Desconocido

## A Palabras clave de Python



*Las* palabras clave en Python (y en la mayoría de los lenguajes de programación) son palabras que tienen un significado especial. Se utilizan como parte del propio lenguaje de programación y, por tanto, no deben utilizarse para otra cosa. Por ejemplo, si intentas utilizar palabras clave como variables, o las utilizas de forma incorrecta, recibirás extraños mensajes de error de la consola de Python. Este apéndice describe cada una de las palabras clave de Python. Te resultará una referencia práctica a medida que sigas programando.

### y

La palabra clave and se utiliza para unir dos expresiones en una sentencia (como una sentencia if ) para decir que ambas expresiones deben ser verdaderas. Aquí tienes un ejemplo:

if age > 12 and age < 20:  
 print('Beware the teenager!!!!')

Este código significa que el valor de la variable age debe ser mayor que 12 y menor que 20 para que se imprima el mensaje.

### como

La palabra clave as puede utilizarse para dar otro nombre a un módulo importado. Por ejemplo, supón que tienes un módulo con un nombre muy largo:

i\_am\_a\_python\_module\_that\_is\_not\_very\_useful .

Sería enormemente molesto tener que escribir el nombre de este módulo cada vez que quisieras utilizarlo:

>>> import i\_am\_a\_python\_module\_that\_is\_not\_very\_useful  
>>> i\_am\_a\_python\_module\_that\_is\_not\_very\_useful.do\_something()  
  
I have done something that is not useful.  
>>> i\_am\_a\_python\_module\_that\_is\_not\_very\_useful.do\_something\_else()  
  
I have done something else that is not useful!!

En su lugar, puedes dar al módulo un nombre nuevo y más corto cuando lo importes, y luego simplemente utilizar ese nuevo nombre (como un apodo), de la siguiente manera:

>>> import i\_am\_a\_python\_module\_that\_is\_not\_very\_useful as notuseful  
>>> notuseful.do\_something()  
  
I have done something that is not useful.  
>>> notuseful.do\_something\_else()  
  
I have done something else that is not useful!!

### Assert

La palabra clave assert se utiliza para decir que un valor debe ser verdadero. Es otra forma de detectar errores y problemas en el código, normalmente en programas más avanzados (por eso no utilizamos assert en *Python para niños* ). Aquí tienes una simple afirmación assert :

>>> mynumber = 10  
>>> assert mynumber < 5  
Traceback (most recent call last):  
 File "<pyshell#1>", line 1, in <module>  
 assert mynumber < 5  
AssertionError

En este ejemplo, afirmamos que el valor de la variable mynumber es menor que 5. No lo es, por lo que Python muestra un error (llamado AssertionError ).

### ASYNC

La palabra clave async se utiliza para definir algo llamado *corrutina nativa .* Se trata de un concepto avanzado utilizado en la programación asíncrona (que consiste en hacer varias cosas en paralelo, o hacer cosas después de cierto tiempo).

### Espera

La palabra clave await también se utiliza para la programación asíncrona (similar a async ).

### Romper

La palabra clave break se utiliza para detener la ejecución de algún código. Podrías utilizar break dentro de un bucle for , así:

age = 10  
for x in range(1, 100):  
 print(f'counting {x}')  
 if x == age:  
 print('end counting')  
 break

Como aquí la variable age tiene el valor 10, este código imprimirá lo siguiente:

counting 1  
counting 2  
counting 3  
counting 4  
counting 5  
counting 6  
counting 7  
counting 8  
counting 9  
counting 10  
end counting

Una vez que el valor de la variable x alcance 10, el código imprimirá el texto "fin del recuento" y saldrá del bucle.

### Clase

La palabra clave class se utiliza para definir un tipo de objeto, como un vehículo, un animal o una persona. Las clases pueden tener una función llamada \_\_init\_\_ , que se utiliza para realizar todas las tareas que necesita un objeto de la clase cuando se crea. Por ejemplo, un objeto de la clase Car puede necesitar una variable color cuando se crea:

>>> class Car:  
 def \_\_init\_\_(self, color):  
 self.color = color  
  
>>> car1 = Car('red')  
>>> car2 = Car('blue')  
>>> print(car1.color)  
red  
>>> print(car2.color)  
blue

### Continúa

La palabra clave continue es una forma de "saltar" al siguiente elemento de un bucle para que no se ejecute el código restante del bloque de bucle. A diferencia de break , no saltamos fuera del bucle, sino que continuamos con el siguiente elemento. Por ejemplo, si tuviéramos una lista de elementos y quisiéramos saltar los elementos que empiezan por *b ,* podríamos utilizar el siguiente código:

>>> my\_items = ['apple', 'aardvark', 'banana', 'badger',  
 'clementine', 'camel']  
>>> for item in my\_items:  
 if item.startswith('b'):  
 continue  
 print(item)  
  
apple  
aardvark  
clementine  
camel

Primero creamos nuestra lista de elementos y luego utilizamos un bucle for para recorrer los elementos y ejecutar un bloque de código para cada uno. Si el elemento empieza por la letra *b ,* pasamos al siguiente elemento. Si no, imprimimos el elemento.

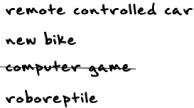
### DEF

La palabra clave def se utiliza para definir una función. Por ejemplo, podemos crear una función para convertir un número de años en el número equivalente de minutos:

>>> def minutes(years):  
 return years \* 365 \* 24 \* 60  
>>> minutes(10)  
5256000

### DEL

La palabra clave del se utiliza para borrar algo. Por ejemplo, si tenías en tu agenda una lista de cosas que querías para tu cumpleaños, pero luego cambiaste de opinión sobre una de ellas, podrías tacharla de la lista y añadir algo nuevo:



En Python, la lista original tendría este aspecto:

what\_i\_want = ['remote controlled car', 'new bike', 'computer game']

Podrías eliminar el juego de ordenador utilizando del y el índice del elemento que quieres eliminar. Luego podrías añadir el nuevo elemento con la función append :

del what\_i\_want[2]  
what\_i\_want.append('roboreptile')

Y luego imprimir la nueva lista:

print(what\_i\_want)  
['remote controlled car', 'new bike', 'roboreptile']

### Elif

La palabra clave elif se utiliza como parte de una sentencia if . Consulta la descripción de la palabra clave if para ver un ejemplo.

### Else

La palabra clave else se utiliza como parte de una sentencia if . Consulta la descripción de la palabra clave if para ver un ejemplo.

### Excepto

La palabra clave except se utiliza para detectar problemas en código bastante complicado.

### Finalmente

La palabra clave finally se utiliza para asegurarse de que, si se produce un error, se ejecute cierto código (normalmente para arreglar el desorden que ha dejado un trozo de código). Esta palabra clave es para programación más avanzada.

### Para

La palabra clave for se utiliza para crear un bucle de código que se ejecuta un determinado número de veces. Aquí tienes un ejemplo:

for x in range(0, 5):  
 print(f'x is {x}')

Este bucle for ejecuta el bloque de código (la sentencia print ) cinco veces, dando como resultado la siguiente salida:

x is 0  
x is 1  
x is 2  
x is 3  
x is 4

### De

Al importar un módulo, puedes importar sólo la parte que necesites utilizando la palabra clave from . Por ejemplo, el módulo turtle introducido en el [Capítulo 4](ch04.xhtml#ch04) tiene una clase llamada Turtle , que utilizamos para crear un objeto Turtle (que incluye el lienzo sobre el que se mueve la tortuga). Así es como importamos todo el módulo turtle y luego utilizamos la clase Turtle :

import turtle  
t = turtle.Turtle()

También podrías importar la clase Turtle por sí sola, y luego utilizarla directamente (sin referirte en absoluto al módulo turtle ):

from turtle import Turtle  
t = Turtle()

Podrías hacer esto para que la próxima vez que mires la parte superior de ese programa, puedas ver todas las funciones y clases que estás utilizando (lo que resulta especialmente útil en programas más grandes que importan muchos módulos). Sin embargo, si decides hacer esto, no podrás utilizar las partes del módulo que no hayas importado. Por ejemplo, el módulo time tiene funciones llamadas localtime y gmtime , pero si importas sólo localtime y luego intentas utilizar gmtime , obtendrás un error:

>>> from time import localtime  
>>> print(localtime())  
(2019, 1, 30, 20, 53, 42, 1, 30, 0)  
>>> print(gmtime())  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
NameError: name 'gmtime' is not defined

El mensaje de error name ’gmtime’ is not defined significa que Python no sabe nada de la función gmtime , porque no la has importado.

Si un módulo concreto tiene varias funciones que quieres utilizar, y no quieres referirte a ellas utilizando nombres de módulo (por ejemplo, time.localtime , o time.gmtime ), puedes importar todo lo que hay en el módulo utilizando un asterisco ( \* ), de esta forma:

>>> from time import \*  
>>> print(localtime())  
(2021, 1, 30, 20, 57, 7, 1, 30, 0)  
>>> print(gmtime())  
(2021, 1, 30, 13, 57, 9, 1, 30, 0)

Este formulario importa todo del módulo time , y ahora puedes referirte a las funciones individuales por su nombre.

### Global

La idea de *ámbito* en los programas se introduce en el [Capítulo 7 .](ch07.xhtml#ch07) El ámbito se refiere a la visibilidad de una variable. Si una variable se define fuera de una función, normalmente se puede ver (es visible) dentro de la función. En cambio, si la variable está definida dentro de una función, normalmente no se puede ver fuera de esa función.  La palabra clave global es una excepción a esta regla. Una variable definida como global puede verse en todas partes. He aquí un ejemplo:

>>> def test():  
 global a  
 a = 1  
 b = 2

¿Qué crees que ocurre cuando llamas a print(a) y luego a print(b) después de ejecutar la función test? La primera funcionará, pero la segunda mostrará un mensaje de error:

>>> test()  
>>> print(a)  
1  
>>> print(b)  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
NameError: name 'b' is not defined

La variable a ha pasado a tener ámbito global dentro de la función, por lo que es visible, incluso una vez finalizada la función; pero b sigue siendo visible sólo dentro de la función. (Debes utilizar la palabra clave global antes de establecer el valor de tu variable).

### Si

La palabra clave if se utiliza para tomar una decisión sobre algo. También puede utilizarse con las palabras clave else y elif (else if). Una sentencia if es una forma de decir: "Si algo es cierto, entonces realiza una acción de algún tipo". He aquí un ejemplo:

if toy\_price > 1000:  
 print('That toy is overpriced')  
elif toy\_price > 100:  
 print('That toy is expensive')  
else:  
 print('I can afford that toy')

Esta sentencia if dice que si el precio de un juguete es superior a 1.000 $, muestre un mensaje de que es caro; de lo contrario, si el precio del juguete es superior a 100 $, muestre un mensaje de que es caro. Si  ninguna de esas condiciones es cierta, debe mostrar el mensaje "Puedo permitirme ese juguete".

### Importar

La palabra clave import indica a Python que cargue un módulo para poder utilizarlo. Por ejemplo, el siguiente código indica a Python que utilice el módulo sys :

import sys

### En

La palabra clave in se utiliza en expresiones para ver si un elemento está dentro de una colección de elementos. Por ejemplo, ¿se puede encontrar el número 1 en una lista (una colección) de números?

>>> if 1 in [1,2,3,4]:  
 print('number is in the list')  
number is in the list

He aquí cómo averiguar si la cadena ’pants’ está en una lista de prendas de vestir:

>>> clothing\_list = ['shorts', 'undies', 'boxers', 'long johns',  
 'knickers']  
>>> if 'pants' in clothing\_list:  
 print('pants is in the list')  
 else:  
 print('where are my pants?')  
where are my pants?

### Es

La palabra clave is es un poco como el operador *igual a* ( == ), que se utiliza para saber si dos cosas son iguales (por ejemplo, 10 == 10 es verdadero y 10 == 11 es falso). Sin embargo, hay una diferencia fundamental entre is y == . Si estás comparando dos objetos (como listas), == puede devolver true , mientras que is puede que no (aunque pienses que los objetos son iguales). Se trata de un concepto avanzado de programación.

### Lambda

La palabra clave lambda se utiliza para crear funciones anónimas, o inline. Esta palabra clave se utiliza en programas más avanzados.

### No local

La palabra clave nonlocal se utiliza para incluir una variable en el ámbito de una función cuando se declara fuera de la función. Esta palabra clave se utiliza en programas más avanzados.

### No

Si algo es verdadero, la palabra clave not lo convierte en falso. Por ejemplo, si creamos una variable a y le asignamos el valor True , y luego imprimimos el valor de esta variable utilizando not , obtendremos el siguiente resultado:

>>> a = True  
>>> print(not a)  
False

Y de forma similar para un valor False , obtenemos True :

>>> b = False  
>>> print(not b)  
True

Esto no parece muy útil hasta que empiezas a utilizar la palabra clave en las sentencias if . Por ejemplo, para averiguar si un elemento no está en una lista, podríamos escribir algo así:

>>> clothing\_list = ['shorts', 'undies', 'boxers', 'long johns',  
 'knickers']  
>>> if 'pants' not in clothing\_list:  
 print('You really need to buy some pants')  
You really need to buy some pants

### O

La palabra clave or une dos condiciones en una sentencia (como una sentencia if ) para decir que al menos una de las condiciones debe ser cierta. Aquí tienes un ejemplo:

if dino == 'Tyrannosaurus' or dino == 'Allosaurus':  
 print('Carnivores')  
elif dino == 'Ankylosaurus' or dino == 'Apatosaurus':  
 print('Herbivores')

En este caso, si el valor de la variable dino es Tyrannosaurus o Allosaurus , el programa imprime Carnivores . Si es Ankylosaurus o Apatosaurus , el programa imprime Herbivores .

### Pasa

A veces, cuando estás desarrollando un programa, quieres escribir sólo pequeñas partes del mismo para probar cosas. El problema de hacer esto es que no puedes tener una sentencia if sin el bloque de código que debe ejecutarse si la condición de la sentencia if es verdadera. Tampoco puedes tener un bucle for sin el bloque de código que debe ejecutarse en el bucle. Por ejemplo, el siguiente código funciona perfectamente:

>>> age = 15  
>>> if age > 10:  
 print('older than 10')  
older than 10

Pero si no rellenas el bloque de código (el cuerpo) de la sentencia if , recibirás un mensaje de error:

>>> age = 15  
>>> if age > 10:  
File "<stdin>", line 2  
 ^  
IndentationError: expected an indented block after 'if' statement on line 1

Este es el mensaje de error que muestra Python cuando deberías tener un bloque de código después de una sentencia de algún tipo (ni siquiera te dejará escribir este tipo de código si estás utilizando IDLE). En casos como éste, puedes utilizar la palabra clave pass para escribir una sentencia pero sin proporcionar el bloque de código que la acompaña.

Por ejemplo, supongamos que quieres crear un bucle for con una sentencia if dentro. Tal vez aún no hayas decidido qué poner en la sentencia if : tal vez utilices la función print , o pongas una interrupción, o cualquier otra cosa. Por ahora, puedes utilizar pass ,  y el código seguirá funcionando (aunque todavía no haga exactamente lo que quieres). Aquí está de nuevo nuestra sentencia if , esta vez utilizando la palabra clave pass :

>>> age = 15  
>>> if age > 10:  
 pass

El siguiente código muestra otro uso de la palabra clave pass :

>>> for x in range(0, 7):  
>>> print(f'x is {x}')  
>>> if x == 4:  
 pass  
  
x is 0  
x is 1  
x is 2  
x is 3  
x is 4  
x is 5  
x is 6

Python sigue comprobando si la variable x contiene el valor 4 cada vez que ejecuta el bloque de código en el bucle, pero no hará nada en consecuencia, por lo que imprimirá todos los números del rango 0 a 7. Más adelante, podrías añadir el código en el bloque de la sentencia if , sustituyendo la palabra clave pass por otra, como break :

>>> for x in range(0, 7):  
 print(f'x is {x}')  
 if x == 4:  
 break  
  
x is 1  
x is 2  
x is 3  
x is 4

La palabra clave pass se utiliza normalmente cuando estás creando una función pero no quieres escribir todavía el código para ella.

### Elevar

La palabra clave raise puede utilizarse para provocar un error. Puede parecer algo extraño, pero en programación avanzada puede ser bastante útil.

### Devolver

La palabra clave return se utiliza para devolver un valor de una función. Por ejemplo, podrías crear una función para calcular el número de segundos que has estado vivo hasta tu último cumpleaños:

def age\_in\_seconds(age\_in\_years):  
 return age\_in\_years \* 365 \* 24 \* 60 \* 60

Cuando llames a esta función, el valor devuelto se puede asignar a otra variable o se puede imprimir:

>>> seconds = age\_in\_seconds(9)  
>>> print(seconds)  
283824000  
>>> print(age\_in\_seconds(12))  
378432000

### Prueba

La palabra clave try inicia un bloque de código que termina con las palabras clave except y finally . Juntos, estos bloques de código try / except / finally se utilizan para gestionar errores en un programa, como por ejemplo asegurarse de que el programa muestra un mensaje útil al usuario en lugar de un error poco amigable de Python. Son muy útiles en programas avanzados.

### Mientras que

La palabra clave while es un poco como for , salvo que un bucle for cuenta a través de un rango (de números), pero un bucle while sigue ejecutándose mientras una expresión es verdadera. Ten cuidado con los bucles while : si la expresión siempre es verdadera, el bucle nunca terminará (esto se llama *bucle infinito* ). He aquí un ejemplo:

>>> x = 1  
>>> while x == 1:  
 print('hello')

Si ejecutas este código, formará un bucle eterno, o al menos hasta que cierres la Shell de Python o pulses CTRL-C para interrumpirlo. Sin embargo, el código siguiente imprimirá "hola" nueve veces (cada vez sumando 1 a la variable x , hasta que x ya no sea menor que 10):

>>> x = 1  
>>> while x < 10:  
 print('hello')  
 x = x + 1

### Con

La palabra clave with se utiliza con un tipo especial de objeto para crear un bloque de código, de forma similar a las palabras clave try y finally , y luego gestiona los recursos de ese objeto. Esta palabra clave se utiliza en programas avanzados.

### Rendimiento

La palabra clave yield es un poco como return , salvo que se utiliza con una clase específica de objeto llamada *generador .* Los generadores crean valores a petición, por lo que, en ese sentido, la función range se comporta como un generador.