

# Curso Python 2025 Día 4: Slicing y mutabilidad



# Hoy:

- ✓ Repaso: Condicionales y secuencias
- ✓ Slicing
- ✓ Memoria y operador identidad
- ✓ Mutabilidad



# Repaso: Bool y condicionales

## Condicionales

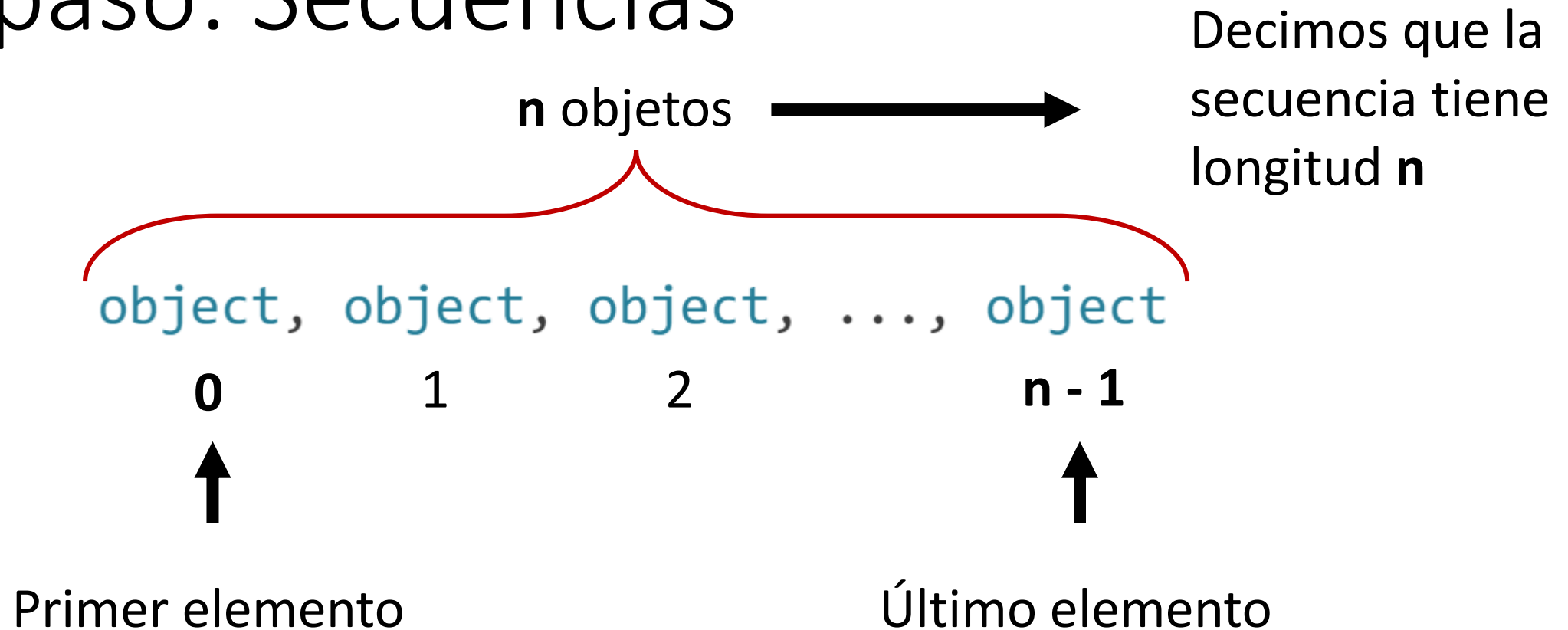
- Constantes bool **True** y **False**
- Definir mediante “:” + indentación
- Todos los objetos tienen valor **truthy**

```
a = 7
if a < 5:
    # Bloque de código 1
else:
    # Bloque de código 2
# Código independiente
```

**False**

Podemos usar un objeto como condición!

# Repaso: Secuencias



# Repaso: Secuencias

`list` → Mutable heterogeneous

`tuple` → Immutable heterogeneous

`str` → Immutable homogeneous

En listas aunque sea posible tipos distintos...  
Se recomienda todo del mismo tipo!

↑  
Solo **char**

# Repaso: 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()
```

# Repaso: 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()
```

# Repaso: Definir str

Podemos utilizar:

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

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

Es posible definir una string vacía:

```
my_empty_str = ""
```



# Repaso: 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

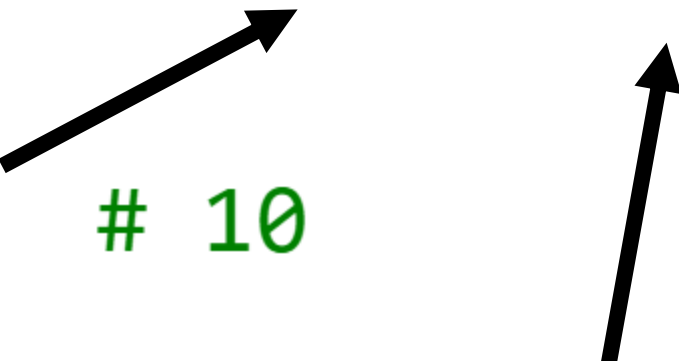
Devuelve el número de elementos (5)



# Slicing

Extraer un “rango” de elementos de una secuencia

```
my_seq = [10, 20, 30, 40]
```



```
my_seq[0]    # 10
```

```
my_seq[1:3]
```

# Slicing

Se crea un nuevo objeto del mismo tipo, para ello necesitamos

- Un inicio del rango (incluido)

- Un final del rango (no incluido)

$$[\text{start}, \text{stop}) \in \mathbb{Z}$$

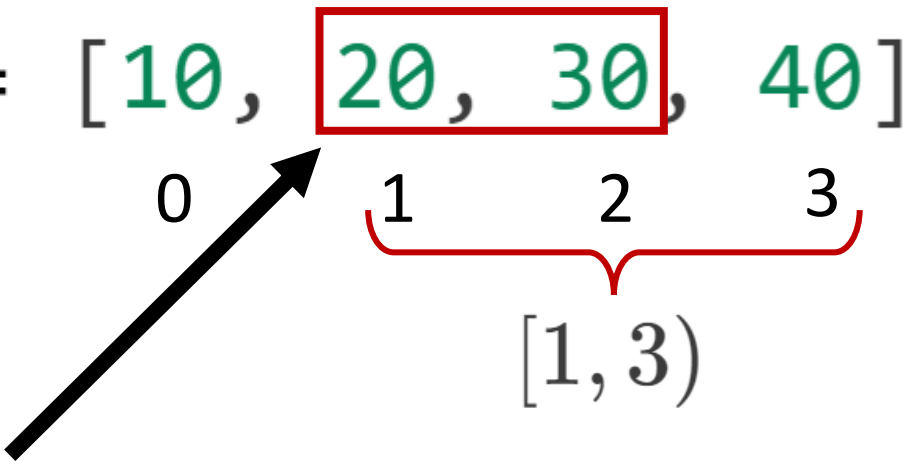
- Opcionalmente un paso (llamado step)

Que utilizaremos como si indexásemos  $\rightarrow$  `my_seq[start:stop:step]`

# Slicing

Ejemplo:

`my_seq = [10, 20, 30, 40]`



0 1 2 3

[1, 3)

(3 no incluido)

`my_seq[1:3] # [20, 30]`

No especificamos step, por defecto es 1

# Slicing

Elementos en posiciones pares

Ejemplo con step:

`my_seq = [10, 20, 30, 40]`

0      1      2      3

`my_seq[start:stop:step]`

`my_seq[0:3:2]`

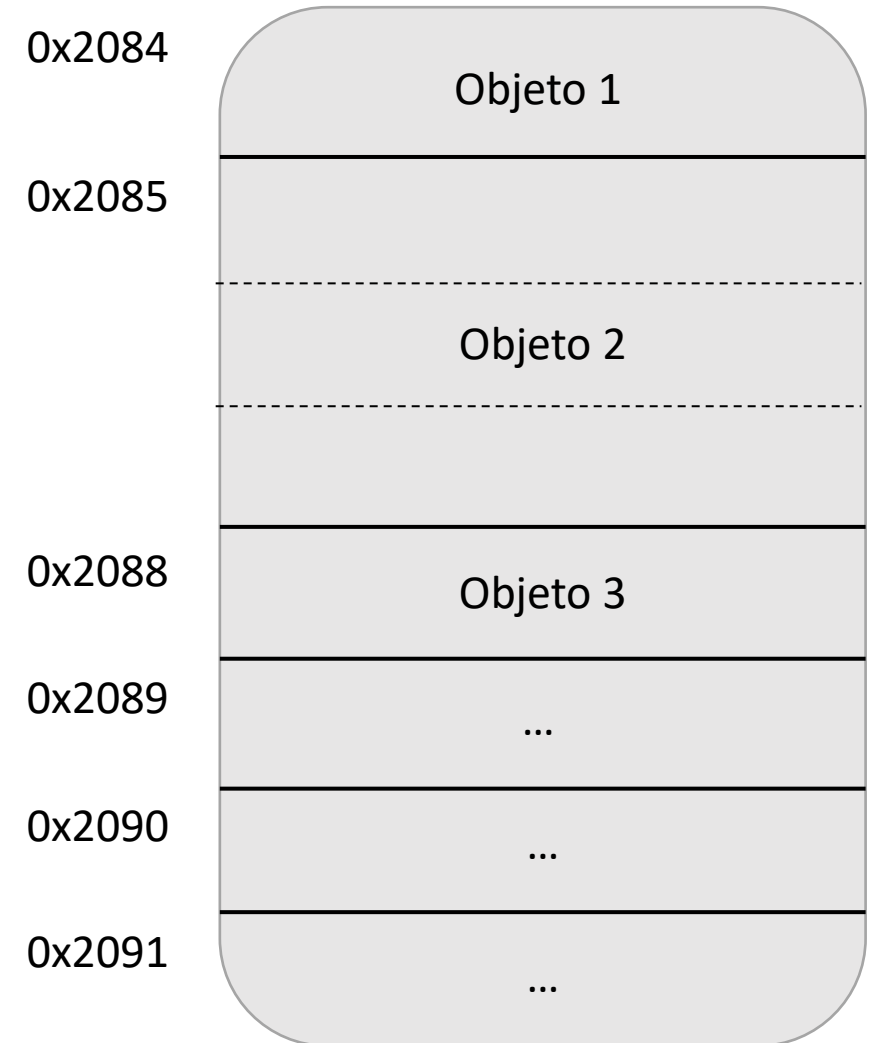
Si no sabemos longitud de la lista:

`my_seq[0:len(my_seq):2]`

Recordamos stop no incluido

# Memoria

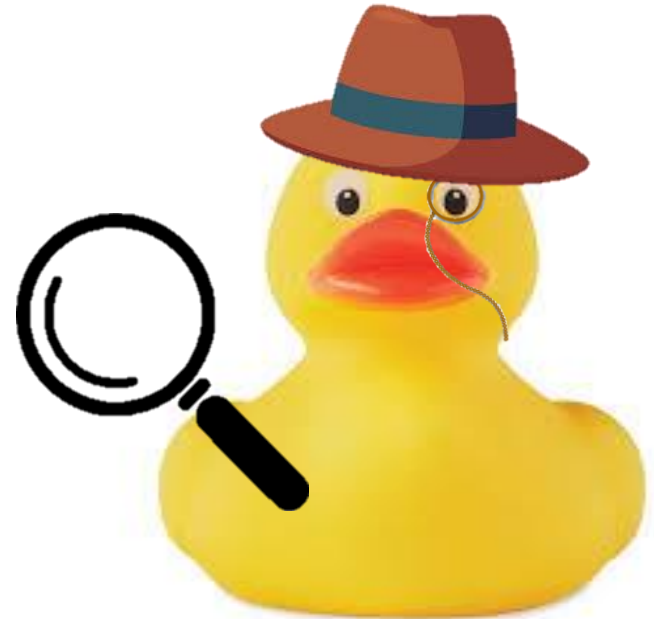
- La memoria es un conjunto de cajones donde podemos pedir y guardar datos
- Habrá datos que ocupen más de un hueco
- Con saber donde empieza cada objeto nos basta



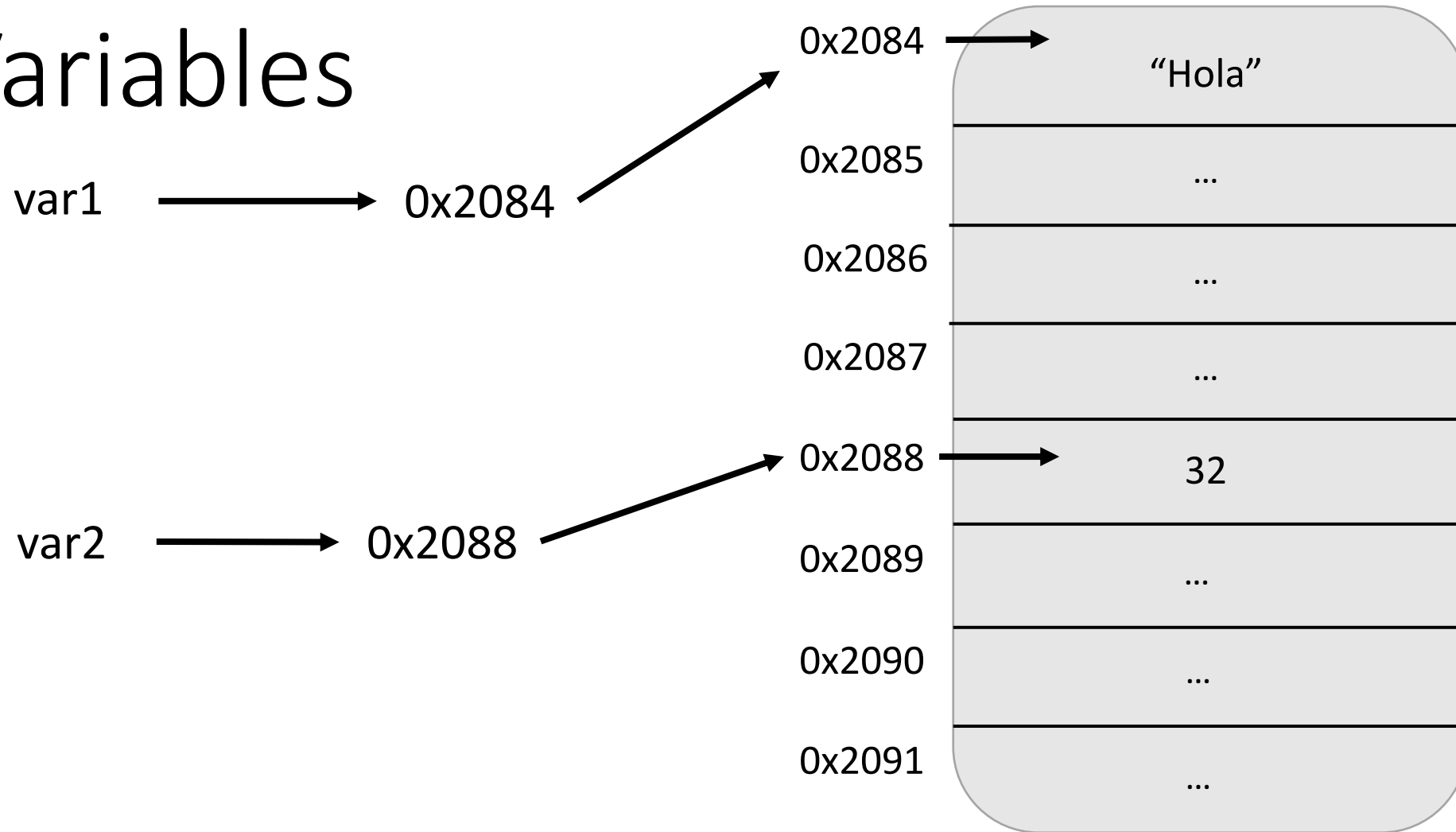
# Variables y referencias

Las variables son etiquetas que  
referencian a objetos...

... en la práctica es cierto pero hay más  
detrás de estas



# Variables





# Variables

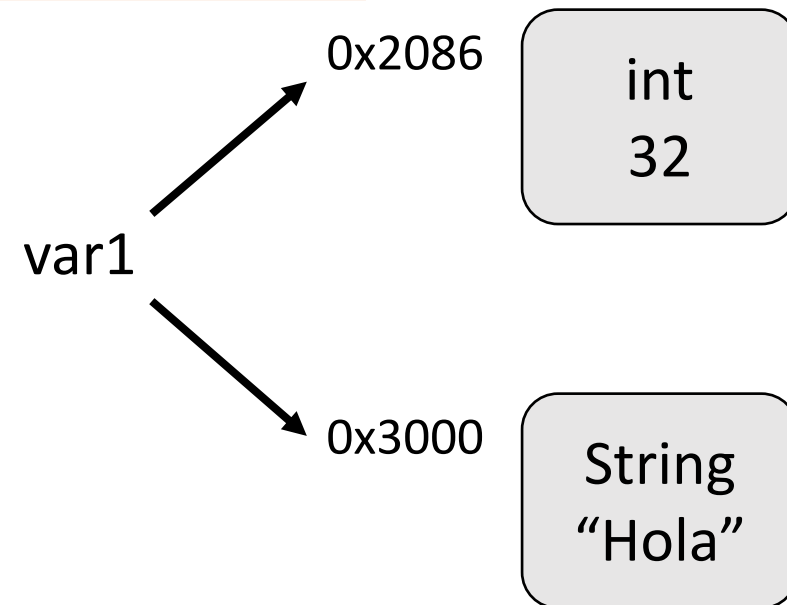
- Son “nombres” de las direcciones de memoria
- Una variable no es una etiqueta de objeto realmente
- Una variable es una referencia a una dirección que contiene un objeto



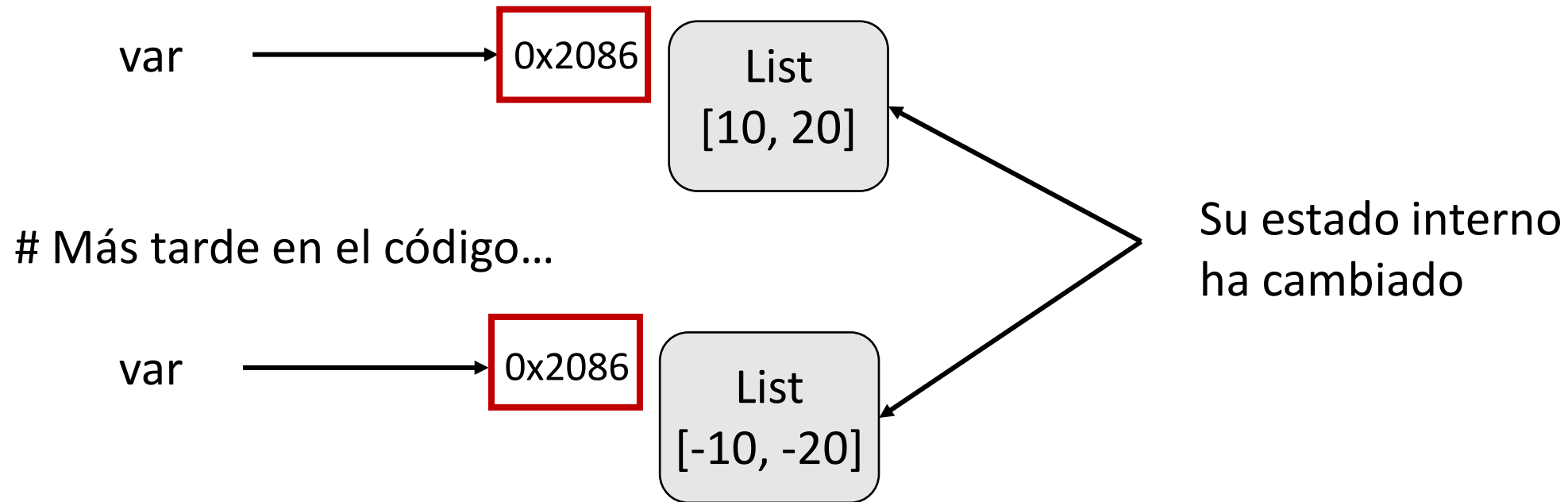
# Variables: Python

- Las variables no tienen un tipo asignado
- Python es un lenguaje dinámico (no estático)
- Si cambiamos el objeto referenciado cambiamos su referencia a otro sitio en memoria

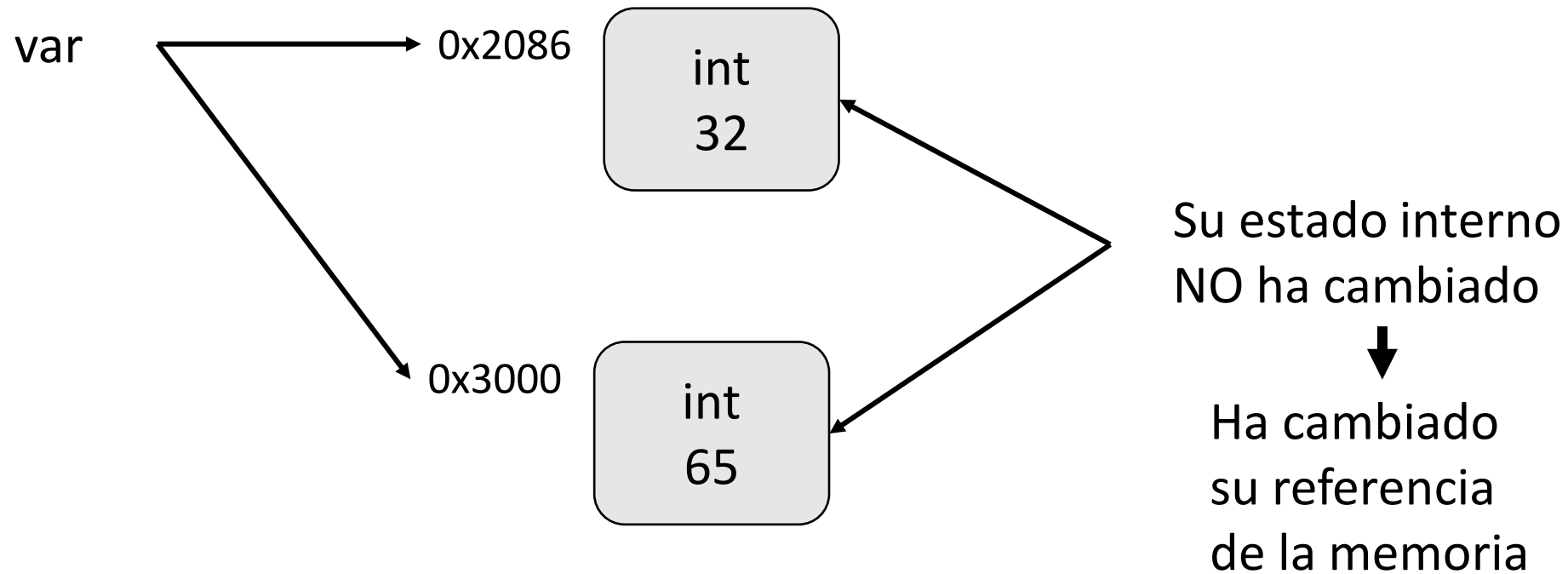
```
var1 = 32  
...  
var1 = "Hola"
```



# Mutabilidad: Objetos mutables



# Mutabilidad: Objetos inmutables



Cuando realizamos una operación entre int obtenemos un nuevo objeto **int** !!!

# Mutabilidad: Ejemplos Python

## INMUTABLES

- Tipos numéricos
- Strings
- Tuplas
- Clases definidas por usuarios

## MUTABLES

- Listas
- Sets
- Diccionarios
- Clases definidas por usuarios

# Mutabilidad:Tuplas vs Listas

- En listas podemos remplazar, añadir y eliminar, en tuplas NO
- Es más seguro trabajar con tipos inmutables

```
lista1 = [1, 2]  
tupla1 = (3, 4)
```

```
lista1.append(3)  
lista1[0] = 100
```

```
# Con tupla dará error  
# tupla1[0] = 100
```

# Mutabilidad: ¡Cuidado!

- Es posible que un objeto inmutable contenga elementos mutables
- Ejemplo: Tupla que contiene listas
- Podemos modificar objetos mutables que se encuentren dentro

```
a = ([1, 2], [3, 4])
```

```
a = (a[0], a[1])
```



La tupla contiene referencias a objetos mutables

# Mutabilidad: Shared references

- Cuando asignamos a una variable otra se comparte la referencia !!!
- En caso de inmutables además se comparte la referencia al realizar asignación con el objeto mismo !!!

