Variables y Expresiones

Algoritmos y Programación Básica (CC2005) - 2026

Variables y Expresiones

Semestre 02, 2026

Datos

Un dato es una pieza básica de información que un programa puede manipular.

- Números
- Texto
- Valores lógicos

Ejemplos:

- `42`
- `3.14`
- `"Hola"`
- `True`

Cada uno representa un tipo distinto de información. Y ese tipo no es un detalle: decide qué se puede hacer con el dato.

Tipos de Datos

El tipo de dato define qué clase de información almacena una variable y qué operaciones tienen sentido sobre ella.

Todo valor en Python tiene un tipo. No existe un dato "suelto" sin tipo.

Por qué importan

El mismo operador hace cosas distintas según el tipo de los operandos.

```
3 + 2        # 5      suma de números
"3" + "2"    # "32"   unión de textos
"□□ * 3     # "□□□□□□repetición de texto
3 * 2        # 6      multiplicación
```

El símbolo `+` no significa "sumar": significa "combinar", y cada tipo define cómo se combina.

Cuando los tipos no coinciden

```
"10" + 5
# TypeError: can only concatenate str (not "int") to str
```

Python no adivina. Si mezclamos texto con número, se detiene y avisa.

Python usa tipado dinámico

- No declaramos el tipo: Python lo deduce del valor.
- Una misma variable puede cambiar de tipo durante el programa.
- El tipo se verifica al ejecutar, no antes.

```
dato = 10        # ahora es int
dato = "diez"    # ahora es str, y es válido
```

En lenguajes como Java o C hay que declarar el tipo y no se puede cambiar. Esa es la diferencia entre tipado dinámico y tipado estático.

Los Cuatro Tipos Básicos

Tipo	Guarda	Ejemplo
`int`	Números enteros	`42`
`float`	Números decimales	`3.14`
`str`	Texto	`"Hola"`
`bool`	Verdadero o falso	`True`

```
numero = 10
precio = 3.5
nombre = "Luis"
es_mayor = True
```

Enteros (int)

Representan números sin parte decimal: positivos, negativos o cero.

```
edad = 20
temperatura = -5
cero = 0
poblacion = 17_000_000   # el guion bajo solo ayuda a leer
```

Sin límite de tamaño

A diferencia de otros lenguajes, en Python un entero puede crecer tanto como la memoria lo permita.

```
print(2 ** 100)
# 1267650600228229401496703205376
```

División entera y residuo

```
7 / 2     # 3.5   división normal, siempre devuelve float
7 // 2    # 3     división entera, descarta la parte decimal
7 % 2     # 1     residuo de la división
```

El operador `%` es muy útil: `numero % 2 == 0` dice si un número es par.

Decimales (float)

Representan números con parte decimal.

```
precio = 19.99
altura = 1.75
negativo = -0.5
cientifico = 1.5e3    # 1500.0
```

La división siempre da float

```
10 / 5      # 2.0, no 2
type(10 / 5)  # <class 'float'>
```

La precisión no es exacta

Los decimales se guardan en binario y muchos valores no tienen representación exacta.

```
0.1 + 0.2          # 0.30000000000000004
0.1 + 0.2 == 0.3   # False
```

Por eso no se comparan decimales con `==`. Se compara si la diferencia es lo bastante pequeña:

```
abs((0.1 + 0.2) - 0.3) < 0.0001   # True
```

Texto (str)

Una cadena es una secuencia de caracteres escrita entre comillas.

```
nombre = "Ana"
apellido = 'Pérez'          # comillas simples o dobles, da igual
vacia = ""                  # una cadena vacía es válida
frase = "Dijo: \"hola\""    # comilla escapada
```

Varias líneas

```
parrafo = """Primera línea
Segunda línea"""
```

Los números como texto no son números

```
"25"      # es texto, no se le puede sumar
25        # es un número
"25" + 1  # TypeError
```

Esta es la confusión más común al empezar, y aparece siempre que se lee datos del usuario.

Booleanos (bool)

Solo admite dos valores: `True` y `False`. Se escriben con mayúscula inicial.

```
es_mayor = True
termino = False
```

Nacen de las comparaciones

Toda comparación produce un booleano.

```
20 >= 18      # True
5 == 5        # True
"a" == "b"    # False
```

Un booleano es un número disfrazado

```
True + True    # 2
int(True)      # 1
int(False)     # 0
```

Valores que se consideran falsos

Python trata como falso al cero, al vacío y a `None`:

```
bool(0)        # False
bool("")       # False
bool([])       # False
bool(None)     # False
bool(42)       # True
bool("hola")   # True
```

Verificar el Tipo

La función `type()` devuelve el tipo de un valor.

```
numero = 10
print(type(numero))
# <class 'int'>
```

Otra forma

`isinstance()` responde con un booleano y es lo que se usa en programas reales.

```
isinstance(numero, int)    # True
isinstance(numero, str)    # False
```

Conversión de Tipos

Convertir es transformar un valor de un tipo a otro. También se le llama casting.

| Función | Convierte a | |----------|----------| | `int()` | Entero | | `float()` | Decimal | | `str()` | Texto | | `bool()` | Booleano |

De decimal a entero

```
numero = 3.14
entero = int(numero)   # 3
```

La conversión trunca, no redondea: `int(3.99)` da 3, no 4. Para redondear existe `round()`.

De texto a número y al revés

```
texto = "25"
numero = int(texto)      # 25

edad = 20
texto_edad = str(edad)   # "20"
```

Conversiones que fallan

```
int("hola")    # ValueError: invalid literal for int()
int("3.5")     # ValueError: hay que pasar por float primero
float("3.5")   # 3.5, correcto
```

`int()` solo acepta texto que represente un entero. Si viene con punto decimal, primero `float()` y después `int()`.

Leer Datos del Usuario

La función `input()` muestra un mensaje y devuelve lo que el usuario escribe.

```
nombre = input("¿Cómo te llamas? ")
print("Hola", nombre)
```

input() siempre devuelve texto

Aunque el usuario escriba 25, lo que llega es `"25"`.

```
edad = input("¿Cuántos años tienes? ")
print(type(edad))     # <class 'str'>

edad + 1              # TypeError
```

La solución es convertir

```
edad = int(input("¿Cuántos años tienes? "))
print(type(edad))     # <class 'int'>

faltan = 18 - edad
print("Te faltan", faltan, "años")
```

Esta es la razón práctica por la que los tipos importan desde el primer programa.

Resumen de Tipos

Tipo	Ejemplo	Convertir con	Cuidado
`int`	`42`	`int()`	Trunca al convertir desde float
`float`	`3.14`	`float()`	No compares con `==`
`str`	`"Hola"`	`str()`	`"25"` no es 25
`bool`	`True`	`bool()`	Se escribe con mayúscula

Más adelante veremos tipos que agrupan varios datos: listas, tuplas, diccionarios y conjuntos.

Problemática

Ya sabemos qué tipos de datos existen. Ahora necesitamos guardarlos en algún lado.

- Queremos resolver problemas con datos.
- Los datos cambian.
- No siempre conocemos los valores antes de ejecutar el programa.

Calcular el promedio de un estudiante:

- El nombre cambia.
- Las notas cambian.
- El resultado cambia.

Es necesaria una forma de guardar información.

Variables

Una variable es un nombre que hace referencia a un valor almacenado en memoria.

Permite trabajar con datos sin conocer su valor de antemano.

Una variable es como:

- Una caja etiquetada.
- Un contenedor con nombre.
- Un espacio donde guardamos algo.

Ejemplo

```
mensaje = "Hola Mundo"
```

- `mensaje` es la variable.
- `Hola Mundo` es el valor almacenado.

La variable no guarda el valor: lo apunta

En Python el valor vive en memoria y la variable es una etiqueta que apunta hacia él.

```
a = 10
b = a      # b apunta al mismo valor
a = 20     # a apunta a otro valor; b sigue en 10
print(b)   # 10
```

Asignación

Asignar significa guardar un valor en una variable.

`variable = valor`

El signo `=` no es igualdad: es una orden de guardar. Se lee de derecha a izquierda.

Ejemplo

```
edad = 18
```

La variable `edad` ahora contiene el valor 18.

Reasignación

```
edad = 18
edad = 19
```

El valor anterior se reemplaza. La variable siempre guarda el valor más reciente y solo puede almacenar un valor a la vez.

Usar la variable en su propia asignación

```
contador = 0
contador = contador + 1   # ahora vale 1
contador += 1             # forma abreviada, ahora vale 2
```

Primero se calcula el lado derecho y después se guarda el resultado.

Asignación múltiple

```
x, y = 1, 2
a = b = 0
```

Reglas para Nombrar Variables

- Deben iniciar con una letra o guion bajo.
- Pueden contener letras, números y `_`.
- No pueden contener espacios ni signos.
- Son sensibles a mayúsculas y minúsculas: `edad` y `Edad` son distintas.
- No pueden usar palabras reservadas.

Incorrecto

```
50marimbas = "ensamble"   # empieza con número
nombre completo = "Ana"   # tiene un espacio
class = "CC2005"          # palabra reservada
```

Correcto

```
nombre_completo = "Ana Pérez"
_privado = 1
notaFinal = 85
```

Palabras reservadas

Son palabras que el lenguaje ya usa y no se pueden reutilizar como nombres.

```
False   None    True    and     as      assert
break   class   continue def    del     elif
else    except  for     from    global  if
import  in      is      lambda  not     or
pass    return  try     while   with    yield
```

Convenciones

- En Python se usa snake_case: `nombre_completo`, `nota_final`.

- El nombre describe el contenido: `promedio_notas` comunica más que `pn` o `x`.
- Las constantes se escriben en mayúsculas: `PI = 3.1416`.

Expresiones

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

Ejemplo

```
x = 10
y = (2 * x) + 5
```

`y` es el resultado de una expresión.

Expresión y sentencia

- Una expresión produce un valor: `2 + 3`, `edad >= 18`.
- Una sentencia ejecuta una acción: `edad = 18`, `print(edad)`.

Toda asignación contiene una expresión del lado derecho.

Operadores Aritméticos

| Operador | Significado | Ejemplo | Resultado | | -------- | ---------- | ---------- | --------- | | `+` | Suma | `5 + 2` | `7` | | `-` | Resta | `5 - 2` | `3` | | `*` | Multiplicación | `5 * 2` | `10` | | `/` | División | `5 / 2` | `2.5` | | `//` | División entera | `5 // 2` | `2` | | `%` | Residuo | `5 % 2` | `1` | | `**` | Potencia | `5 ** 2` | `25` |

Ejemplo

```
a = 5
b = 2
resultado = a + b
```

Ojo con la división

```
10 / 2    # 5.0  siempre float
10 // 2   # 5    entero
10 / 0    # ZeroDivisionError
```

Operadores Relacionales

| Operador | Significado | | -------- | --------------- | | `==` | Igual | | `!=` | Diferente | | `>` | Mayor | | `<` | Menor | | `>=` | Mayor o igual | | `<=` | Menor o igual |

Siempre devuelven un valor booleano.

Ejemplo

```
edad = 20
edad >= 18
```

Resultado: `True`

Comparar textos

Las cadenas se comparan en orden alfabético, según el código de cada carácter.

```
"ana" == "Ana"    # False, las mayúsculas importan
"a" < "b"         # True
```

Operadores Lógicos

Sirven para crear conexiones y formar expresiones más compuestas.

- `and` → verdadero si ambos lados son verdaderos.
- `or` → verdadero si al menos un lado es verdadero.
- `not` → invierte el valor.

Tablas de verdad

| A | B | `A and B` | `A or B` | |---------|---------|-----------|----------| | `True` | `True` | `True` | `True` | | | `True` | `False` | `False` | `True` | | `False` | `True` | `False` | `True` | | `False` | `False` | `False` | `False` |

Ejemplo

```
edad = 20
tiene_credencial = True
edad >= 18 and tiene_credencial
```

Resultado: `True`

Encadenar comparaciones

Python permite escribir rangos de forma natural:

```
0 <= nota <= 100
```

Orden de Operaciones

PEMDASA

1. Paréntesis
2. Exponentes
3. Multiplicación y división (de izquierda a derecha)
4. Adición y sustracción (de izquierda a derecha)
5. Comparaciones
6. Operadores lógicos
7. Asignación

Ejemplo

```
resultado = (5 + 3) * 2     # 16, primero el paréntesis
otro = 5 + 3 * 2            # 11, primero la multiplicación
```

Ante la duda, usa paréntesis: cuestan nada y evitan errores.

Operaciones con Cadenas

Concatenación

Concatenar es unir dos cadenas.

```
nombre = "Ana"
saludo = "Hola " + nombre

print(saludo)
# Hola Ana
```

Solo se concatena texto con texto. Para incluir un número hay que convertirlo:

```
edad = 20
"Tengo " + str(edad) + " años"
```

f-strings

Es la forma moderna y recomendada de armar texto con variables.

```
nombre = "Ana"
edad = 20
print(f"Hola {nombre}, tienes {edad} años")
```

Dentro de las llaves puede ir cualquier expresión: `f"El doble es {edad * 2}"`.

Repetición

```
print("□□ * 3)

# □□□□□□
```

¿No era lo que esperaban?

Los tipos de datos son importantes ya que dictaminan el comportamiento de los operandos dada una operación.

Problemas Comunes

- Usar variables sin definir.
- Confundir `=` (asignar) con `==` (comparar).
- Mezclar tipos incompatibles.
- Olvidar convertir lo que devuelve `input()`.
- Comparar decimales con `==`.

Mezclar tipos

```
numero = "10"
numero + 5
# TypeError: can only concatenate str (not "int") to str
```

Usar una variable antes de definirla

```
print(total)
# NameError: name 'total' is not defined
```

Confundir asignación con comparación

```
edad = 18     # guarda 18 en edad
edad == 18    # pregunta si edad vale 18
```

Resumen

- Todo valor tiene un tipo, y el tipo decide qué operaciones tienen sentido.
- Los cuatro tipos básicos son `int`, `float`, `str` y `bool`.
- Python usa tipado dinámico: deduce el tipo del valor y permite cambiarlo.
- `type()` verifica el tipo; `int()`, `float()`, `str()` y `bool()` convierten entre tipos.
- `input()` siempre devuelve texto: hay que convertirlo para hacer cálculos.
- Una variable es un nombre que apunta a un valor en memoria; `=` guarda, no compara.
- Los nombres siguen reglas (no empiezan con número, no son palabras reservadas) y la convención snake_case.
- Una expresión combina datos, variables y operadores para producir un resultado.
- Ante dudas de precedencia, usa paréntesis.