

Curso Python 2025 Día 3: Condicionales y Secuencias



Hoy:

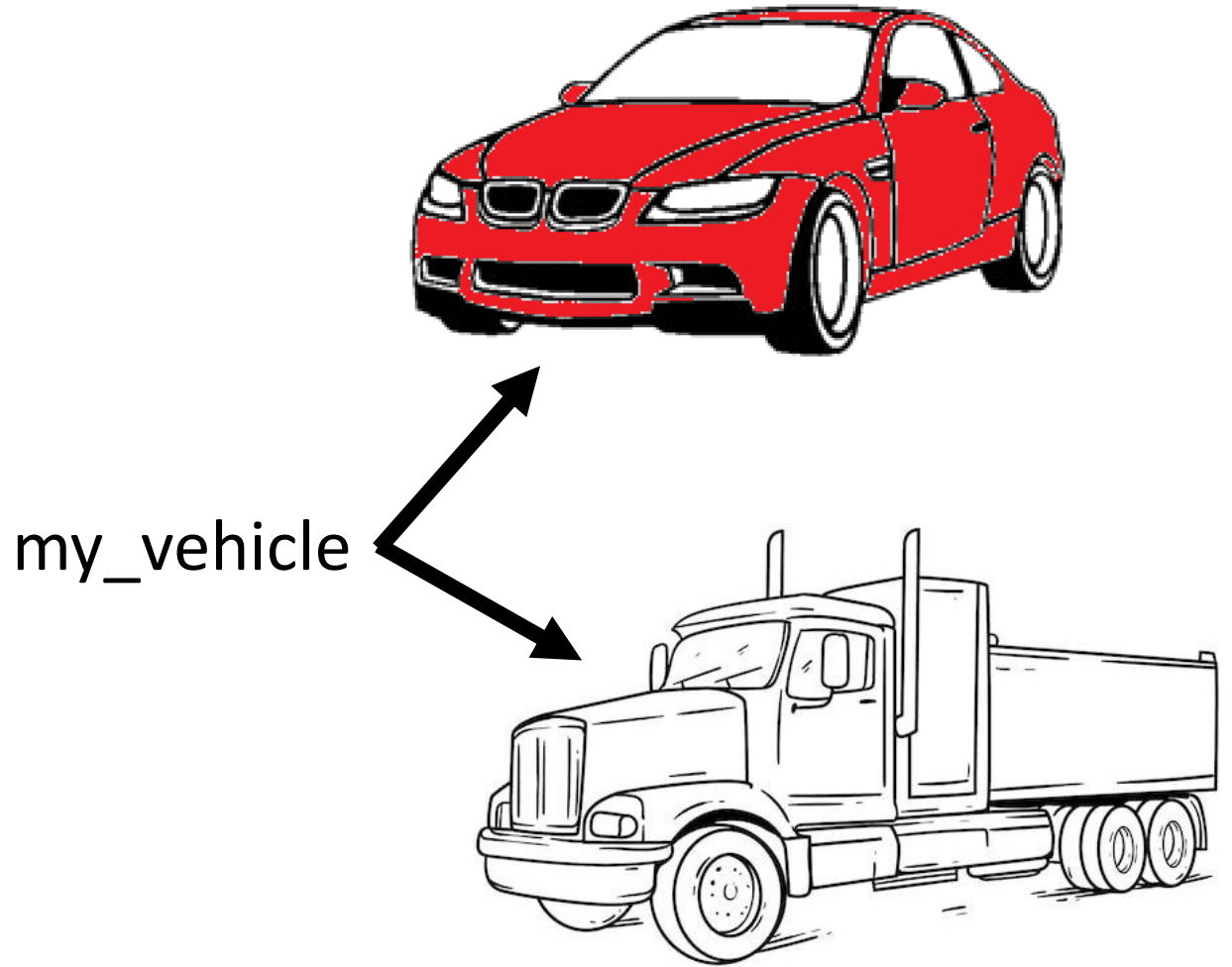
- ✓ Repaso: Variables numéricas + bool
- ✓ Condicionales
- ✓ Definir secuencias: list, tuple, str



Repaso: Variables

Etiquetas para nuestros objetos

- Reutilizar e indicar utilidad de nuestros objetos
- Naturaleza cambiante
 - Objeto referenciado cambia su estado interno
 - Referenciar otro objeto



Repaso: Tipos numéricos

int

float

Formas de obtener:

1. A partir de un literal

1

100

2_000_500

1.0

100.5

0.1234

2. Como resultado de un cálculo

10 + 20

2 ** 3

-50

2.0 + 4.0

1.6 ** 2

2 / 3

Repaso: int y float

Como todos, los tipos numéricos vistos también son objetos

- Tienen un estado interno (su valor)

10

- Tiene funcionalidades (ej: saben sumarse con otros)

(10).__add__(5) → 15

Repaso: Representar floats



$$0.101_2 = \boxed{1} \cdot 2^{-1} + \boxed{0} \cdot 2^{-2} + \boxed{1} \cdot 2^{-3} = \frac{1}{2} + \frac{0}{4} + \frac{1}{8} = 0.625_{10}$$

$$0.1 = \frac{0}{2} + \frac{0}{4} + \frac{0}{8} + \frac{1}{16} + \frac{1}{32} + \frac{0}{64} + \frac{0}{128} + \frac{1}{256} + \frac{1}{512} + \dots$$

Diagram illustrating the binary expansion of 0.1 (decimal) as a sum of powers of 2. Red brackets group terms to show their decimal equivalents:

- Terms $\frac{1}{16} + \frac{1}{32}$ are grouped and labeled 0.09375 .
- Terms $\frac{1}{256} + \frac{1}{512}$ are grouped and labeled 0.001953125 .
- The sum of these two groups is 0.095703125 .

Repaso: Operadores aritméticos

`a + b` # Addition

`a - b` # Subtraction

`a * b` # Multiplication

`a / b` # Division

`a ** b` # Exponentiation

`a // b` # Floor division

`a % b` # Modulo

Realmente lo

determina el objeto

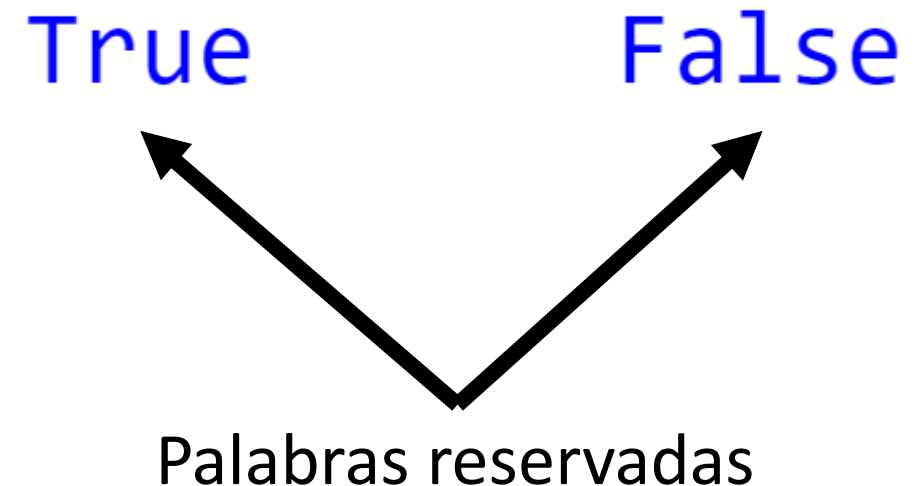
—————→ `a.__mul__(b)`

—————→ NO TRUNCA!!!

Adicional: bool

- Todos los objetos tienen valor **bool**
- Tienen lo que se llama “truthiness”
- Al operar **True** y **False** se comportan igual que **int** 1 y 0

Podemos usar un objeto como condición!



Adicional: palabras reservadas



Existen palabras reservadas que no debemos (y en muchos casos no podremos) utilizar:

- Operadores lógicos: **and, or, not, is, in, ...**
- Constantes: **None, True, False, ...**
- Tipos: **int, float, list, ...**
- Otros...



Condicionales

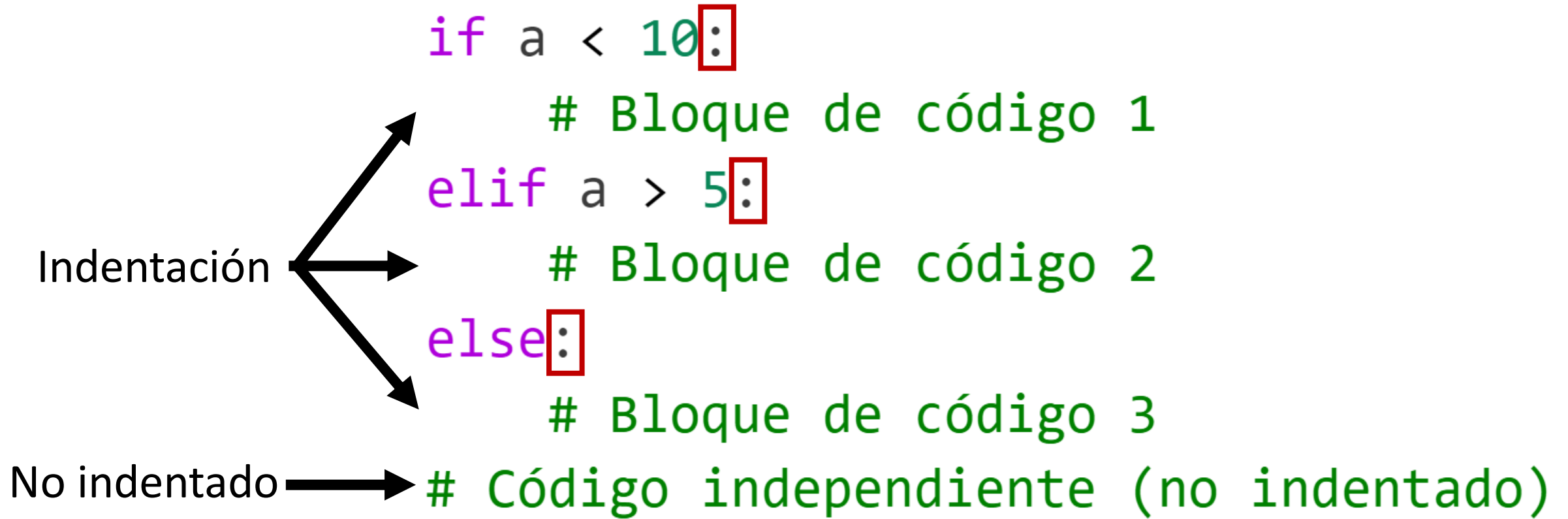
Dada una condición y un bloque de código, para definir un condicional utilizamos:

- Palabras reservadas: **if**, **elif**, **else**
- “:” delimitador condición-bloque de código
- Indentación   IDEs modernas lo gestionan


(Para definir una condición operadores condicionales)



Condicionales



Condicionales: Precedencia operar

Operator	Symbol	Description	Highest
Parentheses	()	Parentheses	
Exponent	**	Exponentiation	
	~X	Bitwise Nor	
Arithmetic	*, /, //, %	Multiplication, Divide, Floor Division, Mod	
	+, -	Addition, Subtraction	
Bitwise	&	Bitwise And	
	^	Bitwise Not	
		Bitwise Or	
Relational, Identity	<, <=, >, >=, !=, ==, is, is not	Comparison, Identity Operators	
Logical Operator	not	Not (Logical Operator)	
	and	And (Logical Operator)	
	or	Or (Logical Operator)	Lowest

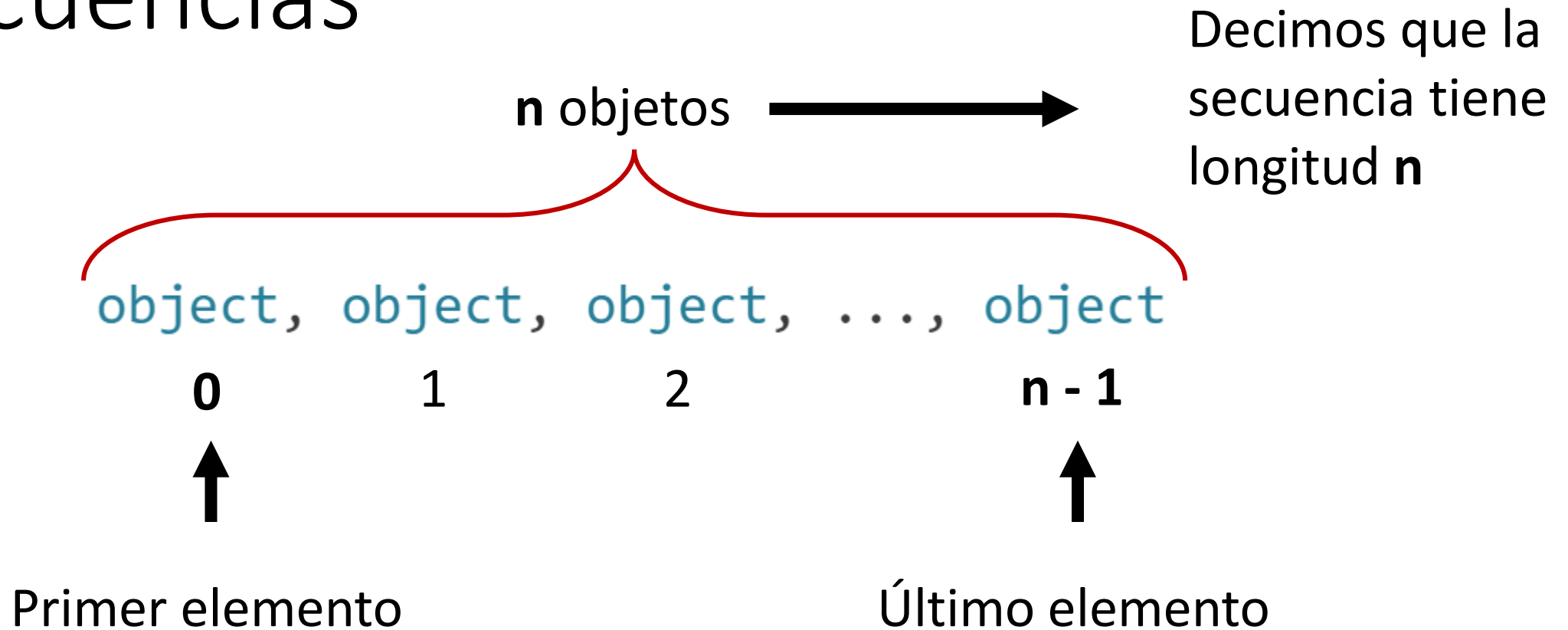
Secuencias

Colección de objetos ordenados

- Existe un orden secuencial
- Podemos acceder a los objetos por índice
- Como todo en Python, son objetos, que en particular pueden contener otros objetos



Secuencias

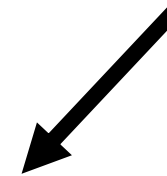


Secuencias

Las podemos clasificar en dos tipos:

- Homogéneas → 1, 2, 3, 4
- Heterogéneas → 34.0, "New York", True

Secuencia string



Dependiendo del caso utilizaremos un tipo de secuencia distinto

Secuencias

`list` → Mutable heterogeneous

`tuple` → Immutable heterogeneous

`str` → Immutable homogeneous

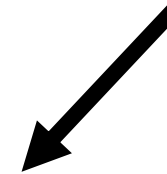
↑
Solo **char**

Secuencias

Las podemos clasificar en dos tipos:

- Homogéneas → 1, 2, 3, 4
- Heterogéneas → 34.0, "New York", True

Secuencia string



Dependiendo del caso utilizaremos un tipo de secuencia distinto

Secuencias heterogéneas: list y tuple

Se recomienda usar:

`list` \longrightarrow `1, 2, 3, 4`

`tuple` \longrightarrow `34.0, "New York", True`

(**list** menos seguro por ser mutable)

Secuencias: Definir list

Para definir una lista 2 opciones:

```
my_list = [10, 20, 30, 40, 50]
```

```
my_list = list(iterable)
```

Es posible definir una lista vacía:


```
my_empty_list = []      my_empty_list = list()
```

Secuencias: Definir tuple

Lo que realmente define a la tupla son las comas!!!

Podemos utilizar:

```
my_tuple = 34.0, "New York", True
my_tuple = (34.0, "New York", True)
my_tuple = tuple(iterable)
```



Es posible definir una tupla vacía:

```
my_empty_tuple = ()    my_empty_tuple = tuple()
```

Secuencias: Definir str

Podemos utilizar:

```
my_str = "Hello world!"
```

```
my_str = 'Hello world!'
```

Es posible definir una string vacía:

```
my_empty_str = ""
```

Secuencias: Definir str

String en varias líneas (sin \n)

```
"""
```

```
This is a string  
with multiple lines
```


```
"""
```

Para documentar!



Secuencias: Raw strings

“\” son caracteres especiales



```
my_lines = r" Line1\n Line2 \nLine3"  
print(my_lines)  # Line1\n Line2 \nLine3
```

(Se imprime la string ignorando escape chars)

Secuencias: Formatted strings

```
some_value = 123.456
```

Nos permiten insertar representación de valores en un string

```
my_fstring = f"Result: {some_value}" 123.46
```

Podemos formatear tipos numéricos (entre otros si lo admiten)

```
my_fstring = f"Result: {some_value:.2f}"
```



Secuencias: Indexing

Para acceder a los elementos de una secuencia (están ordenados)

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

Utilizamos objetos para indexar (por ejemplo **int**)

my_sequence[0] # 1

my_sequence [len(my_sequence) - 1] # 5

Devuelve el número de elementos (5)

