CHECKPOINT 4:

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

The main difference between a list and a tuple in python is that, while lists are mutable, tuples are immutable.

The syntax for creating lists is to use brackets [], while tuples use parentheses ().

In order to choose what is best for you in terms of using a list or a tuple, you should consider if the data structure you are going to work with should be able to be changed or if has to remain unchanged.

If you need to work with a data set that should remain unchanged, you should use a tuple as the example shown below:

tuple = ('January', 'February', 'March', 'April')

In case you know the data you are working with need to be collected in a structure that allows you to perform changes to the original given data, then you should use a list as the one shown below as an example:

list = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

Here are some of the methods you can perform to lists in order to make changes into list data structures (all the examples shown are performed to the original list):

- <u>Sort the items of the list in an ascendant order:</u>

list.sort()

print(list) # Result: ['Friday', 'Monday', 'Thursday', 'Tuesday', 'Wednesday']

- <u>Add elements to the list:</u>

list.extend([“Saturday”])

print(list) # Result: list = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']

- <u>Remove elements from the list:</u>

list.remove(“Monday”)

print(list) # Result: list = ['Tuesday', 'Wednesday', 'Thursday', 'Friday']

- <u>Remove the last element from the list that can be stored in another variable to use it later:</u>

popped_day = list.pop()

print(popped_day)) # Result: Friday

print(list) # Result: list = ['Tuesday', 'Wednesday', 'Thursday', 'Friday']

¿Cuál es el orden de las operaciones?

In Python, when you need to calculate an operation, in order to remember the order in which Python performs the operators, is useful to remember the word PEMDAS. This word is created by using the first letters of each word of the sentence *Please Excuse My Dear Aunt Sally*.

The order of the letters in the word PEMDAS follow the same order as the operators are calculated in an operation in Python as it's shown below:

Please -> Parentheses ()

Excuse -> Exponents **

My -> Multiplication *

Dear -> Division /

Aunt -> Addition +

Sally -> Subtraction -

The order, starting in P and ending in S, is the same one that follows Python when calculates an operation.

The first numbers operated are the ones inside of parentheses. Then the exponents are calculated. Later on, multiplications and divisions are performed. Once we have an operation only remaining additions and subtractions, the operation is resolved from left to right.

In the following example we are going to explain and resolve step by step how the operation is calculated, showing the order used in Python to resolve it:

This is the starting operation we are going to resolve:

(4 + 6) ** 2 - 12 + 5 * 4

First, the parenthesis is operated:

10 ** 2 - 12 + 5 * 4

Now, that the parenthesis is resolved, the exponent is calculated:

100 - 12 + **5 * 4**

Following with the mentioned order, now is time to operate the multiplication:

100 - 12 + 20

Once we only have the addition and subtraction, the operation is calculated from left to right:

108

At the end, we have reach to the result of the operation.

¿Qué es un diccionario Python?

Dictionaries in Python are used to store data values associated to a key (key: value).

The syntax for creating dictionaries is to use curly brackets {}.

Considering that each element of a dictionary has a value associated to a key and that the syntax for creating dictionaries is to use curly brackets, here’s how a dictionary with two elements looks like:

```
 dictionary = {
    "key_1": "value_1",
    "key_2": “value_2”,
}
```

The values associated to each key of an item can be any kind of data type. They can be strings, integers, floats, booleans, lists, tuples or even they could be another dictionary.

Here's a dictionary example with four elements, each of the values associated to a key being a different type of data as string, integer, float and list:

```
dictionary = {
    "name": "Luke",
    "age": 25,
    "height": 1.87,
    "studies": ["business", "law"]
}
```

Dictionaries, as lists, are mutable so we can add, remove or change items after the dictionary has been created. Here are a few examples of how dictionaries can be modified (all the examples shown are performed to the original dictionary):

- <u>Here’s an example of how we can add an element to our dictionary:</u>

```
dictionary[“languages”] = [“Spanish”, “English”, “Basque”]
print(dictionary) # Result:
dictionary = {
    "name": "Luke",
    "age": 25,
    "height": 1.87,
    "studies": ["business", "law"]
    “languages”: [“Spanish”, “English”, “Basque”]
}
```

- You can also remove elements from the dictionary as it’s showed here:

```
del dictionary[“height”]

print(dictionary) # Result:

dictionary = {

   "name": "Luke",

   "age": 25

   "studies": ["business", "law"]

}
```

- In order to know the number of elements in a dictionary you can perform the len() function. Here's how it works for the defined dictionary:

```
print(len(dictionary)) # Result: 4
```

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

While sort() method only can be applied for lists, sorted function admits any iterable.

The difference between both sorting techniques is that, while sort() method sorts a created list, sorted function, when used with lists, creates a new sorted list from an already created list in a new variable.

In both cases you can sort in ascendant or descendant order.

With the following list, we are going to walk through some examples of how the elements of the list can be sorted by using both sorting techniques:

```
list = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

- You can sort the list by using sort method in an ascendant order:

```
list.sort()

print(list) # Result: ['Friday', 'Monday', 'Thursday', 'Tuesday', 'Wednesday']
```

- You can sort the list by using sort method in a descendant order:

```
list.sort(reverse=True)

print(list) # Result: ['Wednesday', 'Tuesday', 'Thursday', 'Monday', 'Friday']
```

- You can sort the list by using sorted function in an ascendant order:

```
new_list = sorted(list)

print(list) # Result: ['Friday', 'Monday', 'Thursday', 'Tuesday', 'Wednesday']
```

- You can sort the list by using sorted function in a descendant order:

```
new_list = sorted(list, reverse=True)

print(new_list) # Result: ['Wednesday', 'Tuesday', 'Thursday', 'Monday', 'Friday']
```

¿Qué es un operador de reasignación?

As commented in the first question, tuples are immutable, same as strings. That means that, once they are created you can't modify them, at least directly.

If you really want to modify the content of a tuple, what you can do is create a new tuple with the same name as the tuple where you want to make a change and make the changes to that new element.

This way you are creating a new tuple by just overwriting the variable name.

Here's an explained example of how you can add an element to a tuple by overwriting the variable name of an existing one:

Here’s the existing tuple:

```
tuple = ('January', 'February', 'March', 'April')
```

Let’s imagine that once the tuple is created you want to add a new month to it. This is how you can add May to the tuple, by assigning the same variable name to the variable you are creating with the change as the one we already had:

```
tuple += ("May",)
```

Here’s how you can check how the tuple looks like with the new added item:

```
print(tuple) # Result: ('January', 'February', 'March', 'April', 'May')
```

Even if you have been able to “add” a new item to a tuple, the former tuple and the one created by overriding the previous one are different objects. We can check that by using ID function. ID function gives as a reference and a unique ID for each object based on where this object is stored in the computer’s memory.

If we perform ID function for both tuples, we can see how each of them have a different ID showing that they are different objects:

Here’s the existing tuple:

```
tuple = ('January', 'February', 'March', 'April')
```

This is how we can check the ID of the tuple:

```
print(id(tuple))
```

In this case the ID is: 1836008902160.

Now, we are preforming “the change” to the tuple:

```
tuple += ("May",)
```

If we check the ID o this tuple, we can see that it’s different from the previous one, meaning that both are different objects:

```
print(id(tuple))
```

In this case the ID is: 1836006178956.

Even if you can “make changes” to an existing tuple by how we have seen with the previous example, tuples are being created for being immutable. That’s why you should be asking yourself that, if you want to perform changes to your tuples, that’s the type of collection structure you should be using or if it’s better using, for example, a list.