# M2C4 Python Preguntas

## M. Alexandra Dodu

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

Las listas y las tuplas son objetos que se emplean para manejar datos. Sin embargo, hay varias diferencias entre ellas.

1. En cuanto a su **sintaxis**, las listas se definen empleando corchetes `[]`, mientras que las tuplas se definen empleando paréntesis `()`.
2. En cuanto a la **finalidad de uso**, las listas se emplean para manejar datos que se puedan ir actualizando a lo largo del tiempo. En cambio, las tuplas se utilizan cuando no se quiera un colección de datos que se vaya actualizando a lo largo del tiempo. Esto nos lleva a la siguiente diferencia:
3. La principal diferencia es la **mutabilidad**. Por un lado, las listas son mutables, es decir, se pueden modificar/actualizar una vez creadas. Por otro lado, las tuplas son inmutables, es decir, no se pueden modificar los elementos que las componen.

Un ejemplo para ilustrarlo:

```
series = ['Los Simpsons', 'Juego de Tronos', 'Los Serrano']
curso_programacion = ('M2C4 Python Assignment', 'Dev Camp', 'Alexandra')

print(series)
print(curso_programacion)
```

```
['Los Simpsons', 'Juego de Tronos', 'Los Serrano']
('M2C4 Python Assignment', 'Dev Camp', 'Alexandra')
```

```
# Sustitución del primer elemento de la lista
series[0] = 'Stranger Things'
print(series)
```

```
['Stranger Things', 'Juego de Tronos', 'Los Serrano']
```

```
# Intento de sustitución del primer elemento de la tupla
curso_programacion[0] = 'Checkpoint 4'
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent
call last)
<ipython-input-5-f126e76c3164> in <cell line: 2>()
      1 # Intento de sustitución del primer elemento de la tupla
----> 2 curso_programacion[0] = 'Checkpoint 4'

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

Como se puede observar, esto da error porque las tuplas son inmutables. Es posible "añadir" nuevos elementos, pero Python en realidad los almacena en una nueva tupla, como se puede ver según los ID:

```
print(id(curso_programacion))
```

```
139474502830656
```

```
curso_programacion += ('completado',)
print(curso_programacion)

print(id(curso_programacion))
```

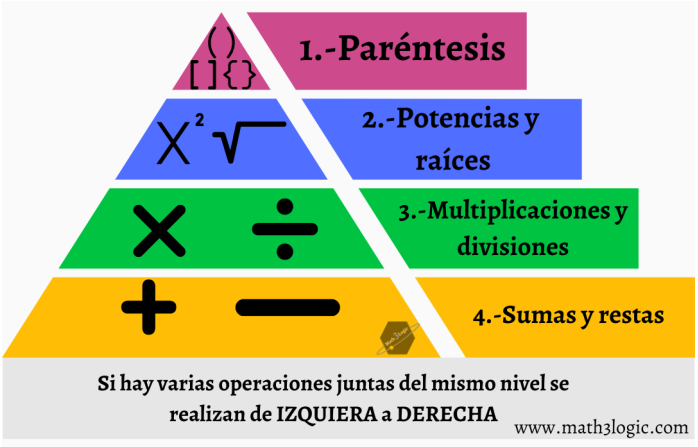
```
('M2C4 Python Assignment', 'Dev Camp', 'Alexandra', 'completado')
139473935684592
```

## ¿Cuál es el orden de las operaciones?

El orden de las operaciones en Python es el orden matemático:

1. Primero los **paréntesis**.
2. **Multiplicaciones/divisiones**, respetando el orden en el que estén.
3. Por último, **sumas/restas**, respetando el orden en el que estén.

Si hay potencias o raíces, se realizando antes que las multiplicaciones/divisiones.




```
ejemplo_operacion = 2 * ( 2**3 - 3 / 2 ) / 4 + 200
print(ejemplo_operacion)
```

```
203.25
```

## ¿Qué es un diccionario Python?

Un diccionario es un objeto que permite almacenar valores (*values*) a unas determinadas claves únicas (*keys*). Se definen empleando llaves `{}` y los pares valor/clave se asocian mediante dos puntos `:`, separando un par del otro mediante una coma `,`. Además, los diccionarios permiten almacenar diferentes tipos de datos (strings, números, listas, otros diccionarios...).

```
bases_nitrogenadas = {
    'ADN' : [ 'adenina', 'citosina', 'guanina', 'timina' ],
    'ARN' : [ 'adenina', 'citosina', 'guanina', 'uracilo' ],
    }

print(bases_nitrogenadas)
```

```
{'ADN': ['adenina', 'citosina', 'guanina', 'timina'], 'ARN': ['adenina', 'citosina', 'guanina', 'uracilo']}
```

Se puede acceder fácilmente a un valor conociendo simplemente su clave, sin la necesidad de conocer su índice de posición.

```
print(bases_nitrogenadas['ADN'])
```

```
['adenina', 'citosina', 'guanina', 'timina']
```

Otra forma de acceder a un valor es mediante el método `get()`. Esta función tiene la ventaja de que, en caso de que la clave no exista, no dará error, sino que devolverá `None` a menos que se especifique otro valor. Esto permite que un bloque de código se pueda seguir ejecutando, indicando si no se ha podido ejecutar debido a claves inexistentes.

```
print(bases_nitrogenadas['ambiguas'])
```

```
---------------------------------------------------------------------
-------
KeyError                                  Traceback (most recent
call last)
<ipython-input-38-f59c34d75ed7> in <cell line: 1>()
----> 1 print(bases_nitrogenadas['ambiguas'])

KeyError: 'ambiguas'
```

```
print(bases_nitrogenadas.get('ambiguas'))
print(bases_nitrogenadas.get('ambiguas', 'Clave inexistente'))
```

```
None
Clave inexistente
```

Si queremos acceder mediante el índice de posición, no podemos hacerlo directamente. Aunque parece una lista, no lo es, así que primero hay que convertirlo a una lista y, después, acceder al elemento que nos interese.

```
print(bases_nitrogenadas[0])
```

```
---------------------------------------------------------------------
-------
KeyError                                  Traceback (most recent
call last)
<ipython-input-30-e83ecf571ca8> in <cell line: 1>()
----> 1 print(bases_nitrogenadas[0])

KeyError: 0
```

```
# Con values() se obtienen todos los valores
print(list(bases_nitrogenadas.values())[0])
```

```
['adenina', 'citosina', 'guanina', 'timina']
```

```
# Con items() se obtienen todos los elementos del diccionario
print(list(bases_nitrogenadas.items())[0][1])
```

```
['adenina', 'citosina', 'guanina', 'timina']
```

Para conocer los valores y las claves de un diccionario, se emplea `.values()` y `.keys()`, respectivamente.

```
print(bases_nitrogenadas.values())
```

```
dict_values([['adenina', 'citosina', 'guanina', 'timina'], ['adenina', 'citosina', 'guanina', 'uracilo']])
```

```
print(bases_nitrogenadas.keys())
```

```
dict_keys(['ADN', 'ARN'])
```

Cuando se necesita iterar sobre los pares clave/valor de un diccionario, se hace uso de la función `items()` justo con el bucle `for`. Esta función nos devuelve el contenido del diccionario en formato tupla, lo que permite acceder fácilmente a sus componentes.

```

```

```
for clave, valor in bases_nitrogenadas.items():
  print(f'Bases nitrogenadas del {clave}:', valor)
```

```
Bases nitrogenadas del ADN: ['adenina', 'citosina', 'guanina', 'timina']
Bases nitrogenadas del ARN: ['adenina', 'citosina', 'guanina', 'uracilo']
```

Además, los diccionarios son mutables, por lo que se pueden modificar o actualizar.

```
# Añadir nuevo par clave/valor
bases_nitrogenadas['ambiguas'] = [ 'N', 'M']
print(bases_nitrogenadas)
```

```
{'ADN': ['adenina', 'citosina', 'guanina', 'timina'], 'ARN': ['adenina', 'citosina', 'guanina', 'uracilo'],
```

```
# Sustitución
bases_nitrogenadas['ambiguas'] = [ 'O', 'P']
print(bases_nitrogenadas)
```

```
{'ADN': ['adenina', 'citosina', 'guanina', 'timina'], 'ARN': ['adenina', 'citosina', 'guanina', 'uracilo'],
```

```
# Eliminación
del bases_nitrogenadas['ambiguas']
print(bases_nitrogenadas)
```

```
{'ADN': ['adenina', 'citosina', 'guanina', 'timina'], 'ARN': ['adenina', 'citosina', 'guanina', 'uracilo']}
```

Por tanto, los diccionarios son objetos flexibles, mutables, heterogéneos, que permiten acceder fácilmente a cualquier dato.

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

El método ordenado se define mediante `sorted()` y la función de ordenación, mediante `.sort()`. Las diferencias entre ellas son:

- El método ordenado se puede emplear sobre diferentes tipos de datos, mientras que la función de ordenación sólo se puede emplear sobre listas.
- El método ordenado devuelve el objeto ordenado en, por ejemplo, si se trata de una lista, una nueva lista. Sin embargo, la función de ordenación ordena el objeto original y lo modifica.

```
series_tele = ['Los Simpsons', 'Juego de Tronos', 'Los Serrano']

print(sorted(series_tele))

print(series_tele)
```

```
['Juego de Tronos', 'Los Serrano', 'Los Simpsons']
['Los Simpsons', 'Juego de Tronos', 'Los Serrano']
```

Se puede observar que con `sorted()` el elemento original series_tele no se ve actualizado. Si empleamos ahora la función de ordenación `.sort()`, se puede observar que el elemento original sí que se ve modificado:

```
print(series_tele)

series_tele.sort()

print(series_tele)
```

```
['Los Simpsons', 'Juego de Tronos', 'Los Serrano']
['Juego de Tronos', 'Los Serrano', 'Los Simpsons']
```

Por tanto, hay que tener cuidado al emplear la función de ordenación, ya que los índices de posición anteriores ya no corresponden a los mismos elementos de la lista.

## ¿Qué es un operador de reasignación?

Un operador de reasignación es un signo o conjunto de signos (operadores) que sirven para modificar o actualizar un objeto.

El operador de reasignación más común es el `=`, que se utiliza principlamente para asignar variables/objetos. Otro uso puede ser, en el caso de objetos mutables, para modificar/reasignar valores. Obviamente, en objetos inmutables como tuplas o strings esto no es posible. Por ejemplo, en el caso de una lista:

```
print(series_tele)

series_tele[0] = 'Sweet Tooth'

print(series_tele)
```

```
['Juego de Tronos', 'Los Serrano', 'Los Simpsons']
['Sweet Tooth', 'Los Serrano', 'Los Simpsons']
```

Otro operador de reasignación es `+=`, que se emplea para añadir un elemento al final de un objeto. Este se puede utilizar también sobre objetos inmutables, como en el caso de una tupla. Este operador es la simplificación de escribir `objeto = objeto + 'nuevo valor'`.

```
curso_programacion = ('M2C4 Python Assignment', 'Dev Camp', 'Alexandra')
print(curso_programacion)

curso_programacion += ('completado',)
print(curso_programacion)
```

```
('M2C4 Python Assignment', 'Dev Camp', 'Alexandra')
('M2C4 Python Assignment', 'Dev Camp', 'Alexandra', 'completado')
```