Funciones integradas

El intérprete de Python tiene una serie de funciones y tipos integrados que siempre están disponibles. Se enumeran aquí en orden alfabético.

Funciones integradas

A

abs()

aiter()

all()

anext()

any()

ascii()

B

bin()

bool()

breakpoint()

bytearray()

bytes()

do

callable()

chr()

classmethod()

compile()

complex()

D

delattr()

dict()

dir()

divmod()

mi

enumerate()

eval()

exec()

F

filter()

float()

format()

frozenset()

GRAMO

getattr()

globals()

yo

hasattr()

hash()

help()

hex()

I

id()

input()

int()

isinstance()

issubclass()

iter()

yo

len()

list()

locals()

METRO

map()

max()

memoryview()

min()

norte

next()

Oh

object()

oct()

open()

ord()

PAG

pow()

print()

property()

R

range()

repr()

reversed()

round()

S

set()

setattr()

slice()

sorted()

staticmethod()

str()

sum()

super()

yo

tuple()

type()

V

vars()

O

zip()

\_

\_\_import\_\_()

abs ( x )

Devuelve el valor absoluto de un número. El argumento puede ser un entero, un número de punto flotante o un objeto que implemente \_\_abs\_\_(). Si el argumento es un número complejo, se devuelve su magnitud.

aiter ( async\_iterable )

Devuelve un iterador asincrónico para un iterable asincrónico . Equivale a llamar a x.\_\_aiter\_\_().

Nota: A diferencia de iter(), aiter()no tiene variante de 2 argumentos.

Agregado en la versión 3.10.

todo ( iterable )

Devuelve Truesi todos los elementos del iterable son verdaderos (o si el iterable está vacío). Equivalente a:

def all(iterable):

for element in iterable:

if not element:

return False

return True

anext que se puede esperar ( async\_iterator )

anext que se puede esperar ( async\_iterator , predeterminado )

Cuando se espera, devuelve el siguiente elemento del iterador asincrónico dado , o el valor predeterminado si se proporciona y el iterador está agotado.

Esta es la variante asíncrona del next()incorporado y se comporta de manera similar.

Esto llama al \_\_anext\_\_()método de async\_iterator y devuelve un awaitable . Awaitable devuelve el siguiente valor del iterador. Si se proporciona el valor predeterminado , se devuelve si el iterador se agota; de lo contrario, StopAsyncIterationse genera.

Agregado en la versión 3.10.

cualquier ( iterable )

Devuelve Truesi algún elemento del iterable es verdadero. Si el iterable está vacío, devuelve False. Equivalente a:

def any(iterable):

for element in iterable:

if element:

return True

return False

ascii ( objeto )

Como repr(), devuelve una cadena que contiene una representación imprimible de un objeto, pero escapa los caracteres no ASCII en la cadena devuelta mediante repr()los escapes \x, \uo \U. Esto genera una cadena similar a la devuelta por repr()en Python 2.

contenedor ( x )

Convierte un número entero en una cadena binaria con el prefijo “0b”. El resultado es una expresión de Python válida. Si x no es un intobjeto de Python, debe definir un \_\_index\_\_()método que devuelva un número entero. Algunos ejemplos:

>>>

bin(3)

'0b11'

bin(-10)

'-0b1010'

Si desea o no el prefijo “0b”, puede utilizar cualquiera de las siguientes formas.

>>>

format(14, '#b'), format(14, 'b')

('0b1110', '1110')

f'{14:#b}', f'{14:b}'

('0b1110', '1110')

Ver también format()para más información.

clase bool ( objeto = False , / )

Devuelve un valor booleano, es decir, uno de Trueo False. El argumento se convierte utilizando el procedimiento de prueba de verdad estándar . Si el argumento es falso o se omite, devuelve False; de lo contrario, devuelve True. La boolclase es una subclase de int(consulte Tipos numéricos: int, float, complex ). No se puede subclasificar más. Sus únicas instancias son Falsey True(consulte Tipo booleano: bool ).

Cambiado en la versión 3.7: El parámetro ahora es solo posicional.

punto de interrupción ( \* args , \*\* kws )

Esta función te lleva al depurador en el sitio de la llamada. Específicamente, llama a sys.breakpointhook(), pasando argsy kwsdirectamente. De manera predeterminada, sys.breakpointhook()llama pdb.set\_trace()sin esperar argumentos. En este caso, es puramente una función de conveniencia, por lo que no tienes que importar pdbo escribir explícitamente tanto código para ingresar al depurador. Sin embargo, sys.breakpointhook()se puede configurar en alguna otra función y breakpoint()la llamará automáticamente, lo que te permite ingresar al depurador de tu elección. Si sys.breakpointhook()no es accesible, esta función generará RuntimeError.

De forma predeterminada, el comportamiento de breakpoint()se puede cambiar con elPYTHONBREAKPOINTVariable de entorno. Consulte sys.breakpointhook()para obtener detalles de uso.

Tenga en cuenta que esto no está garantizado si sys.breakpointhook() se ha reemplazado.

Genera un evento de auditoría builtins.breakpoint con el argumento breakpointhook.

Agregado en la versión 3.7.

clase bytearray ( fuente = b'' )

clase bytearray ( fuente , codificación )

clase bytearray ( fuente , codificación , errores )

Devuelve una nueva matriz de bytes. La bytearrayclase es una secuencia mutable de números enteros en el rango 0 <= x < 256. Tiene la mayoría de los métodos habituales de secuencias mutables, descritos en Tipos de secuencias mutables , así como la mayoría de los métodos que bytestiene el tipo, consulte Operaciones con bytes y bytearray .

El parámetro de origen opcional se puede utilizar para inicializar la matriz de diferentes maneras:

Si es una cadena , también debe proporcionar los parámetros de codificación (y opcionalmente, errores ); bytearray()luego convierte la cadena a bytes usando str.encode().

Si es un entero , la matriz tendrá ese tamaño y se inicializará con bytes nulos.

Si se trata de un objeto que se ajusta a la interfaz de búfer , se utilizará un búfer de solo lectura del objeto para inicializar la matriz de bytes.

Si es un iterable , debe ser un iterable de números enteros en el rango , que se utilizan como contenido inicial de la matriz.0 <= x < 256

Sin un argumento, se crea una matriz de tamaño 0.

Consulte también Tipos de secuencia binaria: bytes, bytearray, memoryview y objetos Bytearray .

bytes de clase ( fuente = b'' )

bytes de clase ( fuente , codificación )

bytes de clase ( fuente , codificación , errores )

Devuelve un nuevo objeto “bytes” que es una secuencia inmutable de números enteros en el rango . es una versión inmutable de – tiene los mismos métodos no mutables y el mismo comportamiento de indexación y segmentación.0 <= x < 256bytesbytearray

En consecuencia, los argumentos del constructor se interpretan como para bytearray().

Los objetos bytes también se pueden crear con literales, consulte Literales de cadenas y bytes .

Consulte también Tipos de secuencia binaria: bytes, bytearray, memoryview , objetos Bytes y operaciones Bytes y Bytearray .

invocable ( objeto )

Devuelve Truesi el argumento del objetoFalse parece invocable, si no lo es. Si devuelve True, aún es posible que falle una llamada, pero si lo es False, la llamada al objeto nunca tendrá éxito. Tenga en cuenta que las clases son invocables (invocar una clase devuelve una nueva instancia); las instancias son invocables si su clase tiene un \_\_call\_\_()método.

Agregado en la versión 3.2: esta función se eliminó por primera vez en Python 3.0 y luego se recuperó en Python 3.2.

chr ( i )

Devuelve la cadena que representa un carácter cuyo punto de código Unicode es el entero i . Por ejemplo, chr(97)devuelve la cadena 'a', mientras que chr(8364)devuelve la cadena '€'. Esta es la inversa de ord().

El rango válido para el argumento es de 0 a 1.114.111 (0x10FFFF en base 16). ValueErrorse generará si i está fuera de ese rango.

@ método de clase

Transformar un método en un método de clase.

Un método de clase recibe la clase como primer argumento implícito, al igual que un método de instancia recibe la instancia. Para declarar un método de clase, utilice este modismo:

class C:

@classmethod

def f(cls, arg1, arg2): ...

El @classmethodformulario es un decorador de funciones : consulte Definiciones de funciones para obtener más detalles.

Se puede llamar a un método de clase en la clase (como C.f()) o en una instancia (como C().f()). La instancia se ignora, excepto su clase. Si se llama a un método de clase para una clase derivada, el objeto de clase derivada se pasa como primer argumento implícito.

Los métodos de clase son diferentes a los métodos estáticos de C++ o Java. Si desea conocerlos, consulte staticmethod()esta sección. Para obtener más información sobre los métodos de clase, consulte La jerarquía de tipos estándar .

Cambiado en la versión 3.9: Los métodos de clase ahora pueden encapsular otros descriptores como property().

Cambiado en la versión 3.10: Los métodos de clase ahora heredan los atributos del método ( \_\_module\_\_, \_\_name\_\_, \_\_qualname\_\_, \_\_doc\_\_y \_\_annotations\_\_) y tienen un nuevo \_\_wrapped\_\_ atributo.

Obsoleto desde la versión 3.11, eliminado en la versión 3.13: los métodos de clase ya no pueden incluir otros descriptores como property().

compilar ( fuente , nombre de archivo , modo , indicadores = 0 , no\_heredar = False , optimizar = -1 )

Compilar el código fuente en un código o un objeto AST. Los objetos de código pueden ejecutarse mediante exec()o eval(). El código fuente puede ser una cadena normal, una cadena de bytes o un objeto AST. Consulta la astdocumentación del módulo para obtener información sobre cómo trabajar con objetos AST.

El argumento del nombre de archivo debe indicar el archivo desde el cual se leyó el código; pasar algún valor reconocible si no se leyó desde un archivo ( '<string>'se usa comúnmente).

El argumento de modo especifica qué tipo de código se debe compilar; puede ser 'exec'si el código fuente consiste en una secuencia de declaraciones, 'eval'si consiste en una sola expresión o 'single'si consiste en una sola declaración interactiva (en el último caso, se imprimirán las declaraciones de expresión que evalúen algo distinto de None).

Los argumentos opcionales flags y dont\_inherit controlan qué opciones del compilador deben activarse y qué características futuras deben permitirse. Si ninguno está presente (o ambos son cero), el código se compila con los mismos flags que afectan al código que está llamando a compile(). Si se proporciona el argumento flags y dont\_inherit no lo está (o es cero), entonces se utilizan las opciones del compilador y las declaraciones futuras especificadas por el argumento flags además de las que se utilizarían de todos modos. Si dont\_inherit es un entero distinto de cero, entonces el argumento flags lo es: los flags (características futuras y opciones del compilador) en el código circundante se ignoran.

Las opciones del compilador y las declaraciones futuras se especifican mediante bits que se pueden combinar mediante la operación OR para especificar varias opciones. El campo de bits necesario para especificar una característica futura determinada se puede encontrar como compiler\_flagatributo en la \_Featureinstancia del \_\_future\_\_módulo. Los indicadores del compilador se pueden encontrar en ast el módulo, con PyCF\_prefijo.

El argumento optimizar especifica el nivel de optimización del compilador; el valor predeterminado de -1selecciona el nivel de optimización del intérprete, tal como se indica en -Olas opciones. Los niveles explícitos son 0(sin optimización; \_\_debug\_\_es verdadero), 1(se eliminan las afirmaciones; \_\_debug\_\_es falso) o 2(también se eliminan las cadenas de documentación).

Esta función se activa SyntaxErrorsi la fuente compilada no es válida y ValueErrorsi contiene bytes nulos.

Si desea analizar el código Python en su representación AST, consulte ast.parse().

Genera un evento de auditoría compile con argumentos sourcey filename. Este evento también puede generarse mediante compilación implícita.

Nota Al compilar una cadena con código de varias líneas en el modo 'single'o 'eval', la entrada debe terminar con al menos un carácter de nueva línea. Esto es para facilitar la detección de instrucciones completas e incompletas en el codemódulo.

Advertencia Es posible que el intérprete de Python se bloquee con una cadena suficientemente grande o compleja al compilar un objeto AST debido a las limitaciones de profundidad de la pila en el compilador AST de Python.

Modificado en la versión 3.2: se permite el uso de saltos de línea en Windows y Mac. Además, la entrada en 'exec'modo ya no tiene que terminar en un salto de línea. Se agregó el parámetro optimizar .

Cambiado en la versión 3.5: Anteriormente, TypeErrorse generaba cuando se encontraban bytes nulos en el origen .

Agregado en la versión 3.8:ast.PyCF\_ALLOW\_TOP\_LEVEL\_AWAIT ahora se pueden pasar indicadores para habilitar soporte para await, , y de nivel superior .async forasync with

clase compleja ( número = 0 , / )

clase compleja ( cadena , / )

clase compleja ( real = 0 , imag = 0 )

Convierte una sola cadena o número en un número complejo, o crea un número complejo a partir de partes reales e imaginarias.

Ejemplos:

>>>

complex('+1.23')

(1.23+0j)

complex('-4.5j')

-4.5j

complex('-1.23+4.5j')

(-1.23+4.5j)

complex('\t( -1.23+4.5J )\n')

(-1.23+4.5j)

complex('-Infinity+NaNj')

(-inf+nanj)

complex(1.23)

(1.23+0j)

complex(imag=-4.5)

-4.5j

complex(-1.23, 4.5)

(-1.23+4.5j)

Si el argumento es una cadena, debe contener una parte real (en el mismo formato que para float()) o una parte imaginaria (en el mismo formato pero con un sufijo 'j'o 'J'), o ambas partes real e imaginaria (el signo de la parte imaginaria es obligatorio en este caso). La cadena puede estar opcionalmente rodeada por espacios en blanco y los paréntesis redondos '('y ')', que se ignoran. La cadena no debe contener espacios en blanco entre '+', '-', el sufijo 'j'o y el número decimal. Por ejemplo, está bien, pero genera . Más precisamente, la entrada debe cumplir con la regla de producción en la siguiente gramática, después de que se eliminen los paréntesis y los caracteres de espacio en blanco iniciales y finales:'J'complex('1+2j')complex('1 + 2j')ValueErrorcomplexvalue

valorcomplejo ::= floatvalue|

floatvalue("j" | "J") |

("j" | "J")

floatvalue sign absfloatvalue

Si el argumento es un número, el constructor sirve como una conversión numérica como inty float. Para un objeto Python general x, complex(x)delega a x.\_\_complex\_\_(). Si \_\_complex\_\_()no está definido, entonces recurre a \_\_float\_\_(). Si \_\_float\_\_()no está definido, entonces recurre a \_\_index\_\_().

Si se proporcionan dos argumentos o se utilizan argumentos de palabras clave, cada argumento puede ser de cualquier tipo numérico (incluido el complejo). Si ambos argumentos son números reales, devuelve un número complejo con el componente real real y el componente imaginario imag . Si ambos argumentos son números complejos, devuelve un número complejo con el componente real real.real-imag.imagy el componente imaginario real.imag+imag.real. Si uno de los argumentos es un número real, solo se utiliza su componente real en las expresiones anteriores.

Si se omiten todos los argumentos, devuelve 0j.

El tipo complejo se describe en Tipos numéricos: int, float, complex .

Cambiado en la versión 3.6: Se permite agrupar dígitos con guiones bajos como en los literales de código.

Cambiado en la versión 3.8: vuelve a \_\_index\_\_()si \_\_complex\_\_()y \_\_float\_\_()no están definidos.

delattr ( objeto , nombre )

Este es un pariente de setattr(). Los argumentos son un objeto y una cadena. La cadena debe ser el nombre de uno de los atributos del objeto. La función elimina el atributo nombrado, siempre que el objeto lo permita. Por ejemplo, es equivalente a . El nombre no necesita ser un identificador de Python (ver ).delattr(x, 'foobar')del x.foobarsetattr()

clase dict ( \*\* kwarg )

clase dict ( mapeo , \*\* kwarg )

clase dict ( iterable , \*\* kwarg )

Crea un nuevo diccionario. El dictobjeto es la clase dictionary. Consulta dictMapping Types — dict para obtener documentación sobre esta clase.

Para otros contenedores, consulte las clases integradas list, set, y tuple, así como el collectionsmódulo.

directorio ( )

dir ( objeto )

Sin argumentos, devuelve la lista de nombres en el ámbito local actual. Con un argumento, intenta devolver una lista de atributos válidos para ese objeto.

Si el objeto tiene un método llamado \_\_dir\_\_(), se llamará a este método y deberá devolver la lista de atributos. Esto permite que los objetos que implementan una función \_\_getattr\_\_()o un atributo personalizado \_\_getattribute\_\_()personalicen la forma en que dir()informan sus atributos.

Si el objeto no proporciona \_\_dir\_\_(), la función intenta recopilar información del \_\_dict\_\_atributo del objeto, si está definido, y de su tipo object. La lista resultante no necesariamente está completa y puede ser inexacta cuando el objeto tiene un . personalizado \_\_getattr\_\_().

El dir()mecanismo predeterminado se comporta de manera diferente con distintos tipos de objetos, ya que intenta producir la información más relevante, en lugar de la información completa:

Si el objeto es un objeto de módulo, la lista contiene los nombres de los atributos del módulo.

Si el objeto es un objeto de tipo o de clase, la lista contiene los nombres de sus atributos y, recursivamente, de los atributos de sus bases.

De lo contrario, la lista contiene los nombres de los atributos del objeto, los nombres de los atributos de su clase y, recursivamente, los nombres de los atributos de las clases base de su clase.

La lista resultante se ordena alfabéticamente. Por ejemplo:

>>>

import struct

dir() # show the names in the module namespace

['\_\_builtins\_\_', '\_\_name\_\_', 'struct']

dir(struct) # show the names in the struct module

['Struct', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_',

'\_\_initializing\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_',

'\_clearcache', 'calcsize', 'error', 'pack', 'pack\_into',

'unpack', 'unpack\_from']

class Shape:

def \_\_dir\_\_(self):

return ['area', 'perimeter', 'location']

s = Shape()

dir(s)

['area', 'location', 'perimeter']

Nota Dado dir()que se proporciona principalmente como una comodidad para su uso en un mensaje interactivo, intenta proporcionar un conjunto de nombres interesante más que un conjunto de nombres definidos de manera rigurosa o consistente, y su comportamiento detallado puede cambiar en las distintas versiones. Por ejemplo, los atributos de metaclase no están en la lista de resultados cuando el argumento es una clase.

divmod ( a , b )

Tome dos números (no complejos) como argumentos y devuelva un par de números que consisten en su cociente y resto cuando se utiliza la división de enteros. Con tipos de operandos mixtos, se aplican las reglas para los operadores aritméticos binarios. Para los enteros, el resultado es el mismo que . Para los números de punto flotante, el resultado es , donde q es generalmente pero puede ser 1 menos que eso. En cualquier caso es muy cercano a a , si no es cero tiene el mismo signo que b , y .(a // b, a % b)(q, a % b)math.floor(a / b)q \* b + a % ba % b0 <= abs(a % b) < abs(b)

enumerate ( iterable , inicio = 0 )

Devuelve un objeto enumerado. iterable debe ser una secuencia, un iterador o algún otro objeto que admita la iteración. El \_\_next\_\_()método del iterador devuelto por enumerate()devuelve una tupla que contiene un recuento (desde el inicio , que tiene como valor predeterminado 0) y los valores obtenidos al iterar sobre iterable .

>>>

seasons = ['Spring', 'Summer', 'Fall', 'Winter']

list(enumerate(seasons))

[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]

list(enumerate(seasons, start=1))

[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]

Equivalente a:

def enumerate(iterable, start=0):

n = start

for elem in iterable:

yield n, elem

n += 1

eval ( fuente , / , globales = Ninguno , locales = Ninguno )

Parámetros :

source ( str| objeto de código ) – Una expresión de Python.

globals ( dict| None) – El espacio de nombres global (predeterminado: None).

locals ( mapping | None) – El espacio de nombres local (predeterminado: None).

Devoluciones :

El resultado de la expresión evaluada.

Aumenta :

Los errores de sintaxis se informan como excepciones.

El argumento de expresión se analiza y evalúa como una expresión de Python (técnicamente hablando, una lista de condiciones) utilizando las asignaciones globals y locals como espacio de nombres global y local. Si el diccionario globals está presente y no contiene un valor para la clave , se inserta \_\_builtins\_\_una referencia al diccionario del módulo integrado bajo esa clave antes de que se analice la expresión . De esa manera, puede controlar qué integraciones están disponibles para el código ejecutado insertando su propio diccionario en globals antes de pasarlo a . Si se omite la asignación locals , se usa de manera predeterminada el diccionario globals . Si se omiten ambas asignaciones, la expresión se ejecuta con globals y locals en el entorno donde se llama. Tenga en cuenta que eval() solo tendrá acceso a los ámbitos anidados (no locales) en el entorno envolvente si ya están referenciados en el ámbito que está llamando (por ejemplo, a través de una declaración).builtins\_\_builtins\_\_eval()eval()eval()nonlocal

Ejemplo:

>>>

x = 1

eval('x+1')

2

Esta función también se puede utilizar para ejecutar objetos de código arbitrarios (como los creados por compile()). En este caso, pase un objeto de código en lugar de una cadena. Si el objeto de código se ha compilado con 'exec'como argumento de modo , eval()el valor de retorno de será None.

Sugerencias: la función admite la ejecución dinámica de instrucciones exec() . Las funciones globals()y locals()devuelven el diccionario global y local actual, respectivamente, que pueden resultar útiles para su uso por parte de eval()o exec().

Si la fuente dada es una cadena, se eliminan los espacios iniciales y finales y las tabulaciones.

Busque ast.literal\_eval()una función que pueda evaluar de forma segura cadenas con expresiones que contengan solo literales.

Genera un evento de auditoría exec con el objeto de código como argumento. También se pueden generar eventos de compilación de código.

Cambiado en la versión 3.13: Los argumentos globales y locales ahora se pueden pasar como palabras clave.

Cambiado en la versión 3.13: La semántica del espacio de nombres local predeterminado se ha ajustado como se describe para el locals()incorporado.

exec ( fuente , / , globales = Ninguno , locales = Ninguno , \* , cierre = Ninguno )

Esta función admite la ejecución dinámica de código Python. source debe ser una cadena o un objeto de código. Si es una cadena, la cadena se analiza como un conjunto de instrucciones Python que luego se ejecutan (a menos que se produzca un error de sintaxis). [ 1 ] Si es un objeto de código, simplemente se ejecuta. En todos los casos, se espera que el código que se ejecuta sea válido como entrada de archivo (consulte la sección Entrada de archivo en el Manual de referencia). Tenga en cuenta que las instrucciones nonlocal, yield, y return no se pueden usar fuera de las definiciones de función, incluso dentro del contexto del código pasado a la exec()función. El valor de retorno es None.

En todos los casos, si se omiten las partes opcionales, el código se ejecuta en el ámbito actual. Si solo se proporciona globals , debe ser un diccionario (y no una subclase de diccionario), que se utilizará tanto para las variables globales como para las locales. Si se proporcionan globals y locals , se utilizan para las variables globales y locales, respectivamente. Si se proporcionan, locals puede ser cualquier objeto de mapeo. Recuerde que a nivel de módulo, globals y locals son el mismo diccionario.

Nota Cuando execse obtienen dos objetos separados como globales y locales , el código se ejecutará como si estuviera integrado en una definición de clase. Esto significa que las funciones y clases definidas en el código ejecutado no podrán acceder a las variables asignadas en el nivel superior (ya que las variables de "nivel superior" se tratan como variables de clase en una definición de clase).

Si el diccionario globals no contiene un valor para la clave \_\_builtins\_\_, se inserta una referencia al diccionario del módulo integrado builtinsdebajo de esa clave. De esa manera, puede controlar qué funciones integradas están disponibles para el código ejecutado insertando su propio \_\_builtins\_\_diccionario en globals antes de pasarlo a exec().

El argumento de cierre especifica un cierre: una tupla de variables de celda. Solo es válido cuando el objeto es un objeto de código que contiene variables libres (de cierre) . La longitud de la tupla debe coincidir exactamente con la longitud del atributo del objeto de código co\_freevars.

Genera un evento de auditoría exec con el objeto de código como argumento. También se pueden generar eventos de compilación de código.

Nota Las funciones integradas globals()y locals()devuelven el espacio de nombres global y local actual, respectivamente, que pueden resultar útiles para utilizar como segundo y tercer argumento de exec().

Nota Los valores locales predeterminados actúan como se describe para la función locals()a continuación. Pase un diccionario de valores locales explícito si necesita ver los efectos del código en los valores locales después de que la función exec()regrese.

Cambiado en la versión 3.11: Se agregó el parámetro de cierre .

Cambiado en la versión 3.13: Los argumentos globales y locales ahora se pueden pasar como palabras clave.

Cambiado en la versión 3.13: La semántica del espacio de nombres local predeterminado se ha ajustado como se describe para el locals()incorporado.

filtro ( función , iterable )

Construya un iterador a partir de aquellos elementos de iterable para los que function es verdadera. iterable puede ser una secuencia, un contenedor que admita iteraciones o un iterador. Si function es None, se supone que la función es identidad, es decir, se eliminan todos los elementos de iterable que sean falsos.

Tenga en cuenta que es equivalente a la expresión del generador si la función no es y si la función es .filter(function, iterable)(item for item in iterable if function(item))None(item for item in iterable if item)None

Consulte itertools.filterfalse()la función complementaria que devuelve elementos del iterable para los cuales la función es falsa.

clase float ( número = 0.0 , / )

clase float ( cadena , / )

Devuelve un número de punto flotante construido a partir de un número o una cadena.

Ejemplos:

>>>

float('+1.23')

1.23

float(' -12345\n')

-12345.0

float('1e-003')

0.001

float('+1E6')

1000000.0

float('-Infinity')

-inf

Si el argumento es una cadena, debe contener un número decimal, opcionalmente precedido por un signo y opcionalmente incrustado en un espacio en blanco. El signo opcional puede ser '+'o '-'; un '+'signo no tiene efecto en el valor producido. El argumento también puede ser una cadena que represente un NaN (no es un número), o un infinito positivo o negativo. Más precisamente, la entrada debe cumplir con la floatvalue regla de producción de la siguiente gramática, después de que se eliminen los caracteres de espacio en blanco iniciales y finales:

signo ::= "+" | "-"

infinito ::= "Infinito" | "inf"

nan ::= "nan"

dígito ::= <un dígito decimal Unicode

, es decir, caracteres de la categoría general Unicode Nd> partedigit ::= digit(["\_"] digit)\*

número ::= [ digitpart] "." digitpart| digitpart["."]

exponente ::= ("e" | "E") [ sign] floatnumber ::= [ ]

absfloatvalue ::= | | floatvalue ::= [ ]digitpart

numberexponentfloatnumberinfinitynan

signabsfloatvalue

Las mayúsculas y minúsculas no son importantes, por lo que, por ejemplo, “inf”, “Inf”, “INFINITY” y “iNfINity” son todas formas aceptables de escribir infinito positivo.

De lo contrario, si el argumento es un número entero o un número de punto flotante, se devuelve un número de punto flotante con el mismo valor (dentro de la precisión de punto flotante de Python). Si el argumento está fuera del rango de un número flotante de Python, OverflowErrorse generará un .

Para un objeto Python general x, float(x)delega en x.\_\_float\_\_(). Si \_\_float\_\_()no está definido, recurre a \_\_index\_\_().

Si no se proporciona ningún argumento, 0.0se devuelve.

El tipo float se describe en Tipos numéricos: int, float, complex .

Cambiado en la versión 3.6: Se permite agrupar dígitos con guiones bajos como en los literales de código.

Cambiado en la versión 3.7: El parámetro ahora es solo posicional.

Cambiado en la versión 3.8: retrocede a \_\_index\_\_()si \_\_float\_\_()no está definido.

formato ( valor , especificación\_de\_formato = '' )

Convierte un valor en una representación “formateada”, tal como lo controla format\_spec . La interpretación de format\_spec dependerá del tipo de argumento de valor ; sin embargo, existe una sintaxis de formato estándar que utilizan la mayoría de los tipos integrados: Minilenguaje de especificación de formato .

El formato predeterminado format\_spec es una cadena vacía que generalmente produce el mismo efecto que llamar a str(value).

Una llamada a se traduce a que omite el diccionario de instancias al buscar el método del valor. Se genera una excepción si la búsqueda del método alcanza y format\_spec no está vacío, o si format\_spec o el valor de retorno no son cadenas.format(value, format\_spec)type(value).\_\_format\_\_(value, format\_spec)\_\_format\_\_()TypeErrorobject

Cambiado en la versión 3.4:object().\_\_format\_\_(format\_spec) se genera TypeError si format\_spec no es una cadena vacía.

clase frozenset ( iterable = set() )

Devuelve un nuevo frozensetobjeto, opcionalmente con elementos tomados de iterable . frozensetes una clase incorporada. Consulta frozensetlos tipos de conjuntos — set, frozenset para obtener documentación sobre esta clase.

Para otros contenedores, consulte las clases integradas set, list, tupley dict, así como el collections módulo.

getattr ( objeto , nombre )

getattr ( objeto , nombre , valor predeterminado )

Devuelve el valor del atributo nombrado del objeto . name debe ser una cadena. Si la cadena es el nombre de uno de los atributos del objeto, el resultado es el valor de ese atributo. Por ejemplo, es equivalente a . Si el atributo nombrado no existe, se devuelve el valor predeterminado si se proporciona; de lo contrario, se genera. name no necesita ser un identificador de Python (consulte ).getattr(x, 'foobar')x.foobarAttributeErrorsetattr()

Nota Dado que la alteración de nombres privados ocurre en el momento de la compilación, uno debe alterar manualmente el nombre de un atributo privado (atributos con dos guiones bajos iniciales) para poder recuperarlo con getattr().

globales ( )

Devuelve el diccionario que implementa el espacio de nombres del módulo actual. Para el código dentro de funciones, esto se establece cuando se define la función y permanece igual independientemente de dónde se llame a la función.

hasattr ( objeto , nombre )

Los argumentos son un objeto y una cadena. El resultado es Truesi la cadena es el nombre de uno de los atributos del objeto, Falsesi no lo es. (Esto se implementa llamando y viendo si genera un o no).getattr(object, name)AttributeError

hash ( objeto )

Devuelve el valor hash del objeto (si tiene uno). Los valores hash son números enteros. Se utilizan para comparar rápidamente las claves del diccionario durante una búsqueda en el diccionario. Los valores numéricos que se comparan como iguales tienen el mismo valor hash (incluso si son de tipos diferentes, como es el caso de 1 y 1.0).

Nota Para los objetos con \_\_hash\_\_()métodos personalizados, tenga en cuenta que hash() se trunca el valor de retorno en función del ancho de bits de la máquina host.

ayuda ( )

ayuda ( solicitud )

Invocar el sistema de ayuda integrado. (Esta función está pensada para uso interactivo). Si no se proporciona ningún argumento, el sistema de ayuda interactivo se inicia en la consola del intérprete. Si el argumento es una cadena, se busca la cadena como el nombre de un módulo, función, clase, método, palabra clave o tema de documentación y se imprime una página de ayuda en la consola. Si el argumento es cualquier otro tipo de objeto, se genera una página de ayuda sobre el objeto.

Tenga en cuenta que si aparece una barra diagonal (/) en la lista de parámetros de una función al invocar help(), significa que los parámetros anteriores a la barra diagonal son solo posicionales. Para obtener más información, consulte la entrada de preguntas frecuentes sobre parámetros solo posicionales .

El módulo agrega esta función al espacio de nombres integrado site.

Cambiado en la versión 3.4: Los cambios en pydocy inspectsignifican que las firmas informadas para los invocables ahora son más completas y consistentes.

hexadecimal ( x )

Convierte un número entero en una cadena hexadecimal en minúsculas con el prefijo “0x”. Si x no es un objeto Python int, debe definir un \_\_index\_\_()método que devuelva un número entero. Algunos ejemplos:

>>>

hex(255)

'0xff'

hex(-42)

'-0x2a'

Si desea convertir un número entero en una cadena hexadecimal en mayúsculas o minúsculas, con prefijo o no, puede utilizar cualquiera de las siguientes formas:

>>>

'%#x' % 255, '%x' % 255, '%X' % 255

('0xff', 'ff', 'FF')

format(255, '#x'), format(255, 'x'), format(255, 'X')

('0xff', 'ff', 'FF')

f'{255:#x}', f'{255:x}', f'{255:X}'

('0xff', 'ff', 'FF')

Ver también format()para más información.

Consulte también int()cómo convertir una cadena hexadecimal en un entero utilizando una base de 16.

Nota Para obtener una representación de cadena hexadecimal para un flotante, utilice el float.hex()método.

id ( objeto )

Devuelve la “identidad” de un objeto. Se trata de un número entero que se garantiza que será único y constante para este objeto durante su vida útil. Dos objetos con vidas útiles que no se superponen pueden tener el mismo id() valor.

Detalle de implementación de CPython: Esta es la dirección del objeto en la memoria.

Genera un evento de auditoría builtins.id con el argumento id.

aporte ( )

entrada ( indicador )

Si el argumento de solicitud está presente, se escribe en la salida estándar sin una nueva línea final. Luego, la función lee una línea de la entrada, la convierte en una cadena (eliminando la nueva línea final) y la devuelve. Cuando se lee EOF, EOFErrorse genera . Ejemplo:

>>>

s = input('--> ')

--> Monty Python's Flying Circus

s

"Monty Python's Flying Circus"

readlineSi se cargó el módulo, input()lo usaremos para proporcionar funciones elaboradas de edición de líneas e historial.

Genera un evento de auditoría builtins.input con argumento promptantes de leer la entrada.

Genera un evento de auditoría builtins.input/result con el resultado después de leer la entrada con éxito.

clase int ( numero = 0 , / )

clase int ( cadena , / , base = 10 )

Devuelve un objeto entero construido a partir de un número o una cadena, o regresa 0si no se dan argumentos.

Ejemplos:

>>>

int(123.45)

123

int('123')

123

int(' -12\_345\n')

-12345

int('FACE', 16)

64206

int('0xface', 0)

64206

int('01110011', base=2)

115

Si el argumento define \_\_int\_\_(), int(x)devuelve x.\_\_int\_\_(). Si el argumento define \_\_index\_\_(), devuelve x.\_\_index\_\_(). Si el argumento define \_\_trunc\_\_(), devuelve x.\_\_trunc\_\_(). Para números de punto flotante, esto trunca hacia cero.

Si el argumento no es un número o si se proporciona la base , debe ser una cadena, byteso bytearrayuna instancia que represente un entero con base en el radio . Opcionalmente, la cadena puede estar precedida por +o - (sin espacio entre ellos), tener ceros a la izquierda, estar rodeada de espacios en blanco y tener guiones bajos simples intercalados entre los dígitos.

Una cadena de enteros en base n contiene dígitos, cada uno de los cuales representa un valor de 0 a n-1. Los valores 0–9 se pueden representar con cualquier dígito decimal Unicode. Los valores 10–35 se pueden representar con ato z(o Ato Z). La base predeterminada es 10. Las bases permitidas son 0 y 2–36. Las cadenas en base 2, -8 y -16 pueden tener opcionalmente como prefijo 0b/ 0B, 0o/ 0Oo 0x/ 0X, al igual que los literales enteros en code . Para la base 0, la cadena se interpreta de forma similar a un literal entero en code , en el sentido de que la base real es 2, 8, 10 o 16 según lo determine el prefijo. La base 0 también prohíbe los ceros iniciales: no es legal, mientras que y son.int('010', 0)int('010')int('010', 8)

El tipo entero se describe en Tipos numéricos: int, float, complex .

Cambiado en la versión 3.4: Si base no es una instancia de inty el objeto base tiene un base.\_\_index\_\_método, se llama a ese método para obtener un entero para la base. Las versiones anteriores usaban base.\_\_int\_\_en lugar de base.\_\_index\_\_.

Cambiado en la versión 3.6: Se permite agrupar dígitos con guiones bajos como en los literales de código.

Cambiado en la versión 3.7: El primer parámetro ahora es solo posicional.

Cambiado en la versión 3.8: retrocede a \_\_index\_\_()si \_\_int\_\_()no está definido.

Cambiado en la versión 3.11: La delegación a \_\_trunc\_\_()está obsoleta.

Modificado en la versión 3.11:int las entradas de cadenas y las representaciones de cadenas se pueden limitar para ayudar a evitar ataques de denegación de servicio. ValueErrorSe genera un error cuando se excede el límite al convertir una cadena en un into cuando la conversión de un inten una cadena excedería el límite. Consulte la documentación sobre la limitación de longitud de conversión de cadenas enteras .

isinstance ( objeto , classinfo )

Devuelve Truesi el argumento object es una instancia del argumento classinfo , o de una subclase (directa, indirecta o virtual ) del mismo. Si object no es un objeto del tipo dado, la función siempre devuelve False. Si classinfo es una tupla de tipo objects (o recursivamente, otras tuplas similares) o un Union Type de múltiples tipos, devuelve Truesi object es una instancia de cualquiera de los tipos. Si classinfo no es un tipo o una tupla de tipos y tuplas similares, TypeErrorse genera una excepción. TypeErrorno se puede generar para un tipo no válido si una verificación anterior tiene éxito.

Cambiado en la versión 3.10: classinfo puede ser un tipo de unión .

issubclass ( clase , información de clase )

Devuelve Truesi la clase es una subclase (directa, indirecta o virtual ) de classinfo . Una clase se considera una subclase de sí misma. classinfo puede ser una tupla de objetos de clase (o recursivamente, otras tuplas similares) o un Union Type , en cuyo caso devuelve Truesi la clase es una subclase de cualquier entrada en classinfo . En cualquier otro caso, TypeError se genera una excepción.

Cambiado en la versión 3.10: classinfo puede ser un tipo de unión .

iter ( objeto )

iter ( objeto , centinela )

Devuelve un objeto iterador . El primer argumento se interpreta de forma muy diferente según la presencia del segundo argumento. Sin un segundo argumento, object debe ser un objeto de colección que admita el protocolo iterable\_\_iter\_\_() (el método), o debe admitir el protocolo de secuencia (el \_\_getitem\_\_()método con argumentos enteros que comienzan en 0). Si no admite ninguno de esos protocolos, TypeErrorse lanza . Si se proporciona el segundo argumento, sentinel , entonces object debe ser un objeto invocable. El iterador creado en este caso llamará a object sin argumentos para cada llamada a su \_\_next\_\_()método; si el valor devuelto es igual a sentinel , StopIterationse lanzará , de lo contrario se devolverá el valor.

Véase también Tipos de iteradores .

Una aplicación útil de la segunda forma de iter()es construir un lector de bloques. Por ejemplo, leer bloques de ancho fijo de un archivo de base de datos binario hasta que se llega al final del archivo:

from functools import partial

with open('mydata.db', 'rb') as f:

for block in iter(partial(f.read, 64), b''):

process\_block(block)

longitud ( s )

Devuelve la longitud (la cantidad de elementos) de un objeto. El argumento puede ser una secuencia (como una cadena, bytes, tupla, lista o rango) o una colección (como un diccionario, un conjunto o un conjunto congelado).

Detalle de implementación de CPython: len se genera OverflowErroren longitudes mayores que sys.maxsize, como .range(2 \*\* 100)

lista de clases

lista de clases ( iterable )

En lugar de ser una función, listen realidad es un tipo de secuencia mutable, como se documenta en Listas y tipos de secuencia: lista, tupla, rango .

lugareños ( )

Devuelve un objeto de mapeo que representa la tabla de símbolos local actual, con nombres de variables como claves y sus referencias enlazadas actualmente como valores.

En el ámbito del módulo, así como cuando se usa exec()o eval()con un solo espacio de nombres, esta función devuelve el mismo espacio de nombres que globals().

En el ámbito de la clase, devuelve el espacio de nombres que se pasará al constructor de la metaclase.

Cuando se utiliza exec()o eval()con argumentos locales y globales separados, devuelve el espacio de nombres local pasado a la llamada de función.

En todos los casos anteriores, cada llamada a locals()en un marco de ejecución determinado devolverá el mismo objeto de mapeo. Los cambios realizados a través del objeto de mapeo devuelto locals()serán visibles como variables locales asignadas, reasignadas o eliminadas, y la asignación, reasignación o eliminación de variables locales afectará inmediatamente el contenido del objeto de mapeo devuelto.

En un ámbito optimizado (que incluye funciones, generadores y corrutinas), cada llamada a locals()devuelve un diccionario nuevo que contiene las vinculaciones actuales de las variables locales de la función y cualquier referencia de celda no local. En este caso, los cambios de vinculación de nombres realizados a través del diccionario devuelto no se vuelven a escribir en las variables locales o referencias de celda no locales correspondientes, y la asignación, reasignación o eliminación de variables locales y referencias de celda no locales no afecta el contenido de los diccionarios devueltos previamente.

Llamar locals()como parte de una comprensión en una función, generador o corrutina es equivalente a llamarla en el ámbito contenedor, excepto que se incluirán las variables de iteración inicializadas de la comprensión. En otros ámbitos, se comporta como si la comprensión se estuviera ejecutando como una función anidada.

Llamarlo locals()como parte de una expresión de generador es equivalente a llamarlo en una función de generador anidada.

Cambiado en la versión 3.12: El comportamiento de locals()en una comprensión se ha actualizado como se describe enNúmero EP 709 .

Cambiado en la versión 3.13: Como parte dePEP 667 , ahora se define la semántica de mutación de los objetos de mapeo devueltos desde esta función. El comportamiento en ámbitos optimizados ahora es como se describió anteriormente. Además de estar definido, el comportamiento en otros ámbitos permanece sin cambios con respecto a las versiones anteriores.

mapa ( función , iterable , \* iterables )

Devuelve un iterador que aplica la función a cada elemento de iterable y genera los resultados. Si se pasan argumentos iterables adicionales, la función debe tomar esa cantidad de argumentos y se aplica a los elementos de todos los iterables en paralelo. Con varios iterables, el iterador se detiene cuando se agota el iterable más corto. Para los casos en los que las entradas de la función ya están organizadas en tuplas de argumentos, consulte itertools.starmap().

max ( iterable , \* , clave = Ninguna )

máx ( iterable , \* , predeterminado , clave = Ninguna )

máx ( arg1 , arg2 , \* args , clave = Ninguna )

Devuelve el elemento más grande en un iterable o el más grande de dos o más argumentos.

Si se proporciona un argumento posicional, debe ser un iterable . Se devuelve el elemento más grande del iterable. Si se proporcionan dos o más argumentos posicionales, se devuelve el más grande de ellos.

Hay dos argumentos opcionales que solo se pueden usar con palabras clave. El argumento clave especifica una función de ordenación de un argumento como la que se usa para list.sort(). El argumento predeterminado especifica un objeto que se devolverá si el iterable proporcionado está vacío. Si el iterable está vacío y no se proporciona el valor predeterminadoValueError , se lanza un .

Si hay varios elementos en el orden máximo, la función devuelve el primero que se encuentra. Esto es coherente con otras herramientas que preservan la estabilidad de la clasificación, como y .sorted(iterable, key=keyfunc, reverse=True)[0]heapq.nlargest(1, iterable, key=keyfunc)

Cambiado en la versión 3.4: Se agregó el parámetro predeterminado de solo palabras clave.

Cambiado en la versión 3.8: La clave puede ser None.

clase memoryview ( objeto )

Devuelve un objeto de “vista de memoria” creado a partir del argumento indicado. Consulta Vistas de memoria para obtener más información.

min ( iterable , \* , clave = Ninguna )

min ( iterable , \* , predeterminado , clave = Ninguna )

min ( arg1 , arg2 , \* args , clave = Ninguna )

Devuelve el elemento más pequeño en un iterable o el más pequeño de dos o más argumentos.

Si se proporciona un argumento posicional, debe ser un iterable . Se devuelve el elemento más pequeño del iterable. Si se proporcionan dos o más argumentos posicionales, se devuelve el más pequeño de ellos.

Hay dos argumentos opcionales que solo se pueden usar con palabras clave. El argumento clave especifica una función de ordenación de un argumento como la que se usa para list.sort(). El argumento predeterminado especifica un objeto que se devolverá si el iterable proporcionado está vacío. Si el iterable está vacío y no se proporciona el valor predeterminadoValueError , se lanza un .

Si hay varios elementos que son mínimos, la función devuelve el primero que se encuentra. Esto es coherente con otras herramientas que preservan la estabilidad de la clasificación, como y .sorted(iterable, key=keyfunc)[0]heapq.nsmallest(1, iterable, key=keyfunc)

Cambiado en la versión 3.4: Se agregó el parámetro predeterminado de solo palabras clave.

Cambiado en la versión 3.8: La clave puede ser None.

siguiente ( iterador )

siguiente ( iterador , predeterminado )

Recupera el siguiente elemento del iterador llamando a su \_\_next\_\_()método. Si se proporciona el valor predeterminado , se devuelve si se agota el iterador; de lo contrario, StopIterationse activa.

objeto de clase

Devuelve un nuevo objeto sin características. objectEs una base para todas las clases. Tiene métodos que son comunes a todas las instancias de clases de Python. Esta función no acepta ningún argumento.

Nota objectLas instancias no tienen \_\_dict\_\_ atributos, por lo que no puedes asignar atributos arbitrarios a una instancia de object.

oct ( x )

Convierte un número entero en una cadena octal con el prefijo “0o”. El resultado es una expresión de Python válida. Si x no es un intobjeto de Python, debe definir un \_\_index\_\_()método que devuelva un número entero. Por ejemplo:

>>>

oct(8)

'0o10'

oct(-56)

'-0o70'

Si desea convertir un número entero en una cadena octal con el prefijo “0o” o no, puede utilizar cualquiera de los siguientes métodos.

>>>

'%#o' % 10, '%o' % 10

('0o12', '12')

format(10, '#o'), format(10, 'o')

('0o12', '12')

f'{10:#o}', f'{10:o}'

('0o12', '12')

Ver también format()para más información.

open ( archivo , modo = 'r' , buffering = -1 , codificación = Ninguno , errores = Ninguno , nueva línea = Ninguno , closefd = True , abridor = Ninguno )

Abre el archivo y devuelve el objeto de archivo correspondiente . Si no se puede abrir el archivo, OSErrorse genera una . Consulta Lectura y escritura de archivos para obtener más ejemplos de cómo utilizar esta función.

file es un objeto similar a una ruta que proporciona la ruta (absoluta o relativa al directorio de trabajo actual) del archivo que se abrirá o un descriptor de archivo entero del archivo que se encapsulará. (Si se proporciona un descriptor de archivo, se cierra cuando se cierra el objeto de E/S devuelto a menos que closefd esté configurado en False.)

mode es una cadena opcional que especifica el modo en el que se abre el archivo. El valor predeterminado es , 'r'lo que significa que se abre para leer en modo de texto. Otros valores comunes son 'w'para escritura (truncando el archivo si ya existe), 'x'para creación exclusiva y 'a'para anexar (lo que en algunos sistemas Unix significa que todas las escrituras se anexan al final del archivo independientemente de la posición de búsqueda actual). En modo de texto, si no se especifica la codificación , la codificación utilizada depende de la plataforma: locale.getencoding()se llama para obtener la codificación de configuración regional actual. (Para leer y escribir bytes sin formato, use el modo binario y deje la codificación sin especificar). Los modos disponibles son:

Personaje

Significado

'r'

Abierto para lectura (predeterminado)

'w'

abierto para escritura, truncando primero el archivo

'x'

Abierto para creación exclusiva, falla si el archivo ya existe

'a'

Abierto para escritura, agregándose al final del archivo si existe

'b'

modo binario

't'

modo texto (predeterminado)

'+'

Abierto a actualización (lectura y escritura)

El modo predeterminado es 'r'(abrir para leer texto, un sinónimo de 'rt'). Los modos 'w+'y 'w+b'abren y truncan el archivo. Los modos 'r+' y 'r+b'abren el archivo sin truncarlo.

Como se mencionó en la Descripción general , Python distingue entre E/S binaria y de texto. Los archivos abiertos en modo binario (incluido 'b'el argumento mode ) devuelven el contenido como bytesobjetos sin ninguna decodificación. En modo texto (el valor predeterminado o cuando 't'se incluye en el argumento mode ), el contenido del archivo se devuelve como str, habiendo sido los bytes decodificados primero utilizando una codificación dependiente de la plataforma o utilizando la codificación especificada si se proporciona.

Nota Python no depende de la noción de archivos de texto del sistema operativo subyacente; todo el procesamiento lo realiza el propio Python y, por lo tanto, es independiente de la plataforma.

buffering es un entero opcional que se utiliza para establecer la política de buffering. Pase 0 para desactivar el buffering (solo se permite en modo binario), 1 para seleccionar el buffering de línea (solo se puede usar al escribir en modo de texto) y un entero > 1 para indicar el tamaño en bytes de un buffer de fragmento de tamaño fijo. Tenga en cuenta que especificar un tamaño de buffer de esta manera se aplica para E/S con buffer binario, pero TextIOWrapper(es decir, archivos abiertos con mode='r+') tendrían otro buffering. Para deshabilitar el buffering en TextIOWrapper, considere usar el write\_throughindicador para io.TextIOWrapper.reconfigure(). Cuando no se proporciona ningún argumento de buffering , la política de buffering predeterminada funciona de la siguiente manera:

Los archivos binarios se almacenan en búfer en fragmentos de tamaño fijo; el tamaño del búfer se elige mediante una heurística que intenta determinar el "tamaño de bloque" del dispositivo subyacente y recurre a io.DEFAULT\_BUFFER\_SIZE. En muchos sistemas, el búfer normalmente tendrá una longitud de 4096 u 8192 bytes.

Los archivos de texto “interactivos” (archivos para los que isatty() se devuelve True) utilizan el almacenamiento en búfer de línea. Otros archivos de texto utilizan la política descrita anteriormente para archivos binarios.

codificación es el nombre de la codificación que se utiliza para decodificar o codificar el archivo. Solo se debe utilizar en modo de texto. La codificación predeterminada depende de la plataforma (lo que locale.getencoding()se devuelva), pero se puede utilizar cualquier codificación de texto compatible con Python. Consulta el codecsmódulo para obtener la lista de codificaciones compatibles.

errors es una cadena opcional que especifica cómo se deben manejar los errores de codificación y decodificación; no se puede utilizar en modo binario. Hay una variedad de controladores de errores estándar disponibles (enumerados en Controladores de errores ), aunque cualquier nombre de controlador de errores que se haya registrado codecs.register\_error()también es válido. Los nombres estándar incluyen:

'strict'para generar una ValueErrorexcepción si hay un error de codificación. El valor predeterminado Nonetiene el mismo efecto.

'ignore'Ignora los errores. Ten en cuenta que ignorar los errores de codificación puede provocar la pérdida de datos.

'replace'hace que se inserte un marcador de reemplazo (como '?') donde hay datos mal formados.

'surrogateescape'representará cualquier byte incorrecto como unidades de código sustituto bajas que van desde U+DC80 a U+DCFF. Estas unidades de código sustituto se convertirán en los mismos bytes cuando surrogateescapese utilice el controlador de errores al escribir datos. Esto es útil para procesar archivos con una codificación desconocida.

'xmlcharrefreplace'Solo se admite cuando se escribe en un archivo. Los caracteres no admitidos por la codificación se reemplazan con la referencia de caracteres XML adecuada .&#nnn;

'backslashreplace'reemplaza datos malformados por secuencias de escape con barra invertida de Python.

'namereplace'(también solo se admite al escribir) reemplaza caracteres no admitidos con \N{...}secuencias de escape.

newline determina cómo analizar los caracteres de nueva línea del flujo. Puede ser None, '', '\n', '\r', y '\r\n'. Funciona de la siguiente manera:

Al leer la entrada de la secuencia, si la nueva línea es None, se habilita el modo de nuevas líneas universales. Las líneas en la entrada pueden terminar en '\n', '\r'o '\r\n', y estas se traducen a '\n'antes de ser devueltas al llamador. Si es '', se habilita el modo de nuevas líneas universales, pero los finales de línea se devuelven al llamador sin traducir. Si tiene alguno de los otros valores legales, las líneas de entrada solo terminan con la cadena dada y el final de línea se devuelve al llamador sin traducir.

Al escribir la salida en el flujo, si la nueva línea es None, cualquier '\n' carácter escrito se traduce al separador de línea predeterminado del sistema, os.linesep. Si la nueva línea es ''o '\n', no se realiza ninguna traducción. Si la nueva línea es cualquiera de los otros valores legales, cualquier '\n' carácter escrito se traduce a la cadena dada.

Si se especifica closefdFalse y se proporciona un descriptor de archivo en lugar de un nombre de archivo, el descriptor de archivo subyacente se mantendrá abierto cuando se cierre el archivo. Si se proporciona un nombre de archivo, debe ser closefdTrue (el valor predeterminado); de lo contrario, se generará un error.

Se puede utilizar un abridor personalizado pasando un objeto invocable como opener . El descriptor de archivo subyacente para el objeto de archivo se obtiene entonces llamando a opener con ( file , flags ). opener debe devolver un descriptor de archivo abierto (pasarlo os.opencomo opener da como resultado una funcionalidad similar a pasar None).

El archivo recién creado no es heredable .

El siguiente ejemplo utiliza el parámetro dir\_fdos.open() de la función para abrir un archivo relativo a un directorio determinado:

>>>

import os

dir\_fd = os.open('somedir', os.O\_RDONLY)

def opener(path, flags):

return os.open(path, flags, dir\_fd=dir\_fd)

with open('spamspam.txt', 'w', opener=opener) as f:

print('This will be written to somedir/spamspam.txt', file=f)

os.close(dir\_fd) # don't leak a file descriptor

El tipo de objeto de archivo devuelto por la open()función depende del modo. Cuando open()se utiliza para abrir un archivo en modo texto ( 'w', 'r', 'wt', 'rt', etc.), devuelve una subclase de io.TextIOBase(específicamente io.TextIOWrapper). Cuando se utiliza para abrir un archivo en modo binario con almacenamiento en búfer, la clase devuelta es una subclase de io.BufferedIOBase. La clase exacta varía: en modo binario de lectura, devuelve un io.BufferedReader; en modos binario de escritura y binario de anexión, devuelve un io.BufferedWriter, y en modo de lectura/escritura, devuelve un . Cuando el almacenamiento en búfer está deshabilitado, se devuelve io.BufferedRandomla secuencia sin procesar, una subclase de io.RawIOBase, .io.FileIO

Consulte también los módulos de manejo de archivos, como fileinput, io (donde open()se declara), os, os.path, tempfiley shutil.

Genera un evento de auditoría open con argumentos path, mode, flags.

Es posible que los argumentos modey flagshayan sido modificados o inferidos de la llamada original.

Cambiado en la versión 3.3:

Se agregó el parámetro de apertura .

'x'Se agregó el modo.

IOErrorsolía ser levantado, ahora es un alias de OSError.

FileExistsErrorAhora se genera si el archivo abierto en modo de creación exclusivo ( 'x') ya existe.

Cambiado en la versión 3.4:

El archivo ahora no es heredable.

Cambiado en la versión 3.5:

Si se interrumpe la llamada del sistema y el controlador de señales no genera una excepción, la función ahora vuelve a intentar la llamada del sistema en lugar de generar una InterruptedErrorexcepción (consultePEP 475 para la justificación).

'namereplace'Se agregó el controlador de errores.

Cambiado en la versión 3.6:

Se agregó soporte para aceptar objetos que implementan os.PathLike.

En Windows, al abrir un búfer de consola es posible que se devuelva una subclase io.RawIOBasedistinta de io.FileIO.

Cambiado en la versión 3.11:'U' Se ha eliminado el modo.

orden ( c )

Dada una cadena que representa un carácter Unicode, devuelve un entero que representa el punto de código Unicode de ese carácter. Por ejemplo, ord('a')devuelve el entero 97y ord('€')(símbolo del euro) devuelve 8364. Este es el inverso de chr().

pow ( base , exp , mod = Ninguno )

Devuelve la base a la potencia exp ; si mod está presente, devuelve la base a la potencia exp , módulo mod (calculado de manera más eficiente que ). La forma de dos argumentos es equivalente a usar el operador de potencia: .pow(base, exp) % modpow(base, exp)base\*\*exp

Los argumentos deben tener tipos numéricos. Con tipos de operandos mixtos, se aplican las reglas de coerción para operadores aritméticos binarios. Para int los operandos, el resultado tiene el mismo tipo que los operandos (después de la coerción) a menos que el segundo argumento sea negativo; en ese caso, todos los argumentos se convierten a float y se entrega un resultado float. Por ejemplo, devuelve , pero devuelve . Para una base negativa de tipo o y un exponente no entero, se entrega un resultado complejo. Por ejemplo, devuelve un valor cercano a . Mientras que, para una base negativa de tipo o con un exponente entero, se entrega un resultado float. Por ejemplo, devuelve .pow(10, 2)100pow(10, -2)0.01intfloatpow(-9, 0.5)3jintfloatpow(-9, 2.0)81.0

Para intlos operandos base y exp , si mod está presente, mod también debe ser de tipo entero y mod debe ser distinto de cero. Si mod está presente y exp es negativo, base debe ser un número primo relativo a mod . En ese caso, se devuelve , donde inv\_base es un inverso de base módulo mod .pow(inv\_base, -exp, mod)

A continuación se muestra un ejemplo de cálculo de una inversa para 38el módulo 97:

>>>

pow(38, -1, mod=97)

23

23 \* 38 % 97 == 1

True

Cambiado en la versión 3.8: Para intlos operandos, la forma de tres argumentos de powahora permite que el segundo argumento sea negativo, lo que permite el cálculo de inversas modulares.

Modificado en la versión 3.8: Permitir argumentos de palabras clave. Antes, solo se admitían argumentos posicionales.

print ( \* objetos , sep = ' ' , fin = '\n' , archivo = None , flush = False )

Imprima objetos en el archivo de flujo de texto , separados por sep y seguidos de end . sep , end , file y flush , si están presentes, deben proporcionarse como argumentos de palabras clave.

Todos los argumentos que no son palabras clave se convierten en cadenas como str()does y se escriben en el flujo, separados por sep y seguidos por end . Tanto sep como end deben ser cadenas; también pueden ser None, lo que significa que se utilizan los valores predeterminados. Si no se proporcionan objetosprint() , solo se escribirá end .

El argumento del archivo debe ser un objeto con un write(string)método; si no está presente o None, sys.stdoutse utilizará . Dado que los argumentos impresos se convierten en cadenas de texto, print()no se pueden utilizar con objetos de archivo en modo binario. Para estos, utilice file.write(...)en su lugar .

El almacenamiento en búfer de salida suele estar determinado por el archivo . Sin embargo, si el vaciado es verdadero, el flujo se vacía a la fuerza.

Cambiado en la versión 3.3: Se agregó el argumento de palabra clave flush .

propiedad de clase ( fget = None , fset = None , fdel = None , doc = None )

Devuelve un atributo de propiedad.

fget es una función para obtener el valor de un atributo. fset es una función para configurar el valor de un atributo. fdel es una función para eliminar el valor de un atributo. Y doc crea una cadena de documentación para el atributo.

Un uso típico es definir un atributo administrado x:

class C:

def \_\_init\_\_(self):

self.\_x = None

def getx(self):

return self.\_x

def setx(self, value):

self.\_x = value

def delx(self):

del self.\_x

x = property(getx, setx, delx, "I'm the 'x' property.")

Si c es una instancia de C , c.xinvocará el captador, invocará el definidor y el eliminador.c.x = valuedel c.x

Si se proporciona, doc será la cadena de documentación del atributo de propiedad. De lo contrario, la propiedad copiará la cadena de documentación de fget (si existe). Esto permite crear propiedades de solo lectura fácilmente utilizando property()como decorador :

class Parrot:

def \_\_init\_\_(self):

self.\_voltage = 100000

@property

def voltage(self):

"""Get the current voltage."""

return self.\_voltage

El @propertydecorador convierte el voltage()método en un “getter” para un atributo de solo lectura con el mismo nombre y establece la cadena de documentación para el voltaje en “Obtener el voltaje actual”.

@ captador

@establecedor​

@deleteador​

Un objeto de propiedad tiene métodos getter, settery deleterque se pueden utilizar como decoradores y que crean una copia de la propiedad con la función de acceso correspondiente establecida en la función decorada. Esto se explica mejor con un ejemplo:

class C:

def \_\_init\_\_(self):

self.\_x = None

@property

def x(self):

"""I'm the 'x' property."""

return self.\_x

@x.setter

def x(self, value):

self.\_x = value

@x.deleter

def x(self):

del self.\_x

Este código es exactamente equivalente al primer ejemplo. Asegúrese de darle a las funciones adicionales el mismo nombre que la propiedad original ( xen este caso).

El objeto de propiedad devuelto también tiene los atributos fget, fsety fdelcorrespondientes a los argumentos del constructor.

Cambiado en la versión 3.5: Las cadenas de documentación de los objetos de propiedad ahora se pueden escribir.

\_\_nombre\_\_

Atributo que contiene el nombre de la propiedad. El nombre de la propiedad se puede cambiar en tiempo de ejecución.

Agregado en la versión 3.13.

rango de clase ( parada )

rango de clase ( inicio , fin , paso = 1 )

En lugar de ser una función, rangees en realidad un tipo de secuencia inmutable, como se documenta en Rangos y tipos de secuencia: lista, tupla, rango .

repr ( objeto )

Devuelve una cadena que contiene una representación imprimible de un objeto. Para muchos tipos, esta función intenta devolver una cadena que generaría un objeto con el mismo valor cuando se pasa a eval(); de lo contrario, la representación es una cadena encerrada entre corchetes angulares que contiene el nombre del tipo del objeto junto con información adicional que a menudo incluye el nombre y la dirección del objeto. Una clase puede controlar lo que esta función devuelve para sus instancias definiendo un \_\_repr\_\_()método. Si sys.displayhook()no es accesible, esta función generará RuntimeError.

Esta clase tiene una representación personalizada que se puede evaluar:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_repr\_\_(self):

return f"Person('{self.name}', {self.age})"

invertido ( seq )

Devuelve un iterador inverso . seq debe ser un objeto que tenga un \_\_reversed\_\_()método o admita el protocolo de secuencia (el \_\_len\_\_()método y el \_\_getitem\_\_()método con argumentos enteros que comienzan en 0).

round ( número , ndigits = Ninguno )

Devuelve un número redondeado a una precisión de ndigits después del punto decimal. Si se omite ndigitsNone o es , devuelve el entero más cercano a su entrada.

Para los tipos integrados que admiten round(), los valores se redondean al múltiplo más cercano de 10 elevado a la potencia menos ndigits ; si dos múltiplos son igualmente cercanos, el redondeo se realiza hacia la opción par (por ejemplo, tanto round(0.5)y round(-0.5)son 0, y round(1.5)es 2). Cualquier valor entero es válido para ndigits (positivo, cero o negativo). El valor de retorno es un entero si se omite ndigitsNone o . De lo contrario, el valor de retorno tiene el mismo tipo que number .

Para un objeto Python general number, rounddelega en number.\_\_round\_\_.

Nota El comportamiento de round()los números de punto flotante puede resultar sorprendente: por ejemplo, da en lugar del valor esperado . Esto no es un error: es el resultado del hecho de que la mayoría de las fracciones decimales no se pueden representar exactamente como un número de punto flotante. Consulte Aritmética de punto flotante: problemas y limitaciones para obtener más información.round(2.675, 2)2.672.68

conjunto de clase

conjunto de clases ( iterable )

Devuelve un nuevo setobjeto, opcionalmente con elementos tomados de iterable . setes una clase incorporada. Consulta setlos tipos de conjuntos — set, frozenset para obtener documentación sobre esta clase.

Para otros contenedores, consulte las clases integradas frozenset, list, tupley dict, así como el collections módulo.

setattr ( objeto , nombre , valor )

Esta es la contraparte de getattr(). Los argumentos son un objeto, una cadena y un valor arbitrario. La cadena puede nombrar un atributo existente o un atributo nuevo. La función asigna el valor al atributo, siempre que el objeto lo permita. Por ejemplo, es equivalente a .setattr(x, 'foobar', 123)x.foobar = 123

El nombre no necesita ser un identificador de Python como se define en Identificadores y palabras clave a menos que el objeto elija imponerlo, por ejemplo, en un personalizado \_\_getattribute\_\_()o a través de \_\_slots\_\_. Un atributo cuyo nombre no sea un identificador no será accesible mediante la notación de punto, pero sí mediante getattr()etc.

Nota Dado que la alteración de nombres privados ocurre en el momento de la compilación, uno debe alterar manualmente el nombre de un atributo privado (atributos con dos guiones bajos iniciales) para poder configurarlo con setattr().

clase slice ( detener )

clase slice ( inicio , fin , paso = Ninguno )

Devuelve un objeto de segmento que representa el conjunto de índices especificados por . Los argumentos de inicio y paso tienen como valor predeterminado .range(start, stop, step)None

comenzar

detener

paso

Los objetos Slice tienen atributos de datos de solo lectura start, stopy stepque simplemente devuelven los valores de los argumentos (o sus valores predeterminados). No tienen ninguna otra funcionalidad explícita; sin embargo, NumPy y otros paquetes de terceros los utilizan.

Los objetos Slice también se generan cuando se utiliza la sintaxis de indexación extendida. Por ejemplo: a[start:stop:step]o . Consulte una versión alternativa que devuelve un iterador .a[start:stop, i]itertools.islice()

Cambiado en la versión 3.12: Los objetos Slice ahora son hashables (siempre que start, stopy stepsean hashables).

ordenado ( iterable , / , \* , clave = Ninguna , inversa = Falso )

Devuelve una nueva lista ordenada a partir de los elementos en iterable .

Tiene dos argumentos opcionales que deben especificarse como argumentos de palabras clave.

key especifica una función de un argumento que se utiliza para extraer una clave de comparación de cada elemento del iterable (por ejemplo, key=str.lower). El valor predeterminado es None(comparar los elementos directamente).

reverse es un valor booleano. Si se establece en True, los elementos de la lista se ordenan como si cada comparación fuera inversa.

Úselo para convertir una función cmpfunctools.cmp\_to\_key() de estilo antiguo en una función de tecla .

Se garantiza que la función incorporada sorted()sea estable. Una clasificación es estable si garantiza que no se cambiará el orden relativo de los elementos que se comparan como iguales; esto resulta útil para la clasificación en varias pasadas (por ejemplo, clasificar por departamento y luego por categoría salarial).

El algoritmo de ordenación utiliza únicamente <comparaciones entre elementos. Si bien definir un \_\_lt\_\_()método será suficiente para la ordenación, PEP 8 recomienda que se implementen las seis comparaciones enriquecidas . Esto ayudará a evitar errores al usar los mismos datos con otras herramientas de ordenamiento, comomax()las que se basan en un método subyacente diferente. La implementación de las seis comparaciones también ayuda a evitar confusiones en las comparaciones de tipos mixtos que pueden llamar al método reflejado\_\_gt\_\_().

Para ver ejemplos de clasificación y un breve tutorial de clasificación, consulte Técnicas de clasificación .

@ método estático

Transformar un método en un método estático.

Un método estático no recibe un primer argumento implícito. Para declarar un método estático, utilice este modismo:

class C:

@staticmethod

def f(arg1, arg2, argN): ...

El @staticmethodformulario es un decorador de funciones : consulte Definiciones de funciones para obtener más detalles.

Un método estático se puede llamar en la clase (como C.f()) o en una instancia (como C().f()). Además, el descriptor del método estático también se puede llamar, por lo que se puede utilizar en la definición de la clase (como f()).

Los métodos estáticos en Python son similares a los que se encuentran en Java o C++. Además, consulte classmethod()una variante que resulta útil para crear constructores de clases alternativos.

Al igual que todos los decoradores, también es posible llamarlos staticmethodcomo una función regular y hacer algo con su resultado. Esto es necesario en algunos casos en los que se necesita una referencia a una función desde el cuerpo de una clase y se desea evitar la transformación automática en un método de instancia. Para estos casos, utilice este modismo:

def regular\_function():

...

class C:

method = staticmethod(regular\_function)

Para obtener más información sobre los métodos estáticos, consulte La jerarquía de tipos estándar .

Cambiado en la versión 3.10: Los métodos estáticos ahora heredan los atributos del método ( , , \_\_module\_\_y \_\_name\_\_) , tienen un nuevo atributo y ahora se pueden llamar como funciones regulares.\_\_qualname\_\_\_\_doc\_\_\_\_annotations\_\_\_\_wrapped\_\_

clase str ( objeto = '' )

clase str ( objeto = b'' , codificación = 'utf-8' , errores = 'strict' )

Devuelve una strversión del objeto . Consulta str()los detalles.

stres la clase de cadena incorporada . Para obtener información general sobre cadenas, consulte Tipo de secuencia de texto: str .

suma ( iterable , / , inicio = 0 )

Las sumas comienzan con los elementos de un iterable de izquierda a derecha y devuelven el total. Los elementos del iterable normalmente son números y no se permite que el valor inicial sea una cadena.

Para algunos casos de uso, existen buenas alternativas a sum(). La forma preferida y rápida de concatenar una secuencia de cadenas es llamar a ''.join(sequence). Para agregar valores de punto flotante con precisión extendida, consulte math.fsum(). Para concatenar una serie de iterables, considere usar itertools.chain().

Cambiado en la versión 3.8: El parámetro de inicio se puede especificar como un argumento de palabra clave.

Cambiado en la versión 3.12: La suma de flotantes cambió a un algoritmo que brinda mayor precisión y mejor conmutatividad en la mayoría de las compilaciones.

clase super

clase super ( tipo , objeto\_o\_tipo = Ninguno )

Devuelve un objeto proxy que delega las llamadas de método a una clase principal o hermana del tipo . Esto resulta útil para acceder a métodos heredados que se han anulado en una clase.

El object\_or\_type determina el orden de resolución del método que se buscará. La búsqueda comienza desde la clase inmediatamente después del tipo .

Por ejemplo, si \_\_mro\_\_object\_or\_type es y el valor de type es , entonces busca .D -> B -> C -> A -> objectBsuper()C -> A -> object

El \_\_mro\_\_atributo de la clase correspondiente a object\_or\_type enumera el orden de búsqueda de resolución de métodos que utilizan tanto getattr()y super(). El atributo es dinámico y puede cambiar siempre que se actualice la jerarquía de herencia.

Si se omite el segundo argumento, el superobjeto devuelto no está vinculado. Si el segundo argumento es un objeto, debe ser verdadero. Si el segundo argumento es un tipo, debe ser verdadero (esto es útil para métodos de clase).isinstance(obj, type)issubclass(type2, type)

Cuando se llama directamente dentro de un método ordinario de una clase, ambos argumentos pueden omitirse (“argumento cero super()”). En este caso, type será la clase que encierra, y obj será el primer argumento de la función que encierra inmediatamente (normalmente self). (Esto significa que el argumento cero super()no funcionará como se espera dentro de funciones anidadas, incluidas las expresiones generadoras, que implícitamente crean funciones anidadas).

Existen dos casos de uso típicos para super . En una jerarquía de clases con herencia simple, super se puede utilizar para hacer referencia a clases padre sin nombrarlas explícitamente, lo que hace que el código sea más fácil de mantener. Este uso es muy similar al uso de super en otros lenguajes de programación.

El segundo caso de uso es el de admitir la herencia múltiple cooperativa en un entorno de ejecución dinámico. Este caso de uso es exclusivo de Python y no se encuentra en lenguajes compilados estáticamente o lenguajes que solo admiten herencia simple. Esto hace posible implementar “diagramas de diamante” donde múltiples clases base implementan el mismo método. Un buen diseño dicta que tales implementaciones tengan la misma firma de llamada en todos los casos (porque el orden de las llamadas se determina en tiempo de ejecución, porque ese orden se adapta a los cambios en la jerarquía de clases y porque ese orden puede incluir clases hermanas que son desconocidas antes del tiempo de ejecución).

Para ambos casos de uso, una llamada de superclase típica se ve así:

class C(B):

def method(self, arg):

super().method(arg) # This does the same thing as:

# super(C, self).method(arg)

Además de las búsquedas de métodos, super()también funciona para búsquedas de atributos. Un posible caso de uso para esto es llamar a descriptores en una clase principal o hermana.

Tenga en cuenta que super()se implementa como parte del proceso de vinculación para búsquedas explícitas de atributos con puntos, como super().\_\_getitem\_\_(name). Lo hace implementando su propio \_\_getattribute\_\_()método para buscar clases en un orden predecible que admita la herencia múltiple cooperativa. En consecuencia, super()no está definido para búsquedas implícitas que utilicen declaraciones u operadores como super()[name].

Tenga en cuenta también que, además de la forma de argumento cero, super()no se limita a su uso dentro de métodos. La forma de dos argumentos especifica los argumentos exactamente y hace las referencias adecuadas. La forma de argumento cero solo funciona dentro de una definición de clase, ya que el compilador completa los detalles necesarios para recuperar correctamente la clase que se está definiendo, así como para acceder a la instancia actual para los métodos ordinarios.

Para obtener sugerencias prácticas sobre cómo diseñar clases cooperativas utilizando super(), consulte la guía para usar super() .

tupla de clase

tupla de clase ( iterable )

En lugar de ser una función, tuplees en realidad un tipo de secuencia inmutable, como se documenta en Tuplas y tipos de secuencia: lista, tupla, rango .

tipo de clase ( objeto )

tipo de clase ( nombre , bases , dict , \*\* kwds )

Con un argumento, devuelve el tipo de un objeto . El valor de retorno es un tipo de objeto y, por lo general, el mismo objeto que devuelve object.\_\_class\_\_.

Se recomienda la isinstance()función incorporada para probar el tipo de un objeto, porque tiene en cuenta las subclases.

Con tres argumentos, devuelve un nuevo objeto de tipo. Esta es esencialmente una forma dinámica de la classdeclaración. La cadena de nombre\_\_name\_\_ es el nombre de la clase y se convierte en el atributo. La tupla de bases contiene las clases base y se convierte en el \_\_bases\_\_atributo; si está vacía, objectse agrega , la base definitiva de todas las clases. El diccionario dict contiene definiciones de atributos y métodos para el cuerpo de la clase; se puede copiar o encapsular antes de convertirse en el atributo. Las dos declaraciones siguientes crean objetos \_\_dict\_\_idénticos :type

>>>

class X:

a = 1

X = type('X', (), dict(a=1))

Ver también:

Documentación sobre atributos y métodos en clases .

Tipo de objetos

Los argumentos de palabras clave proporcionados al formulario de tres argumentos se pasan a la maquinaria de metaclase adecuada (generalmente \_\_init\_subclass\_\_()) de la misma manera que lo harían las palabras clave en una definición de clase (además de la metaclase ).

Consulte también Personalizar la creación de clases .

Cambiado en la versión 3.6: Las subclases typeque no se anulan type.\_\_new\_\_ya no pueden usar el formato de un argumento para obtener el tipo de un objeto.

variables ( )

vars ( objeto )

Devuelve el \_\_dict\_\_atributo de un módulo, clase, instancia o cualquier otro objeto con un \_\_dict\_\_atributo.

Los objetos como módulos e instancias tienen un \_\_dict\_\_ atributo actualizable; sin embargo, otros objetos pueden tener restricciones de escritura en sus \_\_dict\_\_atributos (por ejemplo, las clases usan un types.MappingProxyTypepara evitar actualizaciones directas del diccionario).

Sin argumento, vars()actúa como locals().

Se genera una TypeErrorexcepción si se especifica un objeto pero no tiene un \_\_dict\_\_atributo (por ejemplo, si su clase define el \_\_slots\_\_atributo).

Cambiado en la versión 3.13: El resultado de llamar a esta función sin un argumento se ha actualizado como se describe para la locals()función incorporada.

zip ( \* iterables , estricto = Falso )

Iterar sobre varios iterables en paralelo, produciendo tuplas con un elemento de cada uno.

Ejemplo:

>>>

for item in zip([1, 2, 3], ['sugar', 'spice', 'everything nice']):

print(item)

(1, 'sugar')

(2, 'spice')

(3, 'everything nice')

Más formalmente: zip()devuelve un iterador de tuplas, donde la i -ésima tupla contiene el i -ésimo elemento de cada uno de los iterables de argumentos.

Otra forma de pensarlo zip()es que convierte las filas en columnas y las columnas en filas. Esto es similar a transponer una matriz .

zip()es perezoso: los elementos no se procesarán hasta que se itere el iterable, por ejemplo, mediante un forbucle o envolviéndolo en un list.

Una cosa a tener en cuenta es que los iterables que se pasan zip()pueden tener longitudes diferentes; a veces por diseño y a veces debido a un error en el código que preparó estos iterables. Python ofrece tres enfoques diferentes para abordar este problema:

De forma predeterminada, zip()se detiene cuando se agota el iterable más corto. Ignorará los elementos restantes en los iterables más largos y cortará el resultado a la longitud del iterable más corto:

>>>

list(zip(range(3), ['fee', 'fi', 'fo', 'fum']))

[(0, 'fee'), (1, 'fi'), (2, 'fo')]

zip()se utiliza a menudo en casos en los que se supone que los iterables tienen la misma longitud. En tales casos, se recomienda utilizar la strict=True opción. Su salida es la misma que la de regular zip():

>>>

list(zip(('a', 'b', 'c'), (1, 2, 3), strict=True))

[('a', 1), ('b', 2), ('c', 3)]

A diferencia del comportamiento predeterminado, se genera un error ValueErrorsi un iterable se agota antes que los demás:

>>>

for item in zip(range(3), ['fee', 'fi', 'fo', 'fum'], strict=True):

print(item)

(0, 'fee')

(1, 'fi')

(2, 'fo')

Traceback (most recent call last):

...

ValueError: zip() argument 2 is longer than argument 1

Sin el strict=Trueargumento, cualquier error que resulte en iterables de diferentes longitudes será silenciado, posiblemente manifestándose como un error difícil de encontrar en otra parte del programa.

Los iterables más cortos se pueden rellenar con un valor constante para que todos los iterables tengan la misma longitud. Esto se hace mediante itertools.zip\_longest().

Casos extremos: con un único argumento iterable, zip()devuelve un iterador de 1-tuplas. Sin argumentos, devuelve un iterador vacío.

Consejos y trucos:

Se garantiza el orden de evaluación de izquierda a derecha de los iterables. Esto hace posible una expresión idiomática para agrupar una serie de datos en grupos de longitud n utilizando . Esto repite el mismo iterador las veces necesarias para que cada tupla de salida tenga el resultado de las llamadas al iterador. Esto tiene el efecto de dividir la entrada en fragmentos de longitud n.zip(\*[iter(s)]\*n, strict=True)nn

zip()en conjunción con el \*operador se puede utilizar para descomprimir una lista:

>>>

x = [1, 2, 3]

y = [4, 5, 6]

list(zip(x, y))

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

x2, y2 = zip(\*zip(x, y))

x == list(x2) and y == list(y2)

True

Cambiado en la versión 3.10: Se agregó el strictargumento.

\_\_import\_\_ ( nombre , globales = Ninguno , locales = Ninguno , fromlist = () , nivel = 0 )

Nota Esta es una función avanzada que no es necesaria en la programación diaria de Python, a diferencia de importlib.import\_module().

Esta función se invoca mediante la importdeclaración. Se puede reemplazar (importando el builtinsmódulo y asignándolo a builtins.\_\_import\_\_) para cambiar la semántica de la declaración, pero no seimport recomienda hacerlo ya que suele ser más sencillo utilizar ganchos de importación (consultePEP 302 ) para alcanzar los mismos objetivos y no causa problemas con el código que supone que se utiliza la implementación de importación predeterminada.\_\_import\_\_()También se desaconseja el uso directo de en favor deimportlib.import\_module().

La función importa el nombre del módulo , utilizando potencialmente los valores globales y locales dados para determinar cómo interpretar el nombre en un contexto de paquete. La lista de origen proporciona los nombres de los objetos o submódulos que se deben importar desde el módulo dado por nombre . La implementación estándar no utiliza su argumento local en absoluto y utiliza sus valores globales solo para determinar el contexto de paquete de la importdeclaración.

level especifica si se deben utilizar importaciones absolutas o relativas. 0(valor predeterminado) significa que solo se deben realizar importaciones absolutas. Los valores positivos para level indican la cantidad de directorios principales que se deben buscar en relación con el directorio del módulo que llama \_\_import\_\_()(consultePEP 328 para más detalles).

Cuando la variable de nombre tiene el formato package.module, normalmente se devuelve el paquete de nivel superior (el nombre hasta el primer punto), no el módulo nombrado por nombre . Sin embargo, cuando se proporciona un argumento fromlist no vacío , se devuelve el módulo nombrado por nombre .

Por ejemplo, la declaración da como resultado un código de bytes similar al siguiente:import spam

spam = \_\_import\_\_('spam', globals(), locals(), [], 0)

La declaración da como resultado este llamado:import spam.ham

spam = \_\_import\_\_('spam.ham', globals(), locals(), [], 0)

Tenga en cuenta cómo \_\_import\_\_()se devuelve el módulo de nivel superior aquí porque este es el objeto que está vinculado a un nombre mediante la importdeclaración.

Por otra parte, la afirmación resulta enfrom spam.ham import eggs, sausage as saus

\_temp = \_\_import\_\_('spam.ham', globals(), locals(), ['eggs', 'sausage'], 0)

eggs = \_temp.eggs

saus = \_temp.sausage

Aquí, el spam.hammódulo se devuelve desde \_\_import\_\_(). A partir de este objeto, se recuperan los nombres a importar y se les asignan sus respectivos nombres.

Si simplemente desea importar un módulo (potencialmente dentro de un paquete) por nombre, utilice importlib.import\_module().

Cambiado en la versión 3.3: Ya no se admiten valores negativos para el nivel (lo que también cambia el valor predeterminado a 0).

Cambiado en la versión 3.9: Cuando se utilizan las opciones de línea de comando -Eo , la variable de entorno-IPYTHONCASEOKAhora se ignora.

Notas al pie

[ 1 ]

Tenga en cuenta que el analizador solo acepta la convención de fin de línea de estilo Unix. Si está leyendo el código desde un archivo, asegúrese de utilizar el modo de conversión de nueva línea para convertir las nuevas líneas de estilo Windows o Mac.