

Ayudantías IPS


Impartidas por Diego

Variables

Una variable es un espacio de almacenamiento en la memoria de un programa que se utiliza para guardar un valor que puede cambiar durante la ejecución del mismo.

En términos generales, una variable:

- Tiene un nombre (identificador) que se usa para referirse a ella.
- Tiene un valor que puede ser de diferentes tipos (como números, texto, booleanos, etc.).
- Puede ser modificada (salvo en casos específicos como las constantes).
- Permite trabajar con datos de forma dinámica.



```
1  # Variable de tipo entero
2  edad = 25
3
4  # Variable de tipo cadena de texto
5  nombre = "Juan"
6
7  # Variable de tipo decimal (float)
8  precio = 19.99
9
10 # Variable booleana
11 es_mayor = True
12
```

Tipos de datos

Entero (int)

Representa números enteros, positivos o negativos, sin decimales.

Flotante (float)

Representa números reales con parte decimal. Se usan para valores que requieren mayor precisión, como precios o medidas.

Cadena de texto (str)

Es una secuencia de caracteres. Se usa para representar texto, como nombres, mensajes, direcciones, etc.

Booleano (bool)

Tiene solo dos valores posibles: True o False. Se utiliza en decisiones lógicas y estructuras condicionales.

Lista (list)

Es una colección ordenada y mutable (modificable) de elementos. Puede contener distintos tipos de datos, incluso otras listas.

Tuple (tuple)

Es una colección ordenada pero inmutable, es decir, sus valores no se pueden cambiar después de crearla.

Conjunto (set)

Colección desordenada de elementos únicos. No permite duplicados y no garantiza el orden.

Diccionario (dict)

Es una colección de pares clave: valor. Permite acceder rápidamente a un valor a través de su clave. Muy útil para representar objetos o registros.

```
1  # Tipos de datos en Python
2
3  # Entero (int)
4  edad = 25
5
6  # Flotante / Decimal (float)
7  precio = 19.99
8
9  # Cadena de texto (str)
10 nombre = "Juan"
11 saludo = 'Hola mundo'
12
13 # Booleano (bool)
14 es_mayor = True
15 tiene_permiso = False
16
17 # Lista (list)
18 frutas = ["manzana", "pera", "uva"]
19 numeros = [1, 2, 3, 4, 5]
20
21 # Tupla (tuple)
22 coordenadas = (10.5, 20.3)
23 colores = ("rojo", "verde", "azul")
24
25 # Conjunto (set)
26 dias = {"lunes", "martes", "miércoles"}
27
28 # Diccionario (dict)
29 persona = {
30     "nombre": "Ana",
31     "edad": 30,
32     "ciudad": "Santiago"
33 }
34
```

Operadores aritméticos

Suma: +

Resta: -

Multiplicación: *

División: /

Divide el primer valor por el segundo y devuelve un número decimal (float).

División entera: //

Divide el primer valor por el segundo y devuelve solo la parte entera del resultado.

Módulo: %

Devuelve el residuo de la división entre dos valores.

Potencia: **

Eleva el primer valor a la potencia del segundo.

```
1  # Tipos de datos en Python
2
3  # Entero (int)
4  edad = 25
5
6  # Flotante / Decimal (float)
7  precio = 19.99
8
9  # Cadena de texto (str)
10 nombre = "Juan"
11 saludo = 'Hola mundo'
12
13 # Booleano (bool)
14 es_mayor = True
15 tiene_permiso = False
16
17 # Lista (list)
18 frutas = ["manzana", "pera", "uva"]
19 numeros = [1, 2, 3, 4, 5]
20
21 # Tupla (tuple)
22 coordenadas = (10.5, 20.3)
23 colores = ("rojo", "verde", "azul")
24
25 # Conjunto (set)
26 dias = {"lunes", "martes", "miércoles"}
27
28 # Diccionario (dict)
29 persona = {
30     "nombre": "Ana",
31     "edad": 30,
32     "ciudad": "Santiago"
33 }
34
```

Operadores de Comparación

Igual que: ==

Compara si dos valores son iguales.

Distinto que: !=

Compara si dos valores son diferentes.

Mayor que: >

Verifica si el valor de la izquierda es mayor que el de la derecha.

Menor que: <

Verifica si el valor de la izquierda es menor que el de la derecha.

Mayor o igual que: >=

Verifica si el valor de la izquierda es mayor o igual que el de la derecha.

Menor o igual que: <=

Verifica si el valor de la izquierda es menor o igual que el de la derecha.

```
1  # Operadores de comparación en Python
2
3  a = 10
4  b = 5
5
6  # Igual que
7  igual = a == b
8  print("Igual que:", igual)  # False
9
10 # Distinto que
11 distinto = a != b
12 print("Distinto que:", distinto)  # True
13
14 # Mayor que
15 mayor = a > b
16 print("Mayor que:", mayor)  # True
17
18 # Menor que
19 menor = a < b
20 print("Menor que:", menor)  # False
21
22 # Mayor o igual que
23 mayor_igual = a >= b
24 print("Mayor o igual que:", mayor_igual)  # True
25
26 # Menor o igual que
27 menor_igual = a <= b
28 print("Menor o igual que:", menor_igual)  # False
29
```

Operadores de Lógicos

Y lógico (and)

Devuelve **True** si ambos operandos son verdaderos. De lo contrario, devuelve **False**.

O lógico (or)

Devuelve **True** si al menos uno de los operandos es verdadero. Si ambos son falsos, devuelve **False**.

Negación lógica (not)

Invierte el valor de verdad de un operando. Si es **True**, lo convierte en **False** y viceversa.

```
1  # Operadores lógicos en Python
2
3  a = True
4  b = False
5
6  # Y lógico (and)
7  y_logico = a and b
8  print("Y lógico (and):", y_logico)  # False
9
10 # O lógico (or)
11 o_logico = a or b
12 print("O lógico (or):", o_logico)  # True
13
14 # Negación lógica (not)
15 negacion = not a
16 print("Negación lógica (not):", negacion)  # False
17
```

Condicionales

✓ if (si)

Se usa para ejecutar un bloque de código **sólo si** una condición se cumple (es verdadera).

🔄 elif (sino si)

Se utiliza para verificar otra condición **si la anterior no se cumple**. Puedes tener varios **elif**.

✗ else (sino)

Se ejecuta **si ninguna de las condiciones anteriores se cumple**. Es el "por defecto".

```
1
2 # Ejemplo de condicionales en Python
3
4 edad = 18
5
6 # if: ejecuta el bloque si la condición es verdadera
7 if edad < 18:
8     print("Eres menor de edad")
9
10 # elif: se evalúa si la condición anterior fue falsa
11 elif edad == 18:
12     print("Tienes 18 años, justo la mayoría de edad")
13
14 # else: se ejecuta si ninguna condición anterior se cumple
15 else:
16     print("Eres mayor de edad")
17
```

Ciclos

for

Se utiliza para **iterar** sobre una secuencia (como una lista, tupla, cadena o rango). Ejecuta un bloque de código por cada elemento de la secuencia.

while

Ejecuta un bloque de código **mientras** una condición sea verdadera. Es útil cuando no sabes cuántas veces se repetirá el ciclo.

Ambos ciclos permiten repetir acciones, pero se usan en contextos diferentes:

- **for** → cuando **sabes cuántas veces** o **sobre qué** estás iterando.
- **while** → cuando **la repetición depende de una condición lógica**.

```
1 # Ciclo FOR: recorre una secuencia de elementos
2 print("Ejemplo de ciclo for:")
3 for i in range(5): # range(5) genera los números del 0 al 4
4     print("Iteración:", i)
5
6 # Ciclo WHILE: se repite mientras la condición sea verdadera
7 print("\nEjemplo de ciclo while:")
8 contador = 0
9 while contador < 5:
10     print("Contador:", contador)
11     contador += 1 # incrementa el valor en cada vuelta
12
```