http://docs.python.org/license.html>)
- [2] The Making of Python (<http://www.artima.com/intv/pythonP.html>)
- [3] 1. Whetting Your Appetite (<http://docs.python.org/tutorial/appetite.html>)
- [4] news:a
- [5] <http://svn.python.org/view/python/trunk/Misc/HISTORY?view=markup&pathrev=51814> — Aviso: archivo grande. Ver el final del archivo.
- [6] The fate of reduce() in Python 3000 (<http://www.artima.com/weblogs/viewpost.jsp?thread=98196>)
- [7] Computer Programming for Everybody (<http://www.python.org/doc/essays/cp4e.html>)
- [8] Index of /cp4e (<http://www.python.org/cp4e/>)
- [9] What's New in Python 2.0 (<http://www.amk.ca/python/2.0/>)
- [10] PEP 227 -- Statically Nested Scopes (<http://www.python.org/dev/peps/pep-0227/>)
- [11] PEPs 252 and 253: Type and Class Changes (<http://docs.python.org/whatsnew/2.2.html#peps-252-and-253-type-and-class-changes>)
- [12] PEP 255: Simple Generators (<http://docs.python.org/whatsnew/2.2.html#pep-255-simple-generators>)
- [13] PEP 282 -- A Logging System (<http://www.python.org/dev/peps/pep-0282/>)
- [14] threading — Higher-level threading interface (<http://docs.python.org/library/threading.html>)
- [15] "Holandés" hace referencia a Guido van Rossum, el autor del lenguaje de programación Python, que es holandés. También hace referencia a la gran concentración de desarrolladores holandeses conocidos en relación a otras nacionalidades.
- [16] PEP 20 -- The Zen of Python (<http://www.python.org/dev/peps/pep-0020/>)
- [17] <http://docs.python.org/library/idle.html>
- [18] <http://bpython-interpreter.org/>

- [19] <http://ipython.scipy.org/>
- [20] <http://docs.python.org/tutorial/controlflow.html#defining-functions>
- [21] http://www.acm.uiuc.edu/webmonkeys/book/c_guide/1.3.html
- [22] <http://www.desarrolloweb.com/articulos/583.php>
- [23] <http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html#whitespace-1>
- [24] Clase (informática)
- [25] Encapsulacion en Python (<http://www.genbetadev.com/python/cazadores-de-mitos-las-propiedades-privadas-en-python>)
- [26] <http://www.stackless.com/>
- [27] <http://pippy.sourceforge.net/>
- [28] <http://pypy.org/>
- [29] <http://www.python.org/download/releases/>
- [30] PEP 3000 -- Python 3000 (<http://www.python.org/dev/peps/pep-3000/>)
- [31] 2to3 - Automated Python 2 to 3 code translation (<http://docs.python.org/3.1/library/2to3.html>)
- [32] <http://docs.python.org/3.1/whatsnew/3.0.html>


Bibliografía

- Knowlton, Jim (2009). *Python*. tr: Fernández Vélez, María Jesús (1 edición). Anaya Multimedia-Anaya Interactiva. ISBN 978-84-415-2513-9.
- Martelli, Alex (2007). *Python. Guía de referencia*. tr: Gorjón Salvador, Bruno (1 edición). Anaya Multimedia-Anaya Interactiva. ISBN 978-84-415-2317-3.

Enlaces externos

-  Wikimedia Commons alberga contenido multimedia sobre **Python**. Commons
-  Portal:Software Libre. Contenido relacionado con **Software Libre**.

Wikilibros

-  Wikilibros alberga un libro o manual sobre **Inmersión en Python**.
- Traducción al español del tutorial oficial de Python (<http://python.org.ar/pyar/Tutorial>)
- Python para principiantes (Video tutorial) (<http://codigoweb.in/manual/video-tutorial-python-introduccion-instalacion-2>)
- Libro para aprender a programar en Python (<http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=webpages:abecerra:introprog-py.pdf>)
- Traducción al español del tutorial oficial de Python (<http://python.org.ar/pyar/Tutorial>)
- Introducción a Python para científicos e ingenieros (Formato vídeo) (https://www.youtube.com/playlist?list=PLGBbVX_WvN7bMwYe7wWV5TZt1a58jTggB)

Fuentes y contribuyentes del artículo

Python *Fuente:* <http://es.wikipedia.org/w/index.php?oldid=74311379> *Contribuyentes:* 217-126-150-34.uc.nombres.ttd.es, Aandresortiz, Abalastegui, Adryitan, Ala lee, Alecu, Alexav8, Almorca, Andreums, Anonimato1990, AntBiel, Arcangelsombra, Archimagow, Ark, Ascánder, Asqueladd, AstroNomo, At93, Atardecere, Benjapunk69, Biasoli, Bibliofilotrastornado, Blenderation, Bryant1410, Carlos Pino, Changux, Chassoul, Chuck es dios, Cinabrium, Cinevoro, Cobalttempest, CommonsDelinker, Cplusplus, Crisneda2000, Cybedu, Daniel De Leon Martinez, Dem, Dibujon, Diego4serve, Diegusjaimes, Dnu72, Dodo, Dura-Ace, Durerro, Dusan, Edgardo C, Edorka, Eloy, Elwikipedista, Emijrp, Especiales, FAR, Felga16, Fibonacci, Fixertool, Flashlack, Fmariluis, Fran4004, FrancoCorrea, Frutoseco, Galaxy4, Ganímedes, Garthof, Genba, George.bo, GermanX, Ggenellina, Greek, Gronky, Hari Seldon, Hprmedina, Igneus, Ikks, Indu, Isha, Jarfil, Javi1977, JavierCantero, Javierrami, Jcaraballo, Jcea, Jileon, Jiptohej, Jkbw, Jlenton, JoaquinFerreiro, Jorg Ag, Joserrientos, Jrarias2005, JuanGoldenboy, Karlhangas, Kokoo, Kved, Leek, Leonardocaballero, Leonpolanco, Levhita, Lluvia, LogC, LordT, Lucianobrom, Lucien leGrey, Macar, Machipremier, Madalberta, Mandramas, Manuel Trujillo Berge, Manusoftar, Marxto, Matdrones, Maucendon, Maveric149, Metalzonix, Milestones, Misigon, Moriel, Muimota, Muro de Aguas, NaSz, NeV3rKiLL, Nekmo, Nernix1, Nessa los, Nicop, Ninovolador, Obelix83, Ortisa, PODA, PabloCastellano, Patrias, Patricio.lorente, Patronus1990, Pilaf, Poeta3d, PointToNull, Porao, PuercoPop, Puntoinfinito, Qwertyytrewqwert, Radical88, Rar, Rastrojo, Raulisea, Retama, Ricardo 369, Riviera, Rosarinagazo, Round Em, RoyFocker, Salu2, Sanbec, Sardanapalus, Sauron, Sbassi, ScotXW, Serafin88, Sergio Andres Segovia, Shevek, Shooke, Superzerocool, Surfaz, Syrusakbary, Tano4595, Tgor, Tico, Tin nqn, Tirithel, Tolitose, Tony Rotondas, Tosin2627, Tostadora, Tucapl, Texasatonga, Unf, Varano, Vigesimo, Vitamine, Vitorres, WLoku, Walter closser, Wikijandro, Willtron, Wolfete, YoaR, av202005.reshg.uci.edu, conversion script, 349 ediciones anónimas

Fuentes de imagen, Licencias y contribuyentes

Archivo:Python logo.svg *Fuente:* http://es.wikipedia.org/w/index.php?title=Archivo:Python_logo.svg *Licencia:* desconocido *Contribuyentes:* -

Archivo:Guido van Rossum OSCON 2006.jpg *Fuente:* http://es.wikipedia.org/w/index.php?title=Archivo:Guido_van_Rossum_OSCON_2006.jpg *Licencia:* Creative Commons Attribution-Sharealike 2.0 *Contribuyentes:* Doc Searls

Archivo:Python add5 syntax.svg *Fuente:* http://es.wikipedia.org/w/index.php?title=Archivo:Python_add5_syntax.svg *Licencia:* Copyrighted free use *Contribuyentes:* Xander89

File:LAMP software bundle.svg *Fuente:* http://es.wikipedia.org/w/index.php?title=Archivo:LAMP_software_bundle.svg *Licencia:* Creative Commons Attribution-Sharealike 3.0 *Contribuyentes:* User:ScotXW

Archivo:Python batteries included.jpg *Fuente:* http://es.wikipedia.org/w/index.php?title=Archivo:Python_batteries_included.jpg *Licencia:* Attribution *Contribuyentes:* Frank Stajano

Archivo:Commons-logo.svg *Fuente:* <http://es.wikipedia.org/w/index.php?title=Archivo:Commons-logo.svg> *Licencia:* Public Domain *Contribuyentes:* SVG version was created by User:Grunt and cleaned up by 3247, based on the earlier PNG version, created by Reidab.

Archivo:Heckert GNU white.svg *Fuente:* http://es.wikipedia.org/w/index.php?title=Archivo:Heckert_GNU_white.svg *Licencia:* Creative Commons Attribution-Sharealike 2.0 *Contribuyentes:* Aurelio A. Heckert <aaurium@gmail.com>

Archivo:Wikibooks-logo.svg *Fuente:* <http://es.wikipedia.org/w/index.php?title=Archivo:Wikibooks-logo.svg> *Licencia:* Creative Commons Attribution-Sharealike 3.0 *Contribuyentes:* User:Bastique, User:Ramac et al.

Licencia

Creative Commons Attribution-Share Alike 3.0
[//creativecommons.org/licenses/by-sa/3.0/](http://creativecommons.org/licenses/by-sa/3.0/)