Python

Variables, tipos de datos y operadores

CertiDevs

Índice de contenidos

1. Variables

Una **variable** en Python es un identificador que se utiliza para referenciar o almacenar un valor en la memoria del ordenador.

Las variables permiten a los programadores manipular datos y almacenarlos para su uso posterior.

Python es un lenguaje de programación de **tipado dinámico**, lo que significa que **no es necesario declarar el tipo de dato** de una variable antes de usarla.

2. Reglas para nombrar variables en Python

Hay algunas reglas para nombrar variables en Python:

- Las variables deben comenzar con una letra o un guion bajo (_).
- Las variables pueden contener letras, números y guiones bajos.
- Las variables distinguen entre mayúsculas y minúsculas, por lo que variable y Variable son diferentes.

3. Asignación de valores a variables

La asignación de valores a variables en Python se realiza utilizando el operador de igualdad (=). La sintaxis es la siguiente:

```
nombre_de_la_variable = valor
```

Por ejemplo:

```
nombre = "Juan Pérez"
edad = 30
precio = 19.99
is_student = True
```

4. Uso de variables en Python

Las variables en Python se pueden utilizar en **expresiones** y **operaciones matemáticas**, así como en la manipulación de **cadenas** de caracteres y en otras **estructuras de datos**.

4.1. Operaciones matemáticas

Se pueden realizar operaciones matemáticas utilizando variables:

```
a = 10
```

```
b = 5
suma = a + b
resta = a - b
multiplicacion = a * b
division = a / b
```

4.2. Concatenación de cadenas de caracteres

Las variables que contienen cadenas de caracteres se pueden combinar utilizando el operador de suma (+):

```
nombre = "Juan"
apellido = "Pérez"
nombre_completo = nombre + " " + apellido
```

4.3. Uso de variables en funciones

Las variables también se pueden utilizar en funciones para almacenar argumentos y resultados:

```
def suma(a, b):
resultado = a + b
return resultado
suma_de_numeros = suma(5, 7)
```

5. Tipos de datos básicos

Python es un lenguaje de programación de tipado dinámico, lo que significa que no es necesario declarar el tipo de datos de una variable antes de usarla.

Sin embargo, cada variable tiene un tipo de datos asociado. A continuación, se detallan los principales tipos de datos en Python:

5.1. Números enteros (int)

Los números enteros son números sin decimales. Estos pueden ser positivos, negativos o cero.

Ejemplo:

```
numero_entero = 42
```

5.2. Números de coma flotante (float)

Los números de coma flotante son números que incluyen decimales. Ejemplo:

```
numero_flotante = 3.141592
```

5.3. Números complejos (complex)

Los números complejos consisten en una parte real y una parte imaginaria, donde la parte imaginaria se representa con la letra 'j'. Ejemplo:

```
numero_complejo = 5 + 3j
```

5.4. Cadenas de caracteres (str)

Las cadenas de caracteres son secuencias de caracteres Unicode. Se pueden crear utilizando comillas simples (') o dobles ("") o incluso comillas triples. Ejemplo:

```
cadena = 'Hola, mundo!'
cadena = "Hola, mundo!"
```

Aquí hay un ejemplo de cómo crear una cadena de caracteres utilizando comillas triples:

```
cadena_multilinea = '''
    Esta es una cadena de caracteres
    que se extiende por
    múltiples líneas.
'''
```

También se pueden utilizar comillas triples dobles:

```
cadena_multilinea = """
    Esta es otra cadena de caracteres
    que también se extiende por
    múltiples líneas.
"""
```

En ambos casos, la cadena de caracteres resultante incluirá saltos de línea y espacios tal y como se hayan definido en el código fuente.

Uso de comillas triples en docstrings

En Python, las comillas triples se utilizan a menudo para crear "docstrings", que son comentarios de documentación asociados a funciones y clases.

Estos comentarios son útiles para proporcionar información sobre el propósito y funcionamiento de una función o clase.

Aquí hay un ejemplo de cómo usar comillas triples para escribir un docstring en una función:

```
def suma(a, b):
"""
Esta función toma dos números (a y b) como argumentos y devuelve su suma.
"""
    return a + b
```

5.5. Valores booleanos (bool)

Los valores booleanos representan verdadero (True) o falso (False). Ejemplo:

```
valor_booleano1 = True
valor_booleano2 = False
```

6. Tipos de datos compuestos

6.1. Listas (list)

Las listas son colecciones ordenadas y mutables de elementos. Se crean utilizando corchetes ([]). Ejemplo:

```
lista_de_numeros = [1, 2, 3, 4, 5]
```

6.2. Tuplas (tuple)

Las tuplas son colecciones ordenadas e inmutables de elementos. Se crean utilizando paréntesis (()). Ejemplo:

```
tupla_de_colores = ("rojo", "verde", "azul")
```

6.3. Conjuntos (set)

Los conjuntos son colecciones no ordenadas y sin elementos duplicados. Se crean utilizando llaves ({}) o la función set(). Ejemplo:

```
conjunto_de_frutas = {"manzana", "naranja", "uva"}
```

6.4. Diccionarios (dict)

Los diccionarios son colecciones no ordenadas de pares clave-valor. Se crean utilizando llaves ({}) y pares clave-valor separados por dos puntos. Ejemplo:

```
diccionario_de_precios = {"manzana": 1.5, "naranja": 0.99, "uva": 2.49}
```

7. Operadores

Los operadores son símbolos especiales en Python que realizan operaciones aritméticas, de comparación, lógicas, de asignación y otras operaciones entre operandos. A continuación, se detallan los diferentes tipos de operadores en Python:

7.1. Operadores aritméticos

Los operadores aritméticos realizan operaciones matemáticas básicas.

- Suma (+): suma dos números.
- Resta (-): resta el segundo número del primero.
- Multiplicación (*): multiplica dos números.
- División (/): divide el primer número por el segundo.
- Módulo (%): devuelve el resto de la división del primer número entre el segundo.
- Exponente (**): eleva el primer número a la potencia del segundo número.
- División entera (//): devuelve el cociente entero de la división del primer número entre el segundo.

7.2. Operadores de comparación

Los operadores de comparación comparan dos valores y devuelven un valor booleano (True o False).

- Igual (==): devuelve True si ambos valores son iguales.
- Distinto (!=): devuelve True si ambos valores son diferentes.
- Mayor que (>): devuelve True si el primer valor es mayor que el segundo.
- Menor que (`<`): devuelve True si el primer valor es menor que el segundo.
- Mayor o igual que (`>=`): devuelve True si el primer valor es mayor o igual que el segundo.
- Menor o igual que (`<=`): devuelve True si el primer valor es menor o igual que el segundo.

7.3. Operadores lógicos

Los operadores lógicos realizan operaciones lógicas entre valores booleanos.

- AND lógico (`and`): devuelve True si ambos operandos son True.
- OR lógico (`or`): devuelve True si al menos uno de los operandos es True.
- NOT lógico (`not`): devuelve True si el operando es False, y False si el operando es True.

7.4. Operadores de asignación

Los operadores de asignación se utilizan para asignar un valor a una variable.

- Asignación simple (`=`): asigna el valor del lado derecho al identificador del lado izquierdo.
- Asignación con suma (`+=`): suma el valor del lado derecho al identificador del lado izquierdo y asigna el resultado al identificador.
- Asignación con resta (`-=`): resta el valor del lado derecho al identificador del lado izquierdo y asigna el resultado al identificador.
- Asignación con multiplicación (`*=`): multiplica el valor del lado derecho por el identificador del lado izquierdo y asigna el resultado al identificador.
- Asignación con división (`/=`): divide el identificador del lado izquierdo por el valor del lado derecho y asigna el resultado al identificador.
- Asignación con módulo (`%=`): realiza el módulo del identificador del lado izquierdo por el valor del lado derecho y asigna el resultado al identificador.
- Asignación con exponente (`**=`): eleva el identificador del lado izquierdo a la potencia del valor del lado derecho y asigna el resultado al identificador.
- Asignación con división entera (`//=`): realiza la división entera del identificador del lado izquierdo por el valor del lado derecho y asigna el resultado al identificador.

7.5. Operadores de identidad

Los operadores de identidad comparan la identidad de dos objetos (no su valor) y devuelven un valor booleano.

- `is`: devuelve True si ambos objetos son el mismo objeto.
- `is not`: devuelve True si ambos objetos no son el mismo objeto.

7.6. Operadores de membresía

Los operadores de membresía verifican si un valor es miembro de una secuencia, como cadenas de caracteres, listas o tuplas, y devuelven un valor booleano.

- `in`: devuelve True si el valor especificado se encuentra en la secuencia.
- `not in`: devuelve True si el valor especificado no se encuentra en la secuencia.

7.7. Operadores a nivel de bits

Los operadores a nivel de bits realizan operaciones a nivel de bits entre números enteros.

- AND a nivel de bits (&): realiza una operación AND a nivel de bits entre los operandos.
- OR a nivel de bits (|): realiza una operación OR a nivel de bits entre los operandos.
- XOR a nivel de bits (^): realiza una operación XOR a nivel de bits entre los operandos.
- Desplazamiento a la izquierda (<<): desplaza los bits del número hacia la izquierda el número de posiciones especificado.
- Desplazamiento a la derecha (>>): desplaza los bits del número hacia la derecha el número de posiciones especificado.
- NOT a nivel de bits (~): invierte los bits del número (complemento a uno).