Hoja informativa: Introducción a Python

Variables

Las variables son contenedores de datos. Son similares a cajas con etiquetas, solo que guardan información, no cosas.

Las variables permiten a los desarrolladores y las desarrolladoras almacenar y reutilizar diversos datos en el código.

Cómo declarar una variable

Para declarar una variable:

1. Inventa y escribe el nombre de la variable, por ejemplo, `color`.
2. Escribe el operador de asignación `=`.
3. Escribe el valor que se almacenará en la variable, por ejemplo, un string: `"verde"`.

Todo en conjunto se ve así:

```
color = "verde"
# Declara una variable llamada color
# Asígnale el valor "verde"
```

No puedes declarar una variable sin valor, se le debe asignar algo.

Cómo nombrar variables

Hay ciertas reglas de nomenclatura para las variables. No todas las combinaciones de caracteres servirán.

Hay cuatro reglas principales:

✅ Puedes utilizar letras, dígitos y guiones bajos en los nombres de las variables.

❌ **No puedes** empezar el nombre con un dígito.

❌ **No puedes** tener un nombre que coincida con las palabras clave, o las palabras reservadas, del lenguaje.

❗ Los nombres de las variables distinguen entre mayúsculas y minúsculas. `hello` y `Hello` son dos variables diferentes.

¿Qué son las palabras clave o reservadas??

Las palabras clave, o reservadas, son palabras reservadas especiales que se utilizan para diversos fines. Ayudan al programa a entender qué es exactamente lo que quieres que haga.

Aquí está la lista completa de las palabras clave de Python: `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `False`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `None`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `True`, `try`, `while`, `with`, `yield`.

Asegúrate de no usar estas palabras para nombrar tus variables.

Cómo mostrar una variable

Necesitas la función `print()` para eso.

```
name = "Tom"

print(name)   # Muestra el valor de "name" en la consola
# Salida: Tom
```

Presta atención al orden en el que escribes tu código. No puedes utilizar primero una variable y luego declararla.

Python lee el código de arriba a abajo, línea por línea. Si no encuentra los datos necesarios, se detendrá y generará un error: `NameError: name "name" is not defined` (Error de nombre: el nombre "name" no está definido).

Cómo sobrescribir una variable

Puedes cambiar el valor de una variable. Para ello, declara la variable una vez más y asígnale un nuevo valor.

Así se ve en el código:

```
name = "Tom"  # Declara una variable y asígnale "Tom"
print(name)   # Salida: Tom

name = "Bob"  # Reasigna la variable
print(name)   # Salida: Bob
```

Variable como valor

Puedes asignar variables a otras variables.

Digamos que guardaste el nombre `"Tom"` en la variable `cat_name`. Luego, creaste otra variable para el ratón: `mouse_name`.

Tanto el gato como el ratón se llaman Tom, así que no es necesario volver a escribir `"Tom"`. En su lugar, puedes asignar `cat_name` a `mouse_name`.

```
cat_name = "Tom" # La variable cat_name contiene "Tom"
mouse_name = cat_name # Asigna cat_name a mouse_name
print(mouse_name) # Salida: Tom
```

Esto es más fácil y rápido que copiar y pegar o escribir datos duplicados manualmente.

Los desarrolladores y las desarrolladoras también usan esto para guardar el estado anterior de los datos. Las variables a menudo se sobrescriben, pero a veces necesitas saber exactamente qué estaba almacenado en la variable antes de ser cambiada.

Operadores

Para realizar acciones con datos en el código, se necesitan operadores. Los operadores son símbolos especiales que realizan diversas acciones. Pueden agregar variables, compararlas e incluso comprobar su contenido.

Es imposible escribir un programa sin operadores, ni siquiera uno muy básico.

Operadores aritméticos

Acción	Operador	Ejemplo	Resultado
Sumar	`+`	`3 + 4`	`7`
Restar	`-`	`3 - 4`	`-1`
Multiplicar	`*`	`3 * 4`	`12`
Elevar a la potencia	`**`	`3 ** 4`	`81`
Dividir con resto	`/`	`3 / 4`	`0.75`
Dividir sin resto	`//`	`3 // 4`	`0`
Calcular el módulo	`%`	`3 % 4`	`3`

Precedencia de los operadores lógicos

En programación, el orden de precedencia es el mismo que en matemáticas.

Las expresiones entre paréntesis se calculan primero, seguido de la exponenciación, y después la multiplicación y todos los tipos de división. La adición y la sustracción se realizan en último lugar.

¿Cuándo?	Operación	Operador
Primera prioridad	Expresiones agrupadas	`()`
Segunda prioridad	Exponenciación	`**`
Tercera prioridad	Multiplicación, división con residuo, división sin residuo, módulo	`*`, `/`, `//`, `%`
Cuarta prioridad	Adición y sustracción	`+` y `-`

Concatenación de strings

También puedes utilizar el operador `+` con strings. Cuando agregas strings, se unen y se convierten en un solo string.

> 📢 Se llama **concatenación** a unir strings con el operador `+`.

Por ejemplo, puedes concatenar dos partes de un saludo:

```
text_hello = "Buenos días, "   # Guardar un string en text_hello

team = "equipo"               # Guardar un string en team

print_str = text_hello + team  # Unir los strings

print(print_str).             # Salida: Buenos días, equipo
```

Al usar la concatenación, presta atención a los espacios. El operador de adición no se reemplazará por un espacio, por lo que debes poner uno manualmente dentro del string antes de las comillas; de lo contrario, las palabras se pegarán.

Operadores de comparación

En Python, puedes comparar números y variables numéricas usando operadores de comparación:

Comparación	Operador	Ejemplo
Menor que	`<`	`3 < 4`
Mayor que	`>`	`4 > 3`
Menor o igual que	`<=`	`3 <= 4`
Mayor o igual que	`>=`	`4 >= 3`
Igual a	`==`	`3 == 3`
No igual a	`!=`	`3 != 4`

Cuando comparas dos valores, utilizas una **expresión booleana**. Es una sentencia que puede ser **verdadera** (True) o **falsa** (False).

> 📌 Cada comparación es una expresión booleana.

Por ejemplo, la expresión booleana `3 < 4` es verdadera, por lo que Python generará `True` si muestras esta expresión.

```
result = 3 < 4  # Asigna una expresión booleana a la variable

print(result)   # Salida: True
```

Esto se debe a que `3` es, de hecho, menor que `4`. Por el contrario, `3 > 4` es una expresión booleana falsa. Esto contradice la verdad, así que Python devolverá `False`.

> 📌 El resultado de la expresión booleana es un tipo de dato lógico, o booleano, `bool`. Su valor puede ser `True` o `False`.

Otro punto importante: el operador de igualdad (`==`) se parece al operador de asignación (`=`). Sin embargo, realizan diferentes acciones: `==` compara valores y `=` asigna valores a variables. Asegúrate de no confundirlos.

Control de flujo

Cómo controlar el flujo del código

La palabra clave `if` (si) se utiliza para crear condiciones.

La palabra clave `if` va seguida de una expresión booleana que puede devolver `True` o `False`. Esta es la **condición**.

Si la condición es verdadera, se ejecuta el código que sigue a los dos puntos. Si es falsa, el código no se ejecutará.

```
if <condición>:
    <código que se ejecutará si la condición devuelve True>
```

Para establecer una condición, debes hacer lo siguiente:

- Escribe una sentencia `if`.
- Indica la condición, por ejemplo, `password_characters < 8`.
- Pon dos puntos `:` antes de pasar a la siguiente línea.
- Después de una indentación de cuatro espacios, escribe el código que debe ejecutarse. Por ejemplo, `print('¡La contraseña que ingresaste es demasiado corta!')`.

Este es el código para la condición de longitud de la contraseña. Si hay menos de 8 caracteres, muestra "¡La contraseña que ingresaste es demasiado corta!".

Intenta asignar otro número a la variable y observa qué sucede.

```
# En caso de que el usuario o la usuaria ingrese 7 caracteres
password_symbols = 7
if password_symbols < 8:
    print('¡La contraseña que ingresaste es demasiado corta!')
```

📌 El bloque de código anidado en la sentencia if debe tener una indentación de cuatro espacios.

Condición if-else

Necesitarás una sentencia `if-else`.

Añade otro bloque de código a `if`. Este bloque de código se ejecutará si la condición es falsa:

```
if <condición>:
    <el código que se ejecutará si la condición es verdadera>

else:
    <el código que se ejecutará si la condición es falsa>
```

Así es como se vería nuestro programa de contraseñas:

```
# En caso de que el usuario o la usuaria ingrese 7 caracteres
password_symbols = 7
if password_symbols < 8:
    print('¡La contraseña que ingresaste es demasiado corta!')
else:
    # Si la condición en la sentencia if es falsa, se ejecuta el código en la sentencia else
    print('¡Todo bien!')
```

Condición `elif`

Se puede simplificar este código. Puedes utilizar **una sentencia** `elif`, donde puedes probar otra condición.

Esta sentencia está anidada en `if`. Significa esto: "Si la condición en `if` no se cumple, y la condición en `elif` se cumple, ejecuta el bloque de código en `elif`".

En la sentencia `elif`, pones una condición que puede devolver `True` o `False`.

```
if password_symbols < 0:
    print('¡Error!')
elif password_symbols == 0:
    print('Ingresa la contraseña')
elif password_symbols < 8:
    print('¡La contraseña que ingresaste es demasiado corta!')
elif password_symbols > 15:
    print('¡La contraseña que ingresaste es demasiado larga!')
else:
    # Si no se activa ninguna de las condiciones anteriores, se ejecuta el código de la sentencia else
    print('¡Todo bien!')
```

Se traduce así: "Si la contraseña tiene menos de 0 caracteres, genera un error. Si tiene 0 caracteres, el usuario o la usuaria debe ingresar la contraseña. Si hay menos de 8 caracteres, la contraseña es demasiado corta. Si hay más de 15 caracteres, la contraseña es demasiado larga. Si nada de esto es verdadero, la contraseña es válida".

Tan pronto como se cumple una de las condiciones, todas las sentencias `elif` y `else` subsiguientes se descartan. Si tu contraseña tiene 0 caracteres, el programa mostrará "Ingresa la contraseña" e ignorará el resto de las sentencias.