Programar en Python

Variables:

Las variables son una parte esencial de programación y se utilizan para almacenar todos los diferentes tipos de datos. Incluso podría decirse que son la piedra angular de la programación, y esto se debe a que permiten trabajar y manipular datos. Por lo tanto, es importante que pueda identificar variables y reconocer cómo se utilizan. Declarar variables en Python es muy sencillo. Lo único que debe hacer es declarar un nombre y asignarle un valor. La palabra variable se refiere a algo que se pueda cambiar. Para hacer esto en Python, para una variable que ya se ha declarado, solo necesita asignarla o lo declararla nuevamente. Vamos a analizar un ejemplo. Digamos que a la variable X se le ha asignado el valor de 10. Para cambiar esto, solo tiene que volver a declararlo, así tendrá el valor de 20. Los ejemplos hasta ahora se han basado en convenciones de nomenclatura simples como X, Y y Z. Cuando se trabaja en un proyecto con otros desarrolladores, será cada vez más difícil conocer qué significan estas variables o a qué se refieren. Como programador, escribirá una pérdida de código con el tiempo, y si han pasado unos meses, lo más probable es que no recuerde exactamente lo que el código se suponía que debía hacer. Utilizar variables genéricas como X e Y no da cualquier información sobre esa variable y dónde se utiliza. Dar nombres significativos a sus variables que tengan sentido en el contexto dado le permitirá a usted y otros programadores a comprender fácilmente lo que está pasando. Como programador, es importante comprender que los datos cambiarán a lo largo del ciclo de vida de su programa. Ya sea que esté recibiendo entrada del usuario a través de un formulario web o trabajando con variables dentro del código en sí, la función clave de la variable es mantener una referencia a algún valor. Ahora que comprende qué son las variables y sus funciones en Python, pasemos a una demostración práctica de las variables y cómo utilizarlas. Voy a demostrar cómo usar variables en Python, pero primero quiero hablar brevemente sobre convenciones de nomenclatura. Existen diferentes opciones disponibles para usted como para un desarrollador cuando se trata de nombrar sus variables. Una opción se denomina "CamelCase". La primera letra de la primera palabra es en minúsculas, y la primera letra de cada palabra después de eso no tiene espacios entre palabras. Por ejemplo, si tengo una variable que se denomina "my name" (mi nombre), voy a poner la M de "mi" en minúscula y el N del nombre en mayúsculas con el resto de las letras en minúsculas y sin espacio entre las palabras. Puedo tomar un enfoque diferente con "snake\_case". Al utilizar snake\_case, mantiene todo en minúsculas, pero utiliza un guión bajo entre palabras. Si quiero hacer la variable "mi nombre", "my\_name" sería el resultado de este enfoque. Aunque cuenta con diferentes opciones como desarrollador, es una buena idea ser coherente cuando crea variables a través de sus programas. Voy a borrar la pantalla para poder comenzar. Creo una variable en Python inicializando una variable y asignándole un valor. Todo lo que tengo que hacer es nombrar la variable. Por ejemplo, si escribo X es igual a 5, he declarado una variable y le asigné un valor. También puedo imprimir el valor de una variable al llamar las sentencias "print" y pasar el nombre de la variable, que en este caso es X, así que escribo "print X". Cuando ejecuto el programa, obtengo el valor de cinco, que son las asignaciones que proporcioné a la variable inicial. Volveré a borrar mi pantalla. Tiene varias opciones cuando decide declarar variables. Se puede declarar cualquier tipo diferente de variable en términos de valor. Por ejemplo, X podría ser igual a una cadena denominada "Hello" (Hola). Para ello, escribo X es igula a "hello". Entonces puedo imprimir el valor de nuevo, ejecutarlo y encuentro que el resultado es la palabra "hello". Detrás de escena, Python asigna automáticamente el tipo de datos. Obtendrá más información sobre este tema en un próximo video sobre tipos de datos. Puede declarar varias variables y asignarlas también a un solo valor. Por ejemplo, hacer "a", "b", y "c" igual a 10. Hago esto escribiendo "a" es igual a "b" es igual a "c" es igual a 10. Imprimo los tres valores por separado. Luego hago de nuevo clic en el botón "Run" (Ejecutar), encuentro que las tres de esas asignaciones tienen 10 como su valor. De nuevo, borro la pantalla antes de pasar al siguiente ejemplo. Otra opción que tiene es hacer varias asignaciones. Por ejemplo, escribo "a", "b", "c", separados por comas es igual, 1, 2, 3 también separados por comas. De esta manera, he asignado cada uno de esos valores a la letra correspondiente. "A" es igual a 1, "b" es igual a 2, "c" es igual a 3. Para probar esto, puedo imprimir las tres variables. Hago clic en" Ejecutar", y encontraré que los valores 1, 2, 3 corresponde a la declaración anterior. Otro punto importante que debe tener en cuenta son las asignaciones de variables y cómo puede cambiarlas. Una variable está sujeto a cambios. A lo largo del ciclo de vida de su programa, hará cambios en el valor o la asignación de la variable misma, así que debe saber cómo hacerlo. Vamos a analizar otro ejemplo. Escribo "a" es igual a 10, e imprimo ese valor. Luego, cambio el valor de uno a cinco, e imprimo ese valor también. Cuando hago clic en el botón "Run" (Ejecutar), "a" se imprime como 10 en la primera línea y es cinco en la línea a continuación porque se reasigna el valor. Por último, necesita saber cómo eliminar una variable. Mi variable, su valor es 10, y la imprimí, y luego en una nueva línea, escribo el comando "delete" (eliminar), o "del for short", seguido de un espacio y una letra "a", que representa mi variable. Luego imprimo la variable por medio de la función de impresión, y hago clic en el botón "Run" (Ejecutar). El valor nos da por primera vez 10 porque la variable aún existió, pero después de la eliminación, muestra un error diciendo que "a" no está definido. Acaba de ver convenciones de nomenclatura variable. Ahora ya sabe cómo declarar una variable y asignar su valor, y sabe cómo declarar cualquier tipo diferente de variable en términos de valor. Puede declarar varias variables y asignarlas como valor único, y puede hacer varias asignaciones. Por último, también aprendió a eliminar una variable. Esto nos lleva al final de este video. Ahora puede identificar variables y reconocer cómo utilizarlas en Python.

Tipos de datos básicos:

Los sistemas informáticos deben interpretar los diferentes valores de datos. En programación, los datos pueden venir en diferentes tipos. Al final de este video, podrá describir diferentes tipos de datos en Python. Un tipo de datos es un atributo asociado con un fragmento de datos que indican a un sistema informático cómo interpretar su valor. Saber qué tipos de datos utilizar garantiza que los datos se recogen en el formato preferido. También garantiza que el valor de cada propiedad es como se espera. Python ofrece tipos de datos raw para permitir datos que se asignarán a variables o constantes. Los cinco tipos principales clasificados como literales son numérico, secuencia, diccionario, booleano y conjunto. Algunos de estos tipos de datos se pueden ampliar. Por ejemplo, el tipo de datos numérico puede constar de tipos, entero, flotante y número complejo. Ahora hablemos de los tipos de datos con más detalle, comenzando con el numérico. En programación, debe decidir qué tipo se adaptará a sus necesidades. Por ejemplo, al trabajar con moneda, probablemente vaya a utilizar el tipo numérico flotante, ya que permite contar decimales. Para determinar un tipo de variable, Python también proporciona un nombre de función tipo, que proporcionará el tipo de clase basado en la variable que se pasa. Python ofrece tres opciones diferentes de tipos numéricos, que son enteros, flotantes y números complejos. La clase de entero representa cualquier número no fraccional, es decir, números enteros sin decimales. Estos números pueden ser positivos o negativos, por ejemplo, 10 o menos 10. Los flotantes son números que contienen decimales y se representan con la clase floats, algunos ejemplos podrían ser 10.5 o 6.7. La clase compleja se utiliza para representar números complejos que están formados por números reales e imaginarios, a es igual a 10 más 10j. A continuación, exploraremos los tipos de datos de secuencia. Secuencia de tipos de clústeres contenedores de tipos que contienen uno o más del mismo tipo en una lista ordenada. También se puede acceder a ellos en el índice de la secuencia. Python tiene tres tipos de secuencia diferentes, denominadas cadenas, listas y tuplas. Analicemos brevemente cada una de ellas, comenzando por las cadenas. Una cadena es una secuencia de caracteres encerrada entre comillas simples o dobles. Las cadenas se representan mediante la clase de cadena o str para abreviar. Las listas son una secuencia de uno o más tipos diferentes o similares. Son esencialmente un array y contienen cualquier tipo entre corchetes. Se puede acceder a cada elemento por su índice. Las tuplas son similares a las listas de muchas maneras. Contienen una secuencia ordenada de uno o más tipos, pero la principal diferencia es que son inmutables. Esto significa que los valores dentro de la tupla no se pueden modificar o cambiar. Las tuplas se representan por la clase tupla y mantienen los tipos de datos entre paréntesis. El siguiente tipo de datos es diccionario. El diccionario almacena datos en una estructura de objetos de valor clave. Se puede acceder a cada valor directamente por su clave. Los diccionarios también pueden almacenar cualquier tipo de datos. Por ejemplo, supongamos que declara una variable denominada ed y le asigna un diccionario, el diccionario contiene una agrupación de pares clave-valor. El primer par es a: 22, donde a es la clave y 22 es un valor. El segundo par es b: 44.4, donde b es una clave y 44.4 es el valor. Luego puede generar el valor de 22 accediendo a su clave, que es a. A continuación, exploraremos los tipos de datos booleanos que se representan simplemente como verdadero o falso. Combinado con operadores lógicos, los booleanos se utilizan para comprobar si una condición es verdadera o falsa. En este ejemplo, reviso el tipo de datos subyacente de los valores verdadero y falso. Se devuelve la clase bool, lo que significa que es booleano. El último tipo de datos es set, que es una colección desordenada y no indexada de valores no repetidos. Voy a mostrar un ejemplo de este tipo de datos. Supongamos que asigno un conjunto de cuatro elementos a la variable denominada conjunto de elementos. Luego compruebo el tipo del valor almacenado en la variable conjunto de elementos al pasarla a la función type. Python informa que el tipo de datos subyacente que contiene la variable set de ejemplo es un conjunto. En programación, el tipo de datos es un concepto importante. Las variables pueden almacenar datos de diferentes tipos, y diferentes tipos de datos pueden hacer cosas diferentes. Vamos a explorarlos con más detalle. Siempre que declare una variable en Python, el tipo de datos se le asigna automáticamente en función del valor de esa variable. Permítame demostrarlo al escribir una variable denominada a y asignándole un valor de 10. Para comprobar el tipo de datos que ha sido asignado por Python, selecciono "Print" y utilizo la función type. A continuación, paso la variable "a" como el parámetro y hago clic en "Ejecutar". Desde la salida en la terminal, puedo ver que se asignaba una clase de flotantes porque aparecía una posición decimal. Aquí hay otro ejemplo. Estoy usando la variable "b" y le asigno un valor decimal de 2.3. Para comprobar el tipo de datos asignado por Python, imprimo el tipo b y hago clic en "Ejecutar". Desde la salida en la terminal, puedo ver que se asigna una clase de flotantes porque ahí aparece un decimal. Se trata de una asignación diferente que el entero estándar. Estos son los tipos de datos numéricos que ofrece Python. Para declarar una variable como cadena, puse un texto entre comillas simples o dobles. De nuevo, ejecuto la sentencia "print" para el tipo de función y paso la variable "c" como el perímetro. Luego hago clic en "Ejecutar", las salidas y la terminal ahora muestra las clases int, float y str para la cadena. Esta secuencia también se puede aplicar a otros tipos de datos. Por ejemplo, puedo crear una lista de números mediante la variable "d" y asignándole los números 1, 2, 3, 4. Cuando ejecuto la sentencia print con un tipo de función y paso la variable "d" como parámetro, aparece la lista de clases luego de hacer clic en "Ejecutar". Cada vez que asigno un valor a una variable concreta, Python asigna automáticamente el tipo de dato correcto de esa variable. En este video ha aprendido sobre los diferentes tipos de datos en Python. Lo invito a que comience a experimentar con estos tipos de datos en su código de práctica.

**Tipo de fundición, un aspecto más profundo**

Existen algunos escenarios en los que el tipo de datos de un valor determinado se debe cambiar a otro tipo de datos.

Este proceso se conoce como fundición tipográfica.

Otra forma más informal de referirse al proceso es "conversión de tipo de datos".

El ejemplo más sencillo de convertir datos podría ser la siguiente comparación:

1

print(10 == 10)

En el fragmento de código anterior, le pregunto a Python si el número 10 es igual al número 10, y obtendré el valor booleano **True** impreso, confirmando que, efectivamente, son iguales.

¿Qué ocurre si hago esto?

1

print(10 == 10.00)

De nuevo, el valor booleano **True** aparece en la pantalla.

Ahora, usted podría decir que, bueno, **10** no es  ***técnicamente*** igual a **10.0** porque, uno podría argumentar, el primer número es un entero, y el segundo número es un flotante. Tendría razón, aunque son los mismos números, no son los mismos tipos de datos.

Sin embargo, Python realiza aquí lo que se conoce como **"conversión implícita de tipos"** .

Para comprender cómo funciona, modificaré un poco el ejemplo anterior. En lugar de pedirle a Python que compare los dos números y devuelva si son iguales o no, le pediré a Python que imprima el resultado de sumar estos dos números.

1

print(10 + 10.0)

El resultado impreso es **20.0**.

Esta salida me permite concluir que **cuando Python ejecuta operaciones con enteros y flotantes, convierte implícitamente el tipo de enteros en un flotante y luego completa la operación**.

Para comprenderlo mejor, puedo ampliar mi ejemplo anterior utilizando la función **type()**:

1

print(type(10 + 10.0))

La salida es **<class 'float'>**, entonces se confirma mi conclusión.

Permítame mostrarle un programa pequeño en Python, trabajando con números:

1

2

3

4

user\_num\_1 = input('First number is: ')

user\_num\_2 = input('Second number is: ')

user\_sum = user\_num\_1 + user\_num\_2

print(user\_sum)

Cuando ejecuto este programa, podría, por ejemplo, proporcionar el valor del primer número como 5 y el valor del segundo número también como 5, y esperar que el valor **user\_sum** impreso sea **10**.

Sin embargo, cuando hago exactamente eso, el número **55** se imprime en su lugar.

¿Por qué no funcionaba?

Bueno, la respuesta es bastante simple: todo lo que un usuario escribe, se convierte, mediante Python, al tipo de datos de cadena.

Esto significa que, aunque escribí números en estas dos entradas, lo que se guardó en **user\_num\_1** y **user\_num\_2** en realidad eran cadenas.

Efectivamente, es exactamente lo mismo que si hubiera hecho esto:

1

2

3

4

user\_num\_1 = "5"

user\_num\_2 = "5"

user\_sum = user\_num\_1 + user\_num\_2

print(user\_sum)

Esta vez, la salida de imprimir **user\_sum** sigue siendo **"55"**, pero esto no es ninguna sorpresa.

Para que mi código Python funcione como me propongo, necesito realizar la **conversión explícita de tipos**.

En otras palabras, tengo que convertir el valor de **"5"** al valor de **5**.

Aquí está mi código actualizado:

1

2

3

4

user\_num\_1 = input('First number is: ')

user\_num\_2 = input('Second number is: ')

user\_sum = float(user\_num\_1) + float(user\_num\_2)

print(user\_sum)

Lo que estoy haciendo aquí es asegurarme de que mi programa puede manejar e incluso aceptar flotantes como entradas, y continuar con su funcionamiento normal.

En otras palabras, me aseguro de que si un usuario proporciona el valor flotante de **5.5** como primer y segundo número al ejecutar el código anterior, la salida no arrojará un error. En su lugar, será el **11.0** esperado.

¿Y si decido enviar unas palabras al usuario para contarle lo que ha pasado?

Este es un intento de hacerlo:

1

2

3

4

num\_1 = input('First number is: ')

num\_2 = input('Second number is: ')

user\_sum = float(num\_1) + float(num\_2)

print("The sum of: " + num\_1 + " and " + num\_2 + " is " + user\_sum)

Si ejecuto el código anterior, generará el siguiente error:

**Traceback (most recent call last): File "<string>", line 4, in <module> TypeError: can only concatenate str (not "float") to str**

Lo que significa que no puedo concatenar una cadena y un flotante así. En otras palabras, aunque la **conversión de tipos implícita** de Python funciona cuando utilizo el operador **+** *en cadenas y enteros*, no funciona *en cadenas y flotantes*.

La solución es realizar la conversión de tipos explícita, de la siguiente manera.

1

2

3

4

n1 = input('First number is: ')

n2 = input('Second number is: ')

user\_sum = float(n1) + float(n2)

print("The sum of " + str(n1) + " and " + str(n2) + " is " + str(user\_sum))

Esta vez, la salida es: **The sum of 5.5 and 5.5 is 11.0**.

En Python, es fácil realizar conversiones explícitas, y a veces son muy útiles. Obtendrá más información sobre cómo funciona esto a medida que adquiera más experiencia en Python.

Construcciones en bucle: ejemplos prácticos

Esta lectura le presenta las diferentes construcciones de bucle en Python.

Bucle "For"

Los bucles de datos son una tarea bastante común en cualquier lenguaje de programación. El bucle for facilita el trabajo con cualquier tipo de secuencia en Python. Veamos algunos ejemplos de bucles for y las diferentes formas de utilizarlos.

123456

favorites = ['Creme Brulee', 'Apple Pie', 'Churros', 'Tiramisú', 'Chocolate Cake']

for dessert in favorites:

print('One of my favorite desserts is', dessert)

Restablecer

En el fragmento de código anterior, el bucle for itera sobre el contenido de la lista favorites e imprime una frase con el nombre del postre para cada elemento de la lista.

El bucle for se basa en el tamaño o la longitud de los elementos a iterar.

For Loop ProgramFor Loop Program

Bucle "While"

Por otro lado, un bucle while se basa en que una condición sea verdadera. Una vez que la condición ya no es verdadera, el bucle se detiene. La cantidad de veces que se ejecuta el bucle while no se conoce de antemano como ocurre con el bucle for.

Si toma el ejemplo de bucle anterior for y lo convierte en la alternativa de bucle while, terminará con algo así:

12345678

favorites = ['Creme Brulee', 'Apple Pie', 'Churros', 'Tiramisú', 'Chocolate Cake']

count = 0

while count < len(favorites):

print('One of my favorite desserts is', favorites[count]);

count += 1

Restablecer

Tome en cuenta que necesita declarar una variable counter. La variable counter se compara entonces con la longitud de la lista de favorites. A medida que se hace un bucle alrededor de los datos, se incrementa el counter. Una vez que la condición cont < len(favorites) ya no sea verdadera, el bucle se detendrá.

The graphic explains the flow of a for- loop.

# Función y ámbito de la variable

## Funciones y variables

Es esencial comprender los niveles de ámbito en Python y cómo se puede acceder a las cosas desde los cuatro niveles de ámbito diferentes. A continuación se presentan los cuatro niveles de ámbito y una breve explicación de dónde y cómo se utilizan.

### 1. Ámbito local

Ámbito local se refiere a una variable declarada dentro de una función. Por ejemplo, en el siguiente código, la variable **total** solo está disponible para el código dentro de la función **get\_total**. Cualquier cosa fuera de esta función no tendrá acceso a ella.

1

2

3

4

5

6

7

8

9

10

11

def get\_total(a, b):

    #local variable declared inside a function

    total = a + b;

    return total

print(get\_total(5, 2))

7

# Accessing variable outside of the function:

print(total)

NameError: name 'total' is not defined

EjecutarRestablecer

### 2. Ámbito cerrado

El ámbito cerrado se refiere a una función dentro de otra función o lo que se denomina **función anidada**.

En el siguiente código, agregé una función anidada denominada **double\_it** a la función **get\_total**.

Como **double\_it** está dentro del ámbito de la función **get\_total**, se puede acceder a la variable. Sin embargo, no se puede acceder a la variable cerrada dentro de la función **double\_it** desde dentro de la función **get\_total**.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

def get\_total(a, b):

    #enclosed variable declared inside a function

    total = a + b

    def double\_it():

        #local variable

        double = total \* 2

        print(double)

    double\_it()

    #double variable will not be accessible

    print(double)

    return total

EjecutarRestablecer

### 3. Ámbito global

El ámbito global es cuando se declara una variable fuera de una función. Esto significa que se puede acceder desde cualquier lugar.

En el siguiente código, agregué una variable global denominada **special**. A continuación se puede acceder desde ambas funciones **get\_total** y **double\_it**:

special = 5

def get\_total(a, b):

    #enclosed scope variable declared inside a function

    total = a + b

    print(special)

    def double\_it():

        #local variable

        double = total \* 2

        print(special)

    double\_it()

    return total

EjecutarRestablecer

### 4. Ámbito integrado

El ámbito integrado se refiere a las palabras clave reservadas que Python usa para sus funciones integradas, como **print, def, for, in** y así sucesivamente. Se puede acceder a las funciones con ámbito integrado en cualquier nivel.