

Carolina Martínez García

Chechpoint 4, preguntas teóricas

¿Cuál es la diferencia entre una lista y una tupla en Python?

Las **listas** son como una matriz, son un tipo de datos que permite almacenar muchos valores en un solo contenedor. Los elementos pueden contener varios tipos de dato (números, cadenas, etc). Son **ordenadas y mutables**, sus elementos los puedes cambiar si es necesario.

Por esto es por lo que son una de las estructuras de datos más utilizadas, por su flexibilidad y porque tienen **muchas funciones** para trabajar con sus datos.

Se definen con **corchetes []**. Ejemplo de lista:

...

```
58 #Ejemplo de lista
59 rivers = ['Amazon', 'Nile', 'Yangtze']
60 #Ejemplo de lista con elementos de diferentes tipos
61 mixed_list = ['Amazon', 'Nile', 345, True]
```

...

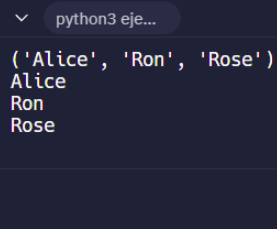
Las **tuplas** son estructuras de datos similares a una lista, pero son **inmutables**, después de crear sus elementos no los puedes cambiar. Al igual que las listas pueden contener varios tipos de datos y también son **ordenadas**. Una de las cosas para las que más se usa una tupla es para desempaquetar.

Las tuplas son útiles para cuando necesitas almacenar datos que no quieres cambiar, como por ejemplo registros de datos, claves de configuración...

Su sintaxis es con **paréntesis ()**. Ejemplo de tupla:

...

```
1 #Ejemplo de tupla
2 names = ('Alice', 'Ron', 'Rose',)
3 print(names)
4 #Ejemplo de desempaquetado de tupla
5 director, manager, tecnico = names
6 print(director)
7 print(manager)
8 print(tecnico)
```



...

La gran diferencia entre una lista y una tupla es que la tupla es inmutable, no se pueden hacer cambios, sin embargo en las listas puedes hacer tantos cambios como quieras.

A la hora de elegir entre crear una lista o una tupla, hay que pensar muy bien si estos datos los necesitarás o querrás cambiar.

¿Cuál es el orden de las operaciones?

El orden de las operaciones matemáticas en Python sigue un proceso que se llama **PEMDAS**:

1. **P**aréntesis ()
2. **E**xponente **
3. **M**ultiplicación *
4. **D**ivisión /
5. **A**ddition +
6. **S**ubstraction –

Ejemplo de orden de una operación:

...

```
ejemplos.py > ...  
1 #Ejemplo de orden de operacion  
2 calculation = 4 + 2 * 3  
3 print(calculation)
```

python3 eje...
10

...

Primero ejecuta la **multiplicación** ya que esta en tercer lugar y **después** realizará la suma, que **está** en un orden posterior a la **multiplicación**. Si necesitamos que haga la suma antes que la **multiplicación**, **podríamos** utilizar **paréntesis** para que le diera prioridad puesto que los **paréntesis** **están** los primeros en el proceso. Por ejemplo:

...

```
ejemplos.py > ...  
1 #Ejemplo de orden de operacion  
2 #calculation = 4 + 2 * 3  
3 #print(calculation)  
4 calculation = (4 + 2) * 3  
5 print(calculation)
```

python3 eje...
18

...

Así primero ha hecho la suma y **después** la **multiplicación**, obteniendo **así** por supuesto un resultado distinto.

¿Qué es un diccionario Python?

En Python un diccionario es un tipo de dato que almacena colecciones de datos en formato clave-valor. Se llama así porque es similar a un diccionario de los de buscar el significado de las palabras, tú tienes un libro donde encuentras palabras (clave) y su significado (valor).

Los diccionarios son **mutables**, se pueden agregar, eliminar o modificar elementos, al igual que se pueden anidar diccionarios dentro de otros.

Las **claves són únicas**, y para acceder a los valores hay que usar su clave, no se puede usar índices como en una lista.

Se definen con llaves ({ }), las claves se separan del valor con dos puntos (:), y van entre comillas (' ') y el par clave-valor se separa del siguiente par con una coma (,).

Los diccionarios son ideales para almacenar datos por ejemplo de usuarios.

Ejemplo:

...

```
ejemplos.py 7 ...
1  #Ejemplo de diccionario anidado al que llamamos a una de
2  sus claves
3  users = {
4      'user1': {
5          'name': 'Jules',
6          'age': '45',
7          'location': 'Spain',
8      },
9      'user2': {
10         'name': 'Susan',
11         'age': '77',
12         'location': 'England',
13     },
14     'user3': {
15         'name': 'Rose',
16         'age': '34',
17         'location': 'England',
18     },
19 }
20 print(users['user2']['age'])
```

...

¿Cuál es la diferencia entre el método ordenado y la función de ordenación?

El método ordenado **sort ()**, se usa para **ordenar listas**, y actúa modificando la lista original, no devuelve una nueva lista. Este método sólo funciona con listas.

Por defecto ordena en orden ascendente. Puede ordenar en orden descendente utilizando `reverse=True`.

Ejemplo:

...

```
1 #Ejemplo de sort()
2 peliculas = ['Titanic', 'Alba', 'El Señor de los Anillos',
3             'Mejor imposible']
4 peliculas.sort()
5 print(peliculas)
6 peliculas.sort(reverse=True)
7 print(peliculas)
```

```
python3 eje...
['Alba', 'El Señor de los Anillos', 'Mejor imposible', 'Titanic']
['Titanic', 'Mejor imposible', 'El Señor de los Anillos', 'Alba']
```

...

La función de ordenación **sorted()**, funciona tanto con listas como con tuplas, diccionarios, cadenas, etc., y actúa devolviendo una nueva lista ordenada sin modificar el original.

Al igual que `sort()` ordena por defecto ascendentemente, y también tiene la opción `reverse=True` para ordenar de modo descendente.

Ejemplo:

...

```
1 #Ejemplo de sorted()
2 names = ('Juan', 'Maria', 'Pedro', 'Ana',)
3 sorted_names = sorted(names)
4 print(sorted_names)
5 sorted_names_reverse = sorted(names, reverse=True)
6 print(sorted_names_reverse)
```

```
python3 eje...
['Ana', 'Juan', 'Maria', 'Pedro']
['Pedro', 'Maria', 'Juan', 'Ana']
```

...

¿Qué es un operador de reasignación?

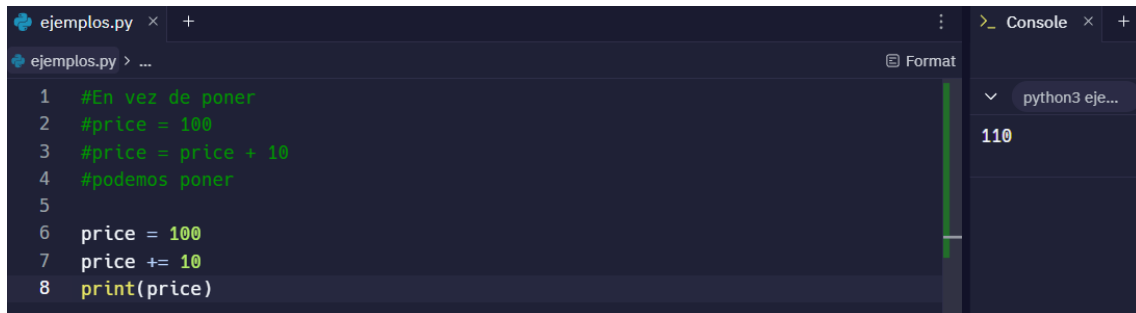
Un operador de reasignación es un código más corto, y por lo tanto más fácil de leer, que sirve para reasignar un valor a una variable.

Al evitar la repetición de la variable, conseguimos simplificar el código.

Los operadores más comunes son `+=`, `-=`, `*=`, `/=`, `//=`, `**=`, siendo los dos primeros los más utilizados.

Por ejemplo:

...



The screenshot shows a code editor with a file named 'ejemplos.py'. The code in the editor is as follows:

```
1 #En vez de poner
2 #price = 100
3 #price = price + 10
4 #podemos poner
5
6 price = 100
7 price += 10
8 print(price)
```

On the right side, there is a 'Console' panel showing the output of the script, which is '110'. The console title is 'python3 eje...'.

...