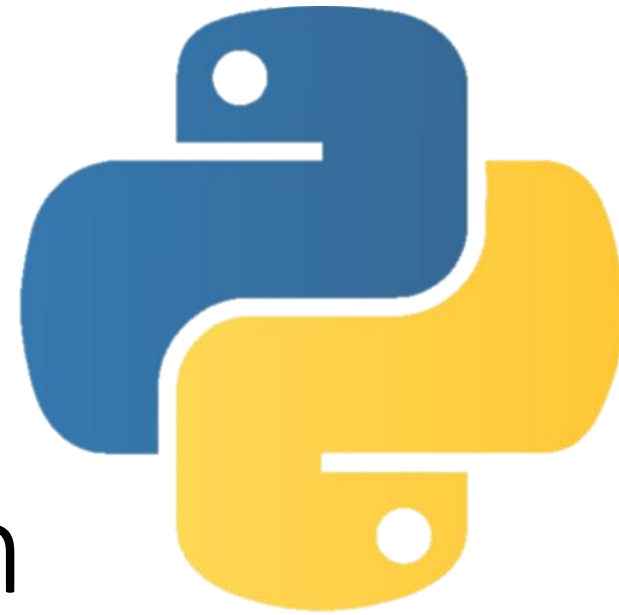


Curso Python
2025

Día 5:
Excepciones e iteración



Hoy:

- ✓ Repaso: Slicing y mutabilidad
- ✓ Excepciones
- ✓ Iterables vs iteradores
- ✓ Loops



Repaso: Slicing

Extraer un “rango” de elementos de una secuencia.
Indexar pero en vez de **int** utilizamos:

- Notación → **start:stop:step**

`my_seq[start:stop:step]`

- Objeto → **slice(start, stop, step)**

`my_seq[slice(start, stop, step)]`

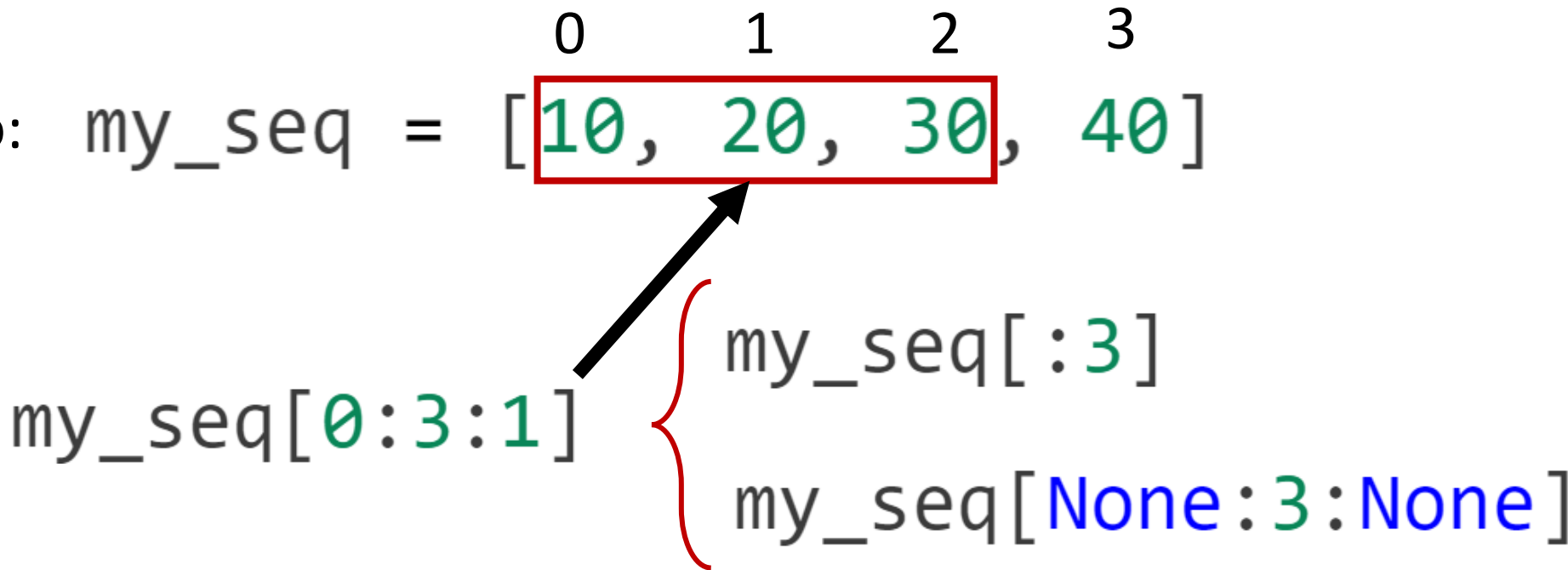
stop no incluido



Recordatorio: `my_sequence[x] == my_sequence.__getitem__(x)`

Repaso: Slicing

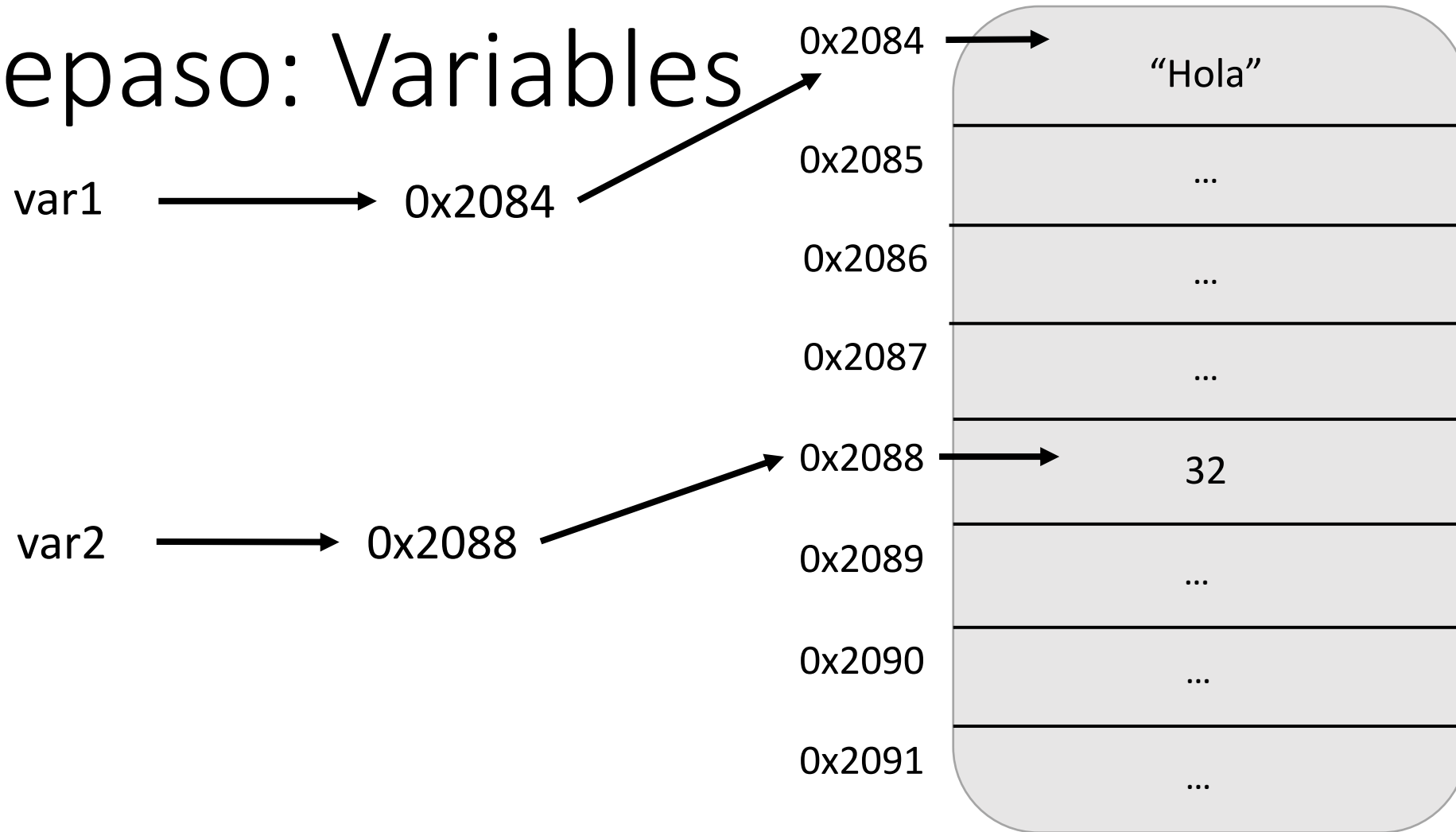
Ejemplo: `my_seq = [10, 20, 30, 40]`



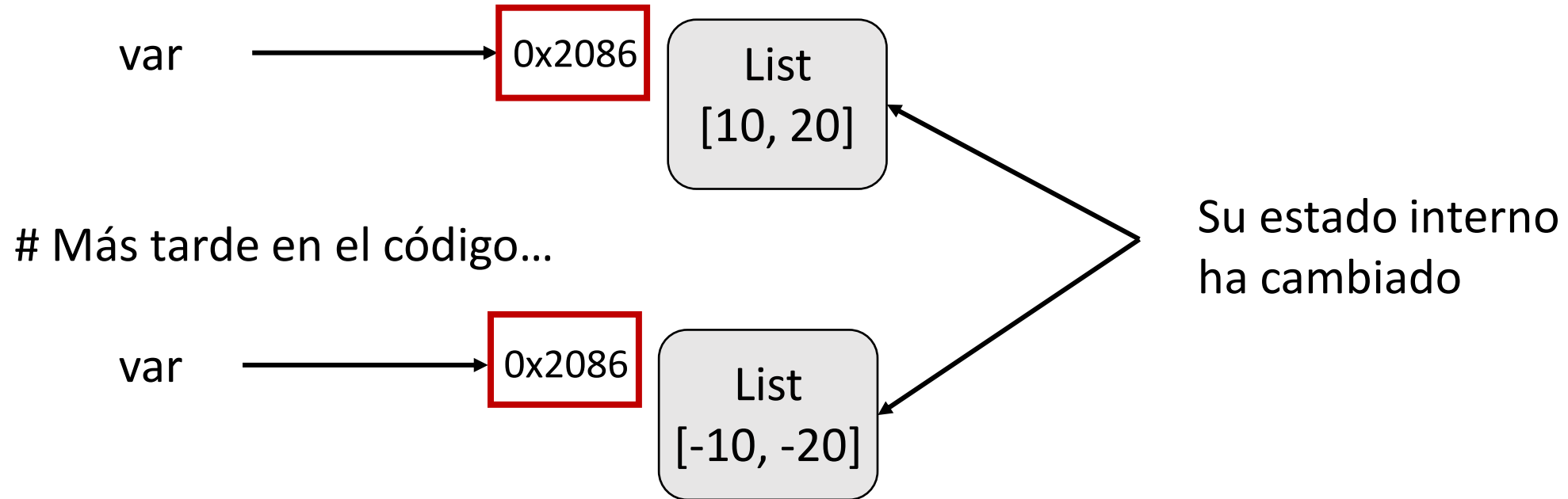
`my_seq[0:3:1]`
 `my_seq[:3]`
 `my_seq[None:3:None]`

Nota: Si no especificamos o damos valor **None** a start, stop o step por defecto toman valores 0, `len(my_seq)`, 1

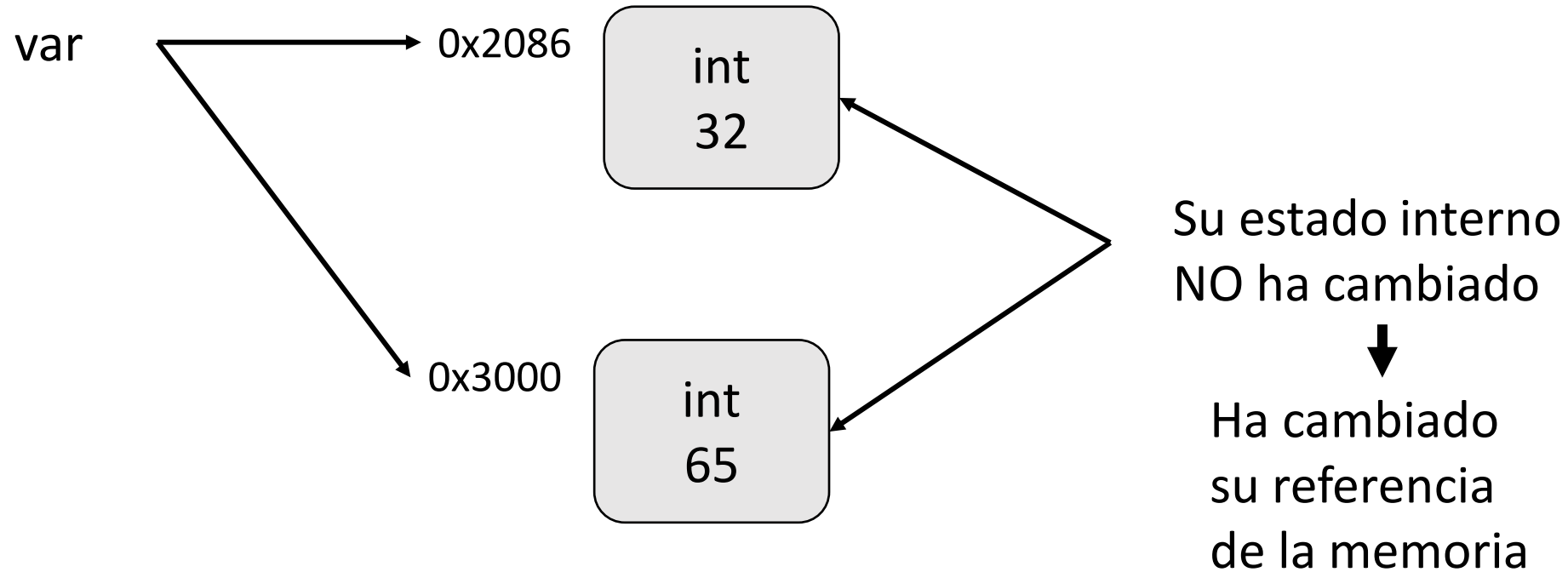
Repaso: Variables



Repaso: Objetos mutables



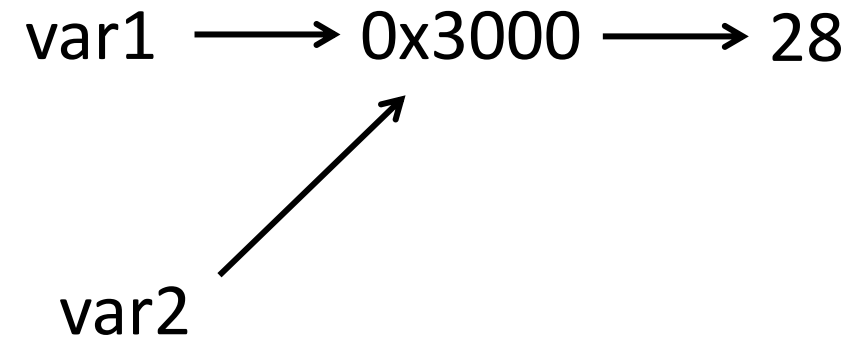
Repaso: Objetos inmutables



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

Repaso: Shared references

- Es posible que un objeto inmutable contenga objetos mutables
- Cuando asignamos a una variable otra se comparte la referencia !!!
- Si queremos una copia de un objeto mutable usamos **copy** y **deepcopy**



Excepciones: Concepto

Una excepción es un evento que **normalmente** ocurre cuando Python se encuentra con un comportamiento inesperado

- Hace que el programa termine de manera abrupta si no se gestiona
- Si no la gestionamos el programa termina

IndexError



[1, 2][2]

1 / 0

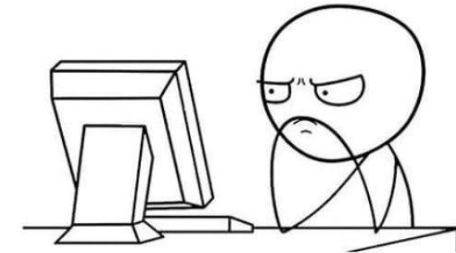


ZeroDivisionError

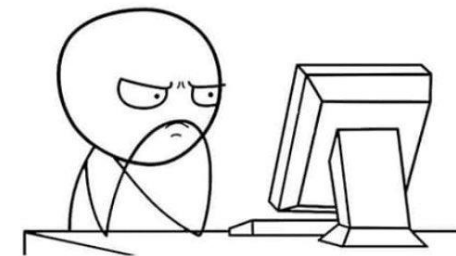
Excepciones: Terminología

- **Exception** es un tipo de objeto en Python
- Se puede “arrojar” **raise Exception**
- Se puede gestionar, lo que llamamos **exception handling**

THE CODE DOESN'T WORK...
WHY??



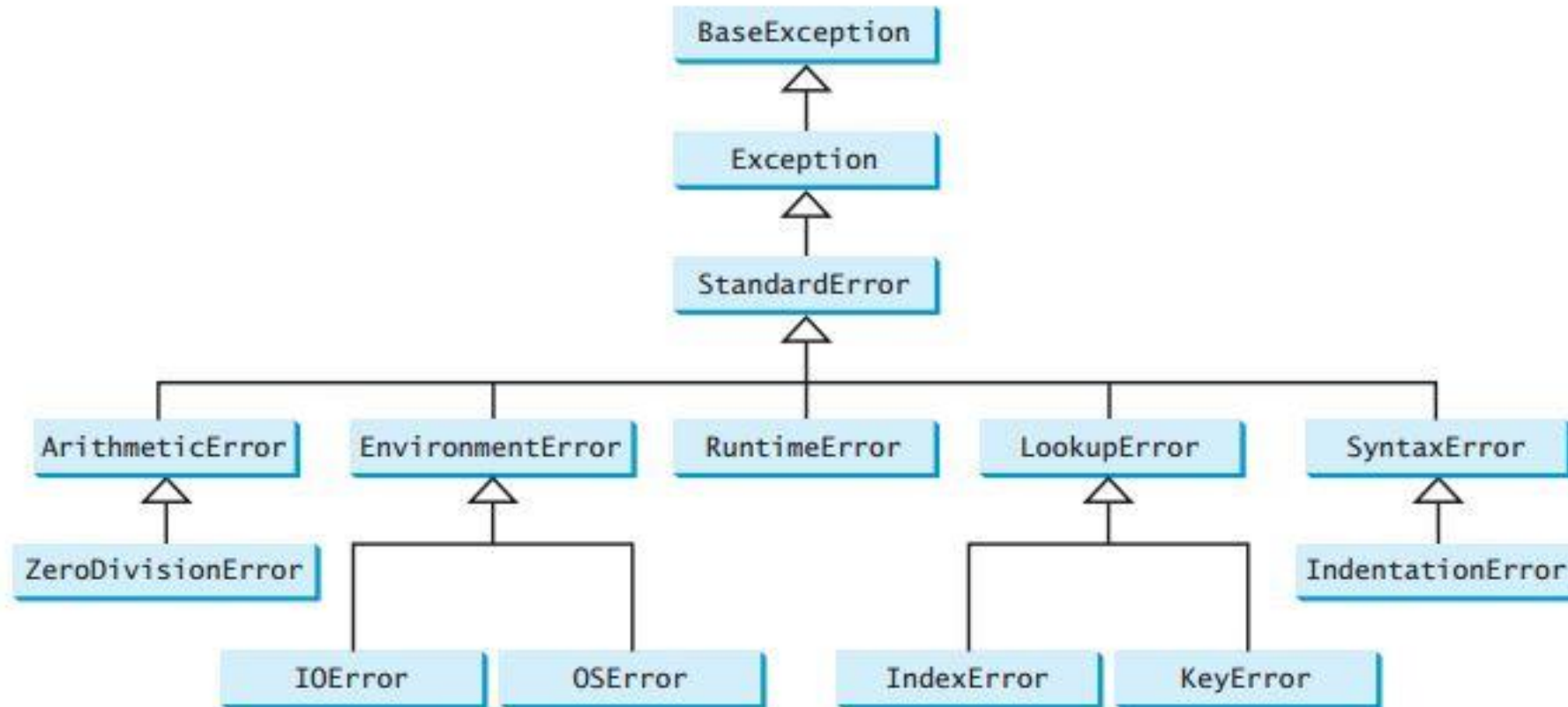
THE CODE IS WORKING...
WHY??



Excepciones: Estructura

Chunk of code	{	<code>try:</code>
		<code>res = a / b</code>
Exception handling	{	<code>except ZeroDivisionError:</code>
		<code>print("Can't div by zero")</code> <code>res = None</code>
Reraise if desired	<code>→</code>	<code>raise ZeroDivisionError</code>

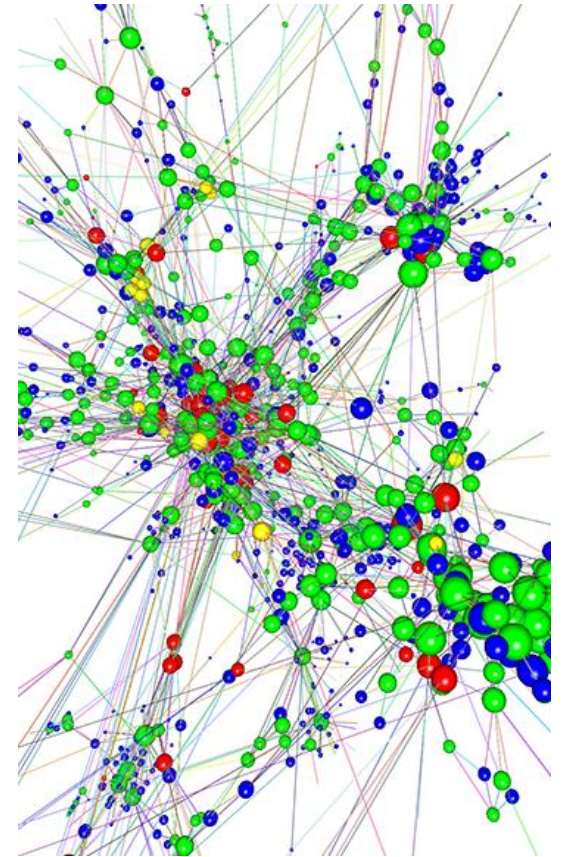
Excepciones: Jerarquía



Excepciones: EAFP vs LBYL

¿Qué podemos hacer ante comportamiento inesperado?

- **Look Before You Leap**: Planear que puede salir mal y poner los medios para que no ocurra
- **Easier to Ask Forgiveness than Permission**: En caso de que algo salga mal lo gestionaremos



En Python por lo general **EAFP** (una excepción es un evento raro)

Excepciones: Ejemplos

```
ex = ZeroDivisionError("Value division by zero")
```

Clase de excepción

Mensaje

`ZeroDivisionError` `IndexError` `StopIteration`



Iterables

- Un iterable es un objeto que podemos recorrer
- Pidiendo objeto por objeto
- Listas, tuplas, strings, ...
- Pedimos objetos hasta que se acaben (**exhaust**)



Iterables

Al iterar intervienen dos cosas:

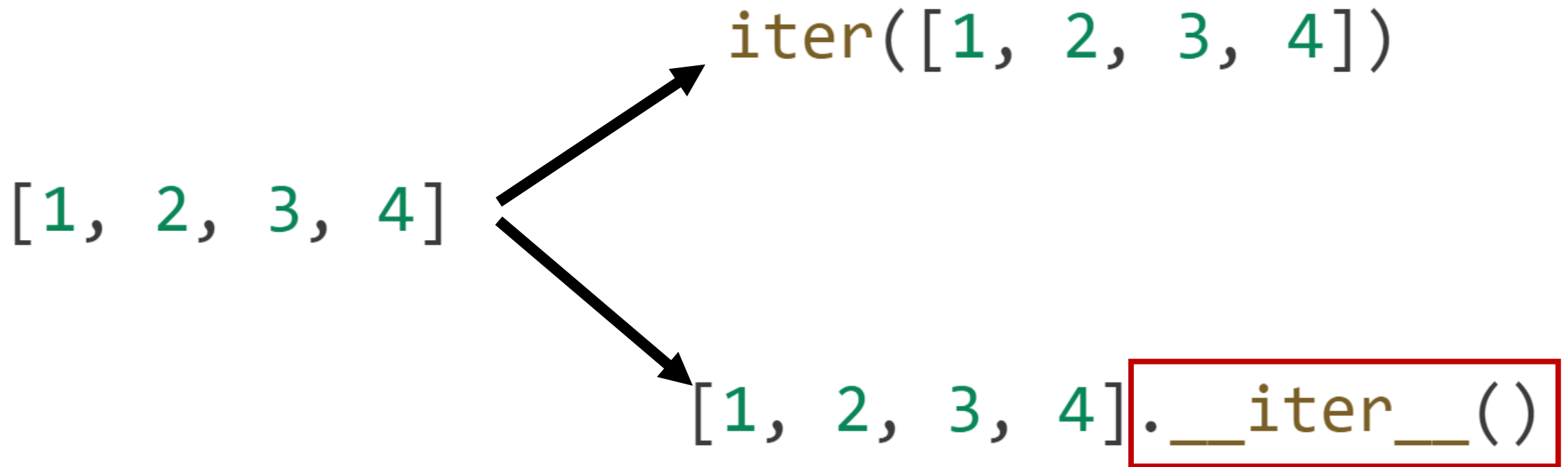
- Una colección de objetos (por ejemplo: secuencias)
- “Algo” capaz de devolver elementos y que distinga entre objetos que ya ha devuelto (recorrer)



Iterator

Iterables vs iteradores

Los iterables tienen una funcionalidad que les permite crear un iterador



Iterables vs iteradores

Un iterador sabe como devolver elementos

```
my_iter = iter([1, 2, 3])
```

```
next(my_iter) → 1
```

```
next(my_iter) → 2
```

```
next(my_iter) → 3
```

```
.__next__()
```



Agotar un iterador

Cuando no quedan más se arroja una excepción

```
my_iter = iter([1, 2, 3])
```

```
next(my_iter)
```

```
next(my_iter)
```

```
next(my_iter)
```

```
next(my_iter)
```

StopIteration



While loop

Esquema:

Condición

```
while a < 3:  
    print(a)  
    a += 1
```

Palabras clave:

break continue

For loop

Esquema:

Iterable
`for i in [1, 2, 3]:`
`print(i)`

Realmente implementa lo anterior...

Iterable → Iterador → StopIteration → Exception handling

For loop

```
try:
    my_iter = iter([1, 2, 3])
    while True:
        print(next(my_iter))
except StopIteration:
    pass
```

Diagram illustrating the relationship between `iter()` and `next()`:

- `iter()` is labeled as **Iterador**.
- `[1, 2, 3]` is labeled as **Iterable**.
- `next(my_iter)` is highlighted with a red box.

→ Aunque vacío hay
manejo de excepción

Teoría

