Explicación 3

Algunas colecciones y herramientas para iterar

In [1]:
```
# Este bloque se puede ignorar, solamente carga datos para los ejemplos
import csv
import gzip
import os
from collections import defaultdict


DIRECTORY = os.getcwd()
DATA_FILE = os.path.join(DIRECTORY, "ep03-data", "goalscorers.csv.gz")

jugadores_por_pais = defaultdict(set)

# https://www.kaggle.com/datasets/martj42/international-football-results-fro
# (CC0: Public Domain)
with gzip.open(DATA_FILE, "rt", newline="", encoding="utf-8") as f:
    reader = csv.reader(f)
    gol_header = next(reader)
    gol_data_rows = [row for row in reader]
    dates = [date for date, *_ in gol_data_rows]
    teams = [row[3] for row in gol_data_rows]
    scorer = [row[4] for row in gol_data_rows]
    for data_row in gol_data_rows:
        jugadores_por_pais[data_row[3]].add(data_row[4])
```

Algunas colecciones

- tuple
- set
- dict

Tuplas

- Permiten agrupar valores
- Son iterables
- Una vez creadas no se les puede agregar ni quitar elementos

In [2]:
```
nombre_y_apellido = ("Fernando", "López")
# Es lo mismo que:
# nombre_y_apellido = "Fernando", "López"
# Esto también es posible
# nombre_y_apellido = tuple(["Fernando", "López"])
```

Conjuntos

- Permiten agrupar valores no-repetidos
- Acceso rápido y eficiente para saber si un elemento está en el conjunto
- Son iterables
- Operaciones habituales de conjuntos: unión, intersección, pertenencia, issubset, etc...
- add/remove/clear

In [3]:
```
impares = {1, 3, 5, 7, 9}
# También se puede hacer
# impares = set(range(1, 10, 2))
```

In [4]:
```
impares.add(3)
impares
```

Out[4]:
```
{1, 3, 5, 7, 9}
```

In [5]:
```
impares.union({23})
```

Out[5]:
```
{1, 3, 5, 7, 9, 23}
```

In [6]:
```
impares.intersection({3, 9, 22})
```

Out[6]:
```
{3, 9}
```

In [7]:
```
5 in impares
```

Out[7]:
```
True
```

Cómo crear un set() a partir de un string

In [8]:
```
cadena_nombres = "Fernando, Lucas, María, Camila, Eduardo, Lucas"
```

In [9]:
```
nombres = cadena_nombres.split(", ")  # Primero lo convertimos en lista con
```

In [10]:
```
set(nombres)
```

Out[10]:
```
{'Camila', 'Eduardo', 'Fernando', 'Lucas', 'María'}
```

In [11]:
```
# Si quisieramos el conjunto de caracteres (sin discriminar mayúsculas y mir
set(cadena_nombres.lower())
```

Out[11]:
```
{' ',
 ',',
 'a',
 'c',
 'd',
 'e',
 'f',
 'i',
 'l',
 'm',
 'n',
 'o',
 'r',
 's',
 'u',
 'í'}
```

Diccionarios

- Permiten guardar valores asociados a una "clave"
- Acceso rápido y eficiente para agregar, borrar u obtener un elemento dada su "clave"
- Son iterables

In [12]:
```
gol = {
    "date": "2022-12-18",
    "home_team": "Argentina",
    "away_team": "France",
    "team": "France",
    "scorer": "Kylian Mbappé",
    "minute": "118",
    "own_goal": "FALSE",
    "penalty": "TRUE",
}

# También se puede escribir
# gol = dict([("date", "2022-12-18"), ("home_team", "Argentina"), ...])
```

In [13]:
```
gol["team"]
```

Out[13]:
```
'France'
```

In [14]:
```
# Un dict() cuya clave es un str y cuyo valor es una tupla
x = {
    "a": (1, 2, 3, 4, 5),
}

x["a"][2]
```

Out[14]:
```
3
```

In [15]:
```
# Accedemos al mismo elemento pero de una forma un poco más larga
t = x["a"]
t[2]
```

Out[15]: 3

In [16]:
```
# Usamos items() para iterar en cada par clave-valor de un diccionario
for clave, valor in gol.items():
    print(f"Para la clave {clave} el valor es {valor}")
```

```
Para la clave date el valor es 2022-12-18
Para la clave home_team el valor es Argentina
Para la clave away_team el valor es France
Para la clave team el valor es France
Para la clave scorer el valor es Kylian Mbappé
Para la clave minute el valor es 118
Para la clave own_goal el valor es FALSE
Para la clave penalty el valor es TRUE
```

In [17]:
```
# items() devuelve un iterable que en cada iteración devuelve una tupla con
gol.items()
```

Out[17]:
```
dict_items([('date', '2022-12-18'), ('home_team', 'Argentina'), ('away_tea
m', 'France'), ('team', 'France'), ('scorer', 'Kylian Mbappé'), ('minute',
'118'), ('own_goal', 'FALSE'), ('penalty', 'TRUE')])
```

In [18]:
```
# En el ejemplo anterior aprovechamos el unpacking de tuplas para que la cla
# en otra
l = [
    [1, 2, [1, 2]],
    [3, 4, [3, 5]],
    [6, 7, [6, 7]],
]

for x, z, y in l:
    print(f"{x=} {z=} {y=}")
```

```
x=1 z=2 y=[1, 2]
x=3 z=4 y=[3, 5]
x=6 z=7 y=[6, 7]
```

In [19]:
```
# Para que el unpacking funcione tengo que tener la misma cantidad de elemen
l = [
    [1, 2, [1, 2]],
    [3, 4, [3, 5]],
    [6, 7],
]

for x, z, y in l:
    print(f"{x=} {z=} {y=}")
```

```
x=1 z=2 y=[1, 2]
x=3 z=4 y=[3, 5]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[19], line 8
      1 # Para que el unpacking funcione tengo que tener la misma cantidad de
 elementos que de variables:
      2 l = [
      3     [1, 2, [1, 2]],
      4     [3, 4, [3, 5]],
      5     [6, 7],
      6 ]
----> 8 for x, z, y in l:
      9     print(f"{x=} {z=} {y=}")

ValueError: not enough values to unpack (expected 3, got 2)
```

Algunas formas de construir/procesar colecciones

- Con estructuras de control
- zip, enumerate
- map, filter y reduce
- Por comprensión
- max, min, sum
- Otras funciones útiles: sorted, reversed

Con estructuras de control

- La típica forma procedural.
- No hay ninguna abstracción.
- Puede generar una colección, o más de una, o ninguna o hacer una mezcla de cosas.

In [21]:
```
numeros_divisibles_por_5 = []
for numero in range(20):
    if numero % 5 == 0:
        numeros_divisibles_por_5.append(numero)
numeros_divisibles_por_5
```

Out[21]:
```
[0, 5, 10, 15]
```

In [22]:
```
numeros_y_su_cuadrado = {}
for numero in range(20):
    numeros_y_su_cuadrado[numero] = numero ** 2

numeros_y_su_cuadrado
```

Out[22]:
```
{0: 0,
 1: 1,
 2: 4,
 3: 9,
 4: 16,
 5: 25,
 6: 36,
 7: 49,
 8: 64,
 9: 81,
 10: 100,
 11: 121,
 12: 144,
 13: 169,
 14: 196,
 15: 225,
 16: 256,
 17: 289,
 18: 324,
 19: 361}
```

In [23]:
```
for numero, cuadrado in numeros_y_su_cuadrado.items():
    print(f"{numero=} {cuadrado=}")
```

```
numero=0 cuadrado=0
numero=1 cuadrado=1
numero=2 cuadrado=4
numero=3 cuadrado=9
numero=4 cuadrado=16
numero=5 cuadrado=25
numero=6 cuadrado=36
numero=7 cuadrado=49
numero=8 cuadrado=64
numero=9 cuadrado=81
numero=10 cuadrado=100
numero=11 cuadrado=121
numero=12 cuadrado=144
numero=13 cuadrado=169
numero=14 cuadrado=196
numero=15 cuadrado=225
numero=16 cuadrado=256
numero=17 cuadrado=289
numero=18 cuadrado=324
numero=19 cuadrado=361
```

In [ ]:
```
conjunto_de_paises_en_las_estadisticas_de_goles = set()
for pais in teams:
    conjunto_de_paises_en_las_estadisticas_de_goles.add(pais)

conjunto_de_paises_en_las_estadisticas_de_goles
```

In [ ]:
```
# El ejemplo anterior es un mal ejemplo, porque puedo construir el conjunto
# a partir de la lista `teams` directamente usando `set()`
set(teams)
```

zip y enumerate

- zip: Permite iterar al mismo tiempo en varias colecciones.
- enumerate: Permite asociar un número a cada iteración.

In [ ]:
```
# Hago un recorte de los primeros 100 datos para que los ejemplos se ejecute
# más rápido
fechas = dates[:100]
paises = teams[:100]
jugadores = scorer[:100]

# Itero al mismo tiempo en 3 listas tomando un elemento de cada una
for fecha, pais, jugador in zip(fechas, paises, jugadores):
    print(f"El día {fecha} el jugador de {pais} {jugador} hizo un gol")
```

In [27]:
```
# También puedo enumerar a los jugadores
for numero, jugador in enumerate(jugadores[:5]):
    print(f"{jugador} es el número {numero} en la tabla de goleadores")
```

```
José Piendibene es el número 0 en la tabla de goleadores
Isabelino Gradín es el número 1 en la tabla de goleadores
Isabelino Gradín es el número 2 en la tabla de goleadores
José Piendibene es el número 3 en la tabla de goleadores
Alberto Ohaco es el número 4 en la tabla de goleadores
```

map, filter y reduce

- map: Aplica una función a cada elemento de un iterable
- filter: Filtra los valores de un iterable de acuerdo al valor de retorno de una función
- reduce: Toma valores de un iterable y va realizando un calculo acumulativo.

map

La función que se le pasa a map debe recibir un argumento y retornar un valor

```
por_dos = lambda x: x * 2
resultado = map(por_dos, range(10))
```

Si ese map fuera un for...

```
resultado = []
for n in range(10):
    resultado.append(n * 2)
```

filter

La función que se le pasa a map debe recibir un argumento y retornar un valor que se interpretará como `True` o `False`

```
divisibles_por_5 = lambda x: x % 5 == 0
resultado = filter(divisibles_por_5, range(10))
```

Si ese filter fuera un for...

```
divisibles_por_5 = []
for n in range(10):
    if n % 5 == 0:
        divisibles_por_5.append(n)
```

reduce

La función que se le pasa a reduce debe recibir 2 argumentos y retornar un resultado

```
sumar = lambda a, b: a + b
resultado = reduce(sumar, range(10))
```

Si ese reduce fuera un for...

```
resultado = 0
for value in range(10):
    resultado = resultado + value
```

Colecciones por comprensión

¿Qué es "por comprensión"?

En matemática...

Def. por extensión x = {2, 4, 6}

Def. por comprensión x = Los numeros naturales menores que 7 y par

x = {n, tales que n pertenece a los naturales, si n < 7 y n mod 2 == 0}

x = {n for n in range(7) if n % 2 == 0}

En código...

For tradicional

In [28]:
```
numeros_divisibles_por_5 = []
for numero in range(20):
    if numero % 5 == 0:
        numeros_divisibles_por_5.append(numero)
numeros_divisibles_por_5
```

Out[28]:
```
[0, 5, 10, 15]
```

Lo mismo por comprensión

In [29]:
```
# Lista por comprensión
numeros_divisibles_por_5 = [
    numero
    for numero in range(20)
    if numero % 5 == 0
]
numeros_divisibles_por_5
```

Out[29]:
```
[0, 5, 10, 15]
```

In [30]:
```
# Cambiando [] por {} tenemos un set() por comprensión
numeros_divisibles_por_5 = {
    numero
    for numero in range(20)
    if numero % 5 == 0
}
numeros_divisibles_por_5
```

Out[30]:
```
{0, 5, 10, 15}
```

For anidado

In [ ]:
```
# Jugadores por pais es un diccionario que por cada país (clave)
# tiene un conjunto de sus jugadores históricos (valor)
jugadores_por_pais
```

In [32]:
```
# Construimos una lista de jugadores cuyo nombre contenga el string "Enzo"
enzos = []
for pais, jugadores in jugadores_por_pais.items():
    for jugador in jugadores:
        if "Enzo" in jugador:
            enzos.append(jugador)
enzos
```

Out[32]:
```
['Enzo Francescoli', 'Enzo Fernández', 'Enzo Scifo', 'Enzo Fernández']
```

Lo mismo por comprensión

In [33]:
```
enzos = [
    jugador
    for jugadores in jugadores_por_pais.values()
    for jugador in jugadores
    if "Enzo" in jugador
]
enzos
```

Out[33]:
```
['Enzo Francescoli', 'Enzo Fernández', 'Enzo Scifo', 'Enzo Fernández']
```

max, min y sum

In [34]:
```
numeros = [1, 4, 34, 656 ,3, -32, 2, -45, 1200, 23]
```

In [35]:
```
max(numeros)
```

Out[35]: 1200

In [36]:
```
min(numeros)
```

Out[36]: -45

In [37]:
```
sum(numeros)
```

Out[37]: 1846

sorted y reversed

In [38]:
```
sorted(numeros)
```

Out[38]: [-45, -32, 1, 2, 3, 4, 23, 34, 656, 1200]

In [40]:
```
invertidos = reversed(numeros)
list(invertidos)
```

Out[40]: [23, 1200, -45, 2, -32, 3, 656, 34, 4, 1]