CONQUER BLOCKS

PYTHON

TUPLAS

REPASO

Estructuras de datos:

1. Listas → in - built
2. Arrays → importar modulo

QUE VEREMOS HOY

Estructuras de datos in - built:

1. Listas
2. Tuplas
3. Sets
4. Diccionarios

TUPLAS

Definición: Las tuplas son listas *inmutables*

➡ **No permiten añadir, eliminar o mover elementos (append, remove…)**

➡ **Permiten extraer porciones pero eso da como resultado una tupla nueva.**

➡ **Permiten comprobar si un elemento se encuentra en la tupla.**

TUPLAS

- ✓ **Más rápidas que las listas**
- ✓ **Ocupan menos espacio (mayor optimización)**
- ✓ **Pueden usarse como llaves de un diccionario.**

Si necesitamos guardar varios elementos pero en el futuro solo queremos recorrerlos para verlos, entonces nos conviene usar tuplas

SINTAXIS BASICA DE UNA TUPLA

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
```

✓ 0.0s

```
<class 'tuple'>
```

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
```

✓ 0.0s

```
<class 'list'>
```

SINTAXIS BASICA DE UNA TUPLA

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
```

✓ 0.0s

```
<class 'tuple'>
```

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
```

✓ 0.0s

```
<class 'list'>
```

```
# sintaxis de una tupla
mi_tupla_2 = "fruta", 45, True
print(type(mi_tupla_2))
```

✓ 0.0s

```
<class 'tuple'>
```

SINTAXIS BASICA DE UNA TUPLA

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
```

✓ 0.0s

```
<class 'tuple'>
```

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
```

✓ 0.0s

```
<class 'list'>
```

```
# sintaxis de una tupla
mi_tupla_2 = "fruta", 45, True
print(type(mi_tupla_2))
```

✓ 0.0s

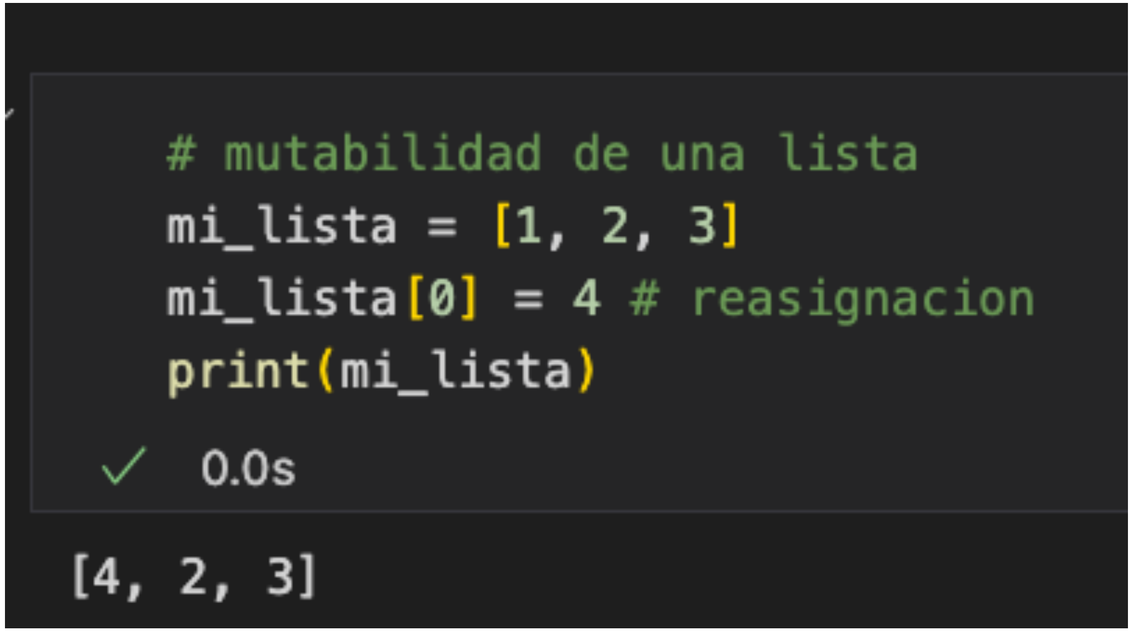
```
<class 'tuple'>
```

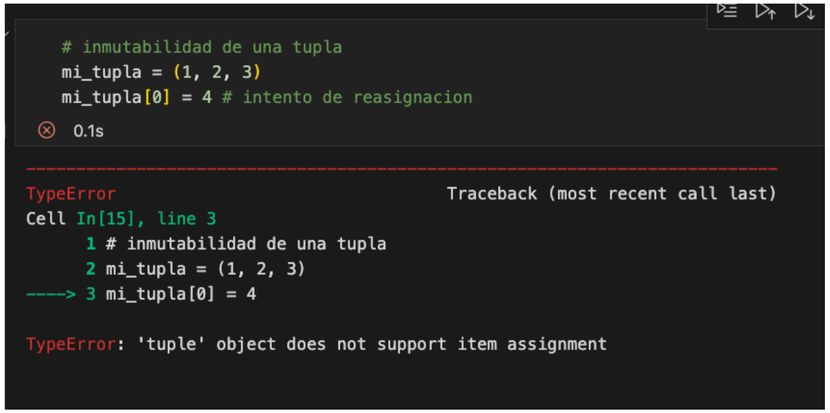
```
#acceder a elementos de un tupla
mi_tupla_1 = ("fruta", 45, True)
print(mi_tupla_1[1])
```

✓ 0.0s

```
45
```

INMUTABILIDAD





INMUTABILIDAD

```python
# mutabilidad de una lista
mi_lista = [1, 2, 3]
mi_lista[0] = 4 # reasignacion
print(mi_lista)
```

✓ 0.0s

```
[4, 2, 3]
```

```python
# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla[0] = 4 # intento de reasignacion
```

⊗ 0.1s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[15], line 3
      1 # inmutabilidad de una tupla
      2 mi_tupla = (1, 2, 3)
----> 3 mi_tupla[0] = 4

TypeError: 'tuple' object does not support item assignment
```

INMUTABILIDAD

```
# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla.append(4)
```

⊗ 0.0s

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent cal
Cell In[19], line 3
      1 # inmutabilidad de una tupla
      2 mi_tupla = (1, 2, 3)
----> 3 mi_tupla.append(4)

AttributeError: 'tuple' object has no attribute 'append'
```

INMUTABILIDAD

```python
# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla.append(4)
```

⊗ 0.0s

```
---------------------------------------------------------------------------
AttributeError                                Trace
Cell In[19], line 3
      1 # inmutabilidad de una tupla
      2 mi_tupla = (1, 2, 3)
----> 3 mi_tupla.append(4)

AttributeError: 'tuple' object has no attribute 'append'
```

```python
# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla.insert(0,4)
```

⊗ 0.0s

```
---------------------------------------------------------------------------
AttributeError                                Traceback (most recent cal
Cell In[20], line 3
      1 # inmutabilidad de una tupla
      2 mi_tupla = (1, 2, 3)
----> 3 mi_tupla.insert(0,4)

AttributeError: 'tuple' object has no attribute 'insert'
```

OPTIMIZACIÓN LISTA VS TUPLA

ESPACIO EN MEMORIA:

TIEMPO DE CREACIÓN:

OPTIMIZACIÓN LISTA VS TUPLA

ESPACIO EN MEMORIA:

```python
# espacio en memoria lista vs tupla
import sys
mi_lista = [0,1,2, "hola", True]
mi_tupla = (0,1,2,"hola", True)
print(sys.getsizeof(mi_lista),'bytes')
print(sys.getsizeof(mi_tupla),'bytes')
```

✓ 0.0s

```
120 bytes
80 bytes
```

TIEMPO DE CREACIÓN:

OPTIMIZACIÓN LISTA VS TUPLA

ESPACIO EN MEMORIA:

```
# espacio en memoria lista vs tupla
import sys
mi_lista = [0,1,2, "hola", True]
mi_tupla = (0,1,2,"hola", True)
print(sys.getsizeof(mi_lista),'bytes')
print(sys.getsizeof(mi_tupla),'bytes')
```

✓ 0.0s

```
120 bytes
80 bytes
```

TIEMPO DE CREACIÓN:

```
# tiempo de creacion list vs tupla
import timeit
print(timeit.timeit(stmt="[0,1,2,3,4,5]", number = 10000000))
print(timeit.timeit(stmt="(0,1,2,3,4,5)", number = 10000000))
```

✓ 0.3s

```
0.24537658299993836
0.037102917000083835
```

OPTIMIZACIÓN LISTA VS TUPLA

ESPACIO EN MEMORIA:

```
# espacio en memoria lista vs tupla
import sys
mi_lista = [0,1,2, "hola", True]
mi_tupla = (0,1,2, "hola", True)
print(sys.getsizeof(mi_lista),'bytes')
print(sys.getsizeof(mi_tupla),'bytes')
```

✓ 0.0s

```
120 bytes
80 bytes
```

x1.5

TIEMPO DE CREACIÓN:

```
# tiempo de creacion list vs tupla
import timeit
print(timeit.timeit(stmt="[0,1,2,3,4,5]", number = 10000000))
print(timeit.timeit(stmt="(0,1,2,3,4,5)", number = 10000000))
```

✓ 0.3s

```
0.24537658299993836
0.037102917000083835
```

x8

LISTA VS ARRAY VS TUPLA

	LISTAS	ARRAYS	TUPLAS
Mutabilidad	Mutable	Mutable	Inmutable
Acceso a elementos	Por índice o slicing	Por índice o slicing	Por índice o slicing
Tamaño de la lista	Dinámico	Fijo	Fijo
Tipo de elementos	Puede contener diferentes tipos	Todos los elementos son del mismo tipo	Puede contener diferentes tipos
Eficiencia	No tan eficiente como los arrays o tuplas	Más eficiente que las listas	Más eficiente que las listas
Uso principal	Cuando se requiere modificar la lista con frecuencia	Cuando se necesita eficiencia en la manipulación de elementos del mismo tipo	Cuando se necesitan elementos inmutables y eficientes

PASAR DE LISTAS A TUPLAS Y VICEVERSA

```python
# pasar de lista a tupla
mi_lista = [0,1,2, "hola", True]
print("mi_lista es de tipo...",type(mi_lista))
mi_tupla = tuple(mi_lista)
print("mi_tupla es de tipo...",type(mi_tupla))
print(mi_tupla)
```

✓ 0.0s

```
mi_lista es de tipo... <class 'list'>
mi_tupla es de tipo... <class 'tuple'>
(0, 1, 2, 'hola', True)
```

```python
# pasar de tupla a lista
mi_tupla = (0,1,2, "hola", True)
print("mi_tupla es de tipo...",type(mi_tupla))
mi_lista = list(mi_tupla)
print("mi_lista es de tipo...",type(mi_lista))
print(mi_lista)
```

✓ 0.0s

```
mi_tupla es de tipo... <class 'tuple'>
mi_lista es de tipo... <class 'list'>
[0, 1, 2, 'hola', True]
```

PASAR DE LISTAS A TUPLAS Y VICEVERSA

```python
# pasar de lista a tupla
mi_lista = [0,1,2, "hola", True]
print("mi_lista es de tipo...",type(mi_lista))
mi_tupla = tuple(mi_lista)
print("mi_tupla es de tipo...",type(mi_tupla))
print(mi_tupla)
```

✓ 0.0s

```
mi_lista es de tipo... <class 'list'>
mi_tupla es de tipo... <class 'tuple'>
(0, 1, 2, 'hola', True)
```

```python
# pasar de tupla a lista
mi_tupla = (0,1,2, "hola", True)
print("mi_tupla es de tipo...",type(mi_tupla))
mi_lista = list(mi_tupla)
print("mi_lista es de tipo...",type(mi_lista))
print(mi_lista)
```

✓ 0.0s

```
mi_tupla es de tipo... <class 'tuple'>
mi_lista es de tipo... <class 'list'>
[0, 1, 2, 'hola', True]
```

TRABAJANDO CON TUPLAS

ACCEDER A ELEMENTOS:

```
#acceder a elementos de un tupla
mi_tupla_1 = ("fruta", 45, True)
print(mi_tupla_1[1])
```

✓ 0.0s

```
45
```

SLICING:

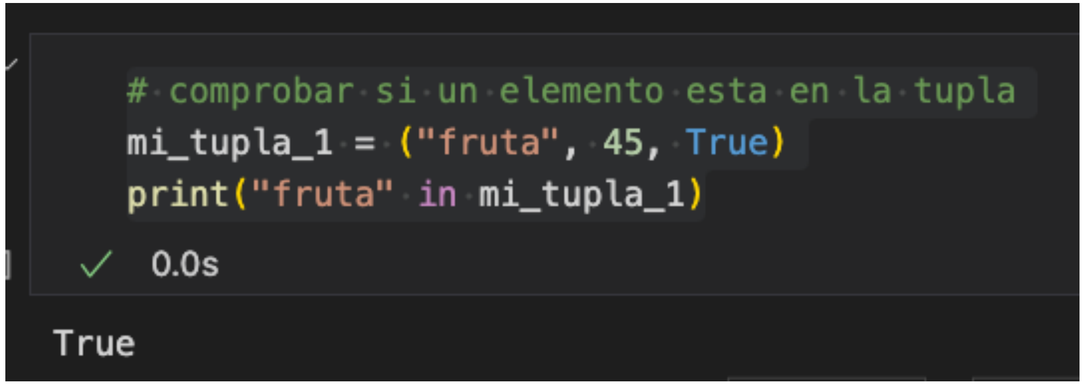
```
# slicing
mi_tupla = (1, 2, 3, 4, 5)
subtupla = mi_tupla[1:3]
print(subtupla) # (2, 3)
```

✓ 0.0s

```
(2, 3)
```

TRABAJANDO CON TUPLAS

COMPROBAR LA EXISTENCIA DE ELEMENTOS:



TRABAJANDO CON TUPLAS

NUMERO DE ELEMENTOS:

```
# longitud de la tupla
mi_tupla = ("fruta", 45, True)
longitud = len(mi_tupla)
print(longitud) # 3
```

✓ 0.0s

```
3
```

NUMERO DE APARICIONES:

```
# numero de apariciones
mi_tupla = ("fruta", 45, True)
num_apariciones = mi_tupla.count(45)
print(num_apariciones)
```

✓ 0.0s

```
1
```

TRABAJANDO CON TUPLAS

NUMERO DE ELEMENTOS:

```python
# longitud de la tupla
mi_tupla = ("fruta", 45, True)
longitud = len(mi_tupla)
print(longitud) # 3
```

✓ 0.0s

```
3
```

NUMERO DE APARICIONES:

```python
# numero de apariciones
mi_tupla = ("fruta", 45, True)
num_apariciones = mi_tupla.count(45)
print(num_apariciones)
```

✓ 0.0s

```
1
```

```python
# numero de apariciones
mi_tupla = ("fruta", 45, True)
num_apariciones = mi_tupla.count("perro")
print(num_apariciones)
```

✓ 0.0s

```
0
```

TRABAJANDO CON TUPLAS

NUMERO DE ELEMENTOS:

```
# longitud de la tupla
mi_tupla = ("fruta", 45, True)
longitud = len(mi_tupla)
print(longitud) # 3
```

✓ 0.0s

```
3
```

NUMERO DE APARICIONES:

```
# numero de apariciones
mi_tupla = ("fruta", 45, True)
num_apariciones = mi_tupla.count(45)
```

```
mi_tupla = (1, 2, 3, 3, 3, 4, 5)
num_apariciones = mi_tupla.count(3)
print(num_apariciones) # 3
```

✓ 0.0s

```
3
```

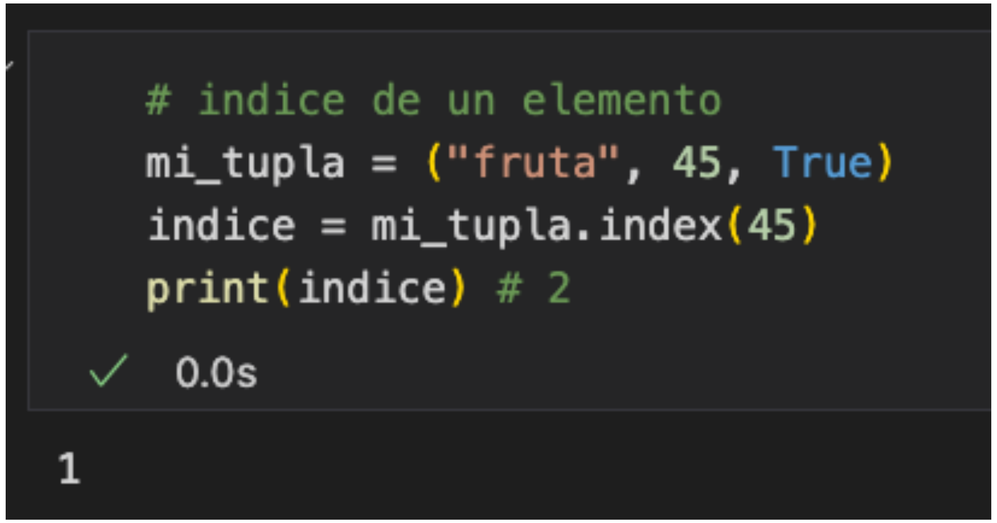
```
print(num_apariciones)
```

✓ 0.0s

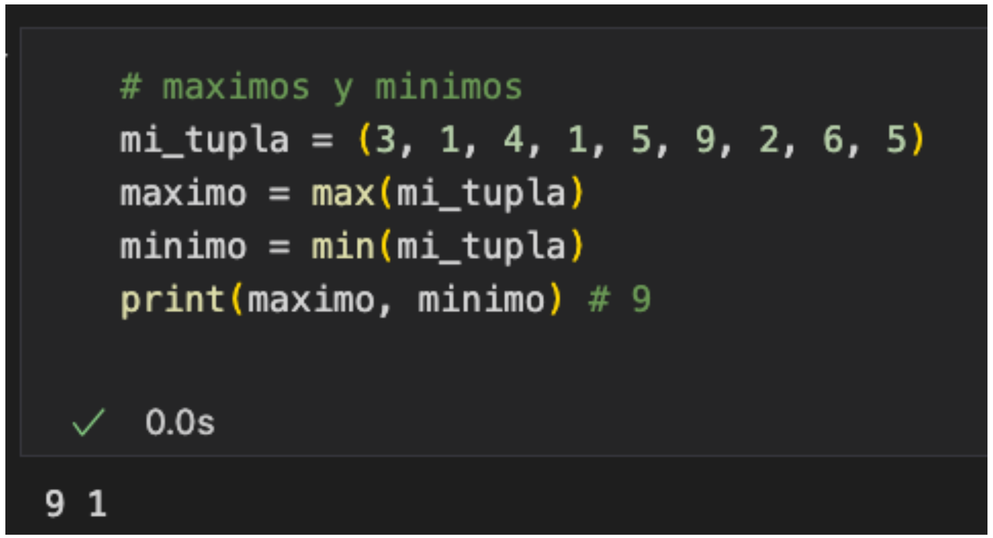
```
0
```

TRABAJANDO CON TUPLAS

ÍNDICE DE UN ELEMENTO:



MÁXIMOS Y MÍNIMOS:



TRABAJANDO CON TUPLAS

ORDENAR ELEMENTOS:

RETORNA UNA LISTA!

```
mi_tupla = (3, 1, 4, 1, 5, 9, 2, 6, 5)
mi_lista_ordenada = sorted(mi_tupla)
print(mi_lista_ordenada)
print(type(mi_lista_ordenada))
```

✓ 0.0s

```
[1, 1, 2, 3, 4, 5, 5, 6, 9]
<class 'list'>
```

ORDENAR ELEMENTOS DE MANERA INVERSA:

RETORNA UN OBJETO DE TIPO “REVERSED”…

```
mi_tupla = (1, 2, 3, 4, 5)
tupla_inversa = reversed(mi_tupla)
print(tupla_inversa)
print(type(tupla_inversa))
```

✓ 0.0s

```
<reversed object at 0x10a0537c0>
<class 'reversed'>
```

TRABAJANDO CON TUPLAS

ORDENAR ELEMENTOS:

RETORNA UNA LISTA!

```
mi_tupla = (3, 1, 4, 1, 5, 9, 2, 6, 5)
mi_tupla_ordenada = tuple(sorted(mi_tupla))
print(mi_tupla_ordenada)
print(type(mi_tupla_ordenada))
```

✓ 0.0s

```
(1, 1, 2, 3, 4, 5, 5, 6, 9)
<class 'tuple'>
```

ORDENAR ELEMENTOS DE MANERA INVERSA:

RETORNA UN OBJETO DE TIPO “REVERSED”…

```
mi_tupla = (1, 2, 3, 4, 5)
tupla_inversa = tuple(reversed(mi_tupla))
print(tupla_inversa)
print(type(tupla_inversa))
```

✓ 0.0s

```
(5, 4, 3, 2, 1)
<class 'tuple'>
```

TRABAJANDO CON TUPLAS

COMBINAR TUPLAS:

```
# combinar tuplas formando una tupla de tuplas
tupla1 = (1, 2, 3)
tupla2 = ('a', 'b', 'c')
tupla_combinada = tuple(zip(tupla1, tupla2))
print(tupla_combinada)
```

✓ 0.0s

```
((1, 'a'), (2, 'b'), (3, 'c'))
```

ACCEDER A LOS ELEMENTOS DE UNA TUPLA DE TUPLAS:

```
# acceder a los elementos de una tupla de tuplas
mi_tupla = ((1, 'a'), (2, 'b'), (3, 'c'))
print(mi_tupla[0][0])
print(mi_tupla[1][1])
print(mi_tupla[2][0])
```

✓ 0.0s

```
1
b
3
```

TRABAJANDO CON TUPLAS

SLICING DE UNA TUPLA DE TUPLAS:

```
mi_tupla = ((1, 2, 3), (4, 5, 6), (7, 8, 9))

# Slicing de una tupla interior
tupla_interior = mi_tupla[1]
print(tupla_interior) # (4, 5, 6)
```

✓ 0.0s

```
(4, 5, 6)
```

```
mi_tupla = ((1, 2, 3), (4, 5, 6), (7, 8, 9))

# Slicing de una porción de la tupla de tuplas
porcion_tupla = mi_tupla[0:2]
print(porcion_tupla) # ((1, 2, 3), (4, 5, 6))
```

✓ 0.0s

```
((1, 2, 3), (4, 5, 6))
```

```
mi_tupla = ((1, 2, 3), (4, 5, 6), (7, 8, 9))

# Slicing de una porción de una tupla interior
porcion_interior = mi_tupla[2][0:2]
print(porcion_interior) # (7, 8)
```

✓ 0.0s

```
(7, 8)
```

TUPLA UNITARIA

```
mi_tupla = (1)
print(type(mi_tupla))
```

✓ 0.0s

```
<class 'int'>
1
```

```
mi_tupla = (1,)
print(type(mi_tupla))
print(mi_tupla)
```

✓ 0.0s

```
<class 'tuple'>
(1,)
```

EMPAQUETAMIENTO Y DESEMPAQUETAMIENTO

```python
# empaquetamiento
mi_tupla = "fruta", 45, True
print(mi_tupla)
```

✓ 0.0s

```
('fruta', 45, True)
```

```python
# desempaquetamiento
mi_tupla = ("fruta", 45, True)
string, entero, booleano = mi_tupla
print(string)
print(entero)
print(booleano)
```

✓ 0.0s

```
fruta
45
True
```

DESEMPAQUETAMIENTO: POSIBLES ERRORES

```
# errores en el desempaquetamiento
mi_tupla = ("fruta", 45, True)
string, entero = mi_tupla
```

⊗ 0.0s

```
ValueError                                Traceback (most
Cell In[24], line 3
      1 # errores en el desempaquetamiento
      2 mi_tupla = ("fruta", 45, True)
----> 3 string, entero = mi_tupla

ValueError: too many values to unpack (expected 2)
```

```
# errores en el desempaquetamiento
mi_tupla = ("fruta", 45, True)
string, entero, booleano, otra_variable = mi_tupla
```

⊗ 0.0s

```
ValueError                                Traceback (most recent ca
Cell In[25], line 3
      1 # errores en el desempaquetamiento
      2 mi_tupla = ("fruta", 45, True)
----> 3 string, entero, booleano, otra_variable = mi_tupla

ValueError: not enough values to unpack (expected 4, got 3)
```

REPASO

1) Diferencias entre array, lista y tupla

2) Inmutabilidad de una tupla

3) Performance tupla vs lista

4) Bases del trabajo con tuplas (acceso a elementos, métodos y funciones, empaquetamiento y desempaquetamiento…)

5) Trabajo con tuplas de tuplas (acceso a elementos y slicing)

CONQUER BLOCKS