Variables y Expresiones

Semestre 02, 2025

Introducción

Las variables y expresiones son los bloques fundamentales de cualquier programa.

Permiten almacenar datos y realizar operaciones sobre ellos para resolver problemas.

Datos

Un dato es una pieza básica de información que un programa puede manipular, como números, texto o valores lógicos.

Ejemplos:

- `42` (número entero)
- `3.14` (número de punto flotante)
- `"Hola Mundo"` (cadena de texto)
- `True` (valor booleano)

Variables

Es un nombre que hace referencia a un valor almacenado en memoria.

"Cajita" donde almacenamos información

Nos permite trabajar con datos sin conocer su valor de antemano.

Ejemplo

```
mensaje_inicial = "Hola Mundo!"
```

Aquí `mensaje_inicial` es una variable que almacena el texto `"Hola Mundo!"`.

Reglas para nombrar variables

1. Deben iniciar con una letra (a-z, A-Z).
2. Pueden contener letras, números y guiones bajos (`_`).
3. Son sensibles a mayúsculas y minúsculas (`contador` y `Contador` son diferentes).
4. No se pueden usar palabras reservadas de Python.

❌ Ejemplo inválido:

```python
50marimbas = "ensamble"
```

✅ Ejemplo válido:

```python
nombre_completo = "Ana Pérez"
```

Asignaciónes

La asignación reserva un espacio en memoria y almacena un valor.

Sintaxis: `variable = valor`

```
edad = 25
```

```

```

Si la variable existía, se actualiza su valor, de lo contrario se reserva el espacio y se guarda el valor.

Tipos de datos

Python no requiere declarar tipos explícitamente.

Loosely Typed

El tipo se asigna automáticamente según el valor.

Principales

- `int:` números enteros (e.g., `42`)
- `float:` números con decimales (e.g., `3.14`)
- `str:` cadenas de texto (e.g., `"Hola"`)
- `bool:` valores lógicos (`True` o `False`)

Determinar y cambiar el tipo de datos

Ver tipo de datos de una variable

```python
type(edad)
```

Convertir tipo a otro tipo de datos

```python
numero = 3.14
```

```
entero = int(numero)
```

Aquí `entero` tendrá el valor `3`.

Operadores

Aritméticos

Operador	Descripción	Ejemplo
`+`	Suma o concatenación	`x + y`
`-`	Resta	`x - y`
`*`	Multiplicación o repetición	`x * y`
`/`	División	`x / y`
`//`	División entera	`x // y`
`%`	Módulo (residuo)	`x % y`
`**`	Potencia	`x ** y`

Relacionales

Operador	Descripción	Ejemplo
`==`	Igual a	`x == y`
`!=`	Diferente de	`x != y`
`>`	Mayor que	`x > y`
`<`	Menor que	`x < y`

| `>=` | Mayor o igual que | `x >= y` |

| `<=` | Menor o igual que | `x <= y` |

Comparan valores y devuelven un resultado booleano.

Lógicos

| Operador | Descripción | Ejemplo |

| ———— | ———————————————— | ———————————— |

| `and` | Verdadero si ambos son True | `(x > 0) and (y > 0)` |

| `or` | Verdadero si al menos uno es True | `(x > 0) or (y > 0)` |

| `not` | Invierte el valor lógico | `not(x > 0)` |

Combinar condiciones y devuelven un resultado booleano.

Expresiones

Una **expresión** es la combinación de datos, variables y operadores que produce un resultado.

Ejemplo

```
x = 10

y = (2 * x) + (x / 5)

print(y)
```

Resultado: `25.0`

Orden de operaciones (PEMDASA)

Python sigue la jerarquía:

1. Paréntesis `()`
2. Exponentes `**`
3. Multiplicación y División `* / // %`
4. Suma y Resta `+ -`
5. De izquierda a derecha
6. Asignación

Ejemplo

```
resultado = (5 + 3) * 2 ** 2 / 4

print(resultado) # Salida: 8.0
```

Operaciones con cadenas

Concatenación (+)

```
nombre = "Ana"

saludo = "Hola, " + nombre

print(saludo) # Hola, Ana
```

Repetición (*)

```
print("🐍" * 3) # 🐍🐍🐍
```

Ingreso de datos

Usamos `input()` para interactuar con el usuario:

```
nombre = input("¿Cuál es tu nombre? ")

print("Hola,", nombre)
```

Comentarios

- Los comentarios ayudan a documentar el código.
- Empiezan con `#`.
- No son necesarios pero si útiles.

```
# Este programa calcula el área de un triángulo

base = 5

altura = 10

area = (base * altura) / 2

print(area)
```

```
Dataview (inline field '='): Error:
-- PARSING FAILED ------------------------------------------------
-
```

```
> 1 | =
    | ^

Expected one of the following:

'(', 'null', boolean, date, duration, file link, list ('[1, 2,
3]'), negated field, number, object ('{ a: 1, b: 2 }'), string,
variable
```