Preguntas Teóricas

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

   Si bien las *listas* y las *tuplas* son objetos que tienen un tipo de estructura de datos muy similar, se diferencian en un aspecto no menor: las segundas son *inmutables*, no así las primeras[1]. Ahora bien, ¿qué significa que un objeto (de *python*) sea inmutable? Básicamente, que los valores (items) asociados a dicho objeto no pueden cambiar.

   Tal vez la mejor manera de entender esta diferencia sea a través de un pequeño ejemplo de cada objeto. Definimos en primer lugar una lista, a la cual posteriormente eliminamos el último elemento a través un método llamado pop(). Como se puede ver al imprimir la lista una vez hechos los cambios, ella posee ahora solo dos elementos, y no tres, lo que deja en evidenca el caracter *mutable* de las listas.

   ```
   >>> mylist = [1, 2, 3]
   >>> mylist.pop()
   3
   >>> print(mylist)
   [1, 2]
   ```

   Veámos a continuación qué sucede en el caso de una tupla. Como veremos, al intentar aplicar el método 'pop()' nos aparece un error indicándonos que dicho objeto no poee dicho atributo, confirmándonos de ese modo, su carácter *inmutable*.

   ```
   >>> mytuple = (1, 2, 3)
   >>> mytuple.pop()
   AttributeError: 'tuple' object has no attribute 'pop'
   ```

2. ¿Cuál es el orden de las operaciones?

   El orden de las operaciones tiene que ver con el orden en que se llevan a cabo las distintas operaciones matemáticas dentro de una expresión algebraica que involucra dos o más de dichas operaciones. Para aprenderse dicho orden, los alumnos de habla inglesa utilizan a menudo la siguiente nemotecnia:

   (a) **P**lease - **P**arentesis

   (b) **E**xcuse - **E**xponencial

[1] Junto a dicha diferencia "de fondo", quizás sea bueno recordar también una diferencia más formal; a saber, que las *listas* se escriben con paréntesis cuadrado, en cambio las *tuplas* con paréntesis redondo.

(c) **M**y - **M**ultiplicación

(d) **D**ear - **D**ivisión

(e) **A**unt - **A**dición

(f) **S**ally - **S**ustracción

En otras palabras, en primer lugar se resuelven los paréntesis, luego los exponentes o potencias, a continuación la multiplicación y así sucesivamente. Es fundamental seguir dicho orden, ya que de no hacerlo, lo más probable es que se obtenga un resultado distinto (y, por ende, incorrecto).

Supongamos la siguiente expresión algebraica:

$$(2 + 3)^2 \text{ x } 4 - 50 / 5$$

Como veremos a continuaciòn, ella se resuelve en las siguientes etapas (5) o pasos según el orden antes mencionado (PEMDAS).

$$\begin{gathered}(2 + 3)^2 \text{ x } 4 - 50 / 5 \\ (5)^2 \text{ x } 4 - 50 / 5 \\ 25 \text{ x } 4 - 50 / 5 \\ 100 - 50 / 5 \\ 100 - 10 \\ 90\end{gathered}$$

En este caso, si uno hiciera por ejemplo la sustracción antes de la multiplicación o de la división, uno obtendría un resultado distinto.

Por último, me parece que es importante señalar que este principio de orden no es un principio solo de *python*, si no uno de las matemáticas en general. Dicho de otro modo, ya sea que uno resuelva la expresión anterior en *python*, en *excel* o a "lápiz y papel", uno *debe* necesariamente alcanzar el resultado antes obtenido (90).

3. ¿Qué es un diccionario Python?

En términos generales, los *diccionarios* en python son una forma para almacenar o coleccionar datos, tal como lo pueden ser las listas, las tuplas o los conjuntos. Ahora bien, se diferencian de estas últimas, como veremos a continuación, en virtud de lo que podríamos llamar su estructura así como de las características o propiedades de los datos que la forman.

La principal característica de los diccionarios es que los *items* que la forman están constituidos por dos elementos: una llave (*key*), por un lado, y un valor (*value*), por el otro; donde el primer elemento hace referencia al segundo. Veámos a continuación el ejemplo presentado previamente en los ejercicios.

```
languages = {
        "fr": "french",
        "es": "spanish",
        "la": "latin",
        "en": "english",
        "eu": "euskera",
        }
```

Es importante hacer notar, y tal como se puede observar en el ejemplo, los diccionarios en *python* se escriben utilizando los paréntesis llamados en castellano *llaves* ({ }), o *curly brackets* en inglés.

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

   Teniendo en cuenta que si bien ambos *mecanismos* (para "agruparlos" de algún modo), se asemejan entre sí al ser un tipo de modo o forma de ordenación de listas, se diferencian significativamente entre ellos, sin embargo, por el hecho de que uno modifica la lista *inicial*, en cambio el otro no lo hace en virtud de que crea una nueva.

   Para comprender mejor la diferencia recién mencionada, analizamos a continuación un ejemplo sencillo. Definiremos una pequeña lista de números (llamémosla *a*, por ejemplo) a la cual aplicaremos ambos modos de ordenación: en primer lugar el método ordenado (list.sort()) y luego la función de ordenación (sorted()).

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> print(a)
[1, 2, 3, 4, 5]
```

   Como se puede observar del ejemplo, al imprimir la lista *a* después de haber aplicado el método ordenado, dicha lista ha sido modificada: la lista *a* ya no es una lista con números desordenados en ella, sino una con números ordenados ascendentemente. Distinto es lo que sucede en el caso de la función de ordenación (sorted()).

```
>>> a = [5, 2, 3, 1, 4]
>>> sorted(a)
[1, 2, 3, 4, 5]
>>> print(a)
[5, 2, 3, 1, 4]
```

   Como se puede observar acá, la lista *a* no ha sido modificada al aplicar la función de ordenación; más bien se ha creado una nueva (sorted(a)).

```
>>> sorted(a) == [1, 2, 3, 4, 5]
True
```

5. ¿Qué es un operador de asignación?

   Los operadores de asignación, tal como su nombre lo indica, son operadores utilizados para asociar un determinado *valor* a una *variable* cualquiera. El operador de asignación más básico en python es el signo igual (=), por el cual se establece la asociación recién mencionada.

```
>>> a = 10
>>> print(a)
10
```

   Ahora bien, en python existen también lo que se llama los operadores de asignación abreviados. ¿En qué consisten dichos operadores? Son operadores que permiten no solo asignar un determinado valor a una variable, sino también realizar operaciones aritméticas. Algunos casos de dichos operadores son +=, -=, *=, /= y %=.

   Veámos un ejemplo del primer caso: (a += 5). ¿Qué quiere decir dicha expresión? Básicamente dos cosas. En primer lugar, que al valor ya asociado a la variable *a* le sumamos 5 (operación aritmética). Y en segundo lugar, que la variable *a* deja de estar asociada al valor inicial y pasa a estar asociada al valor que resulta de la suma anterior (operación asignación).

```
>>> a = 10
>>> print(a)
10
>>> a += 5
>>> print(a)
15
```