Bloque 2

Lectura de ficheros (continuación) Métodos sobre tuplas, lista y conjuntos (clear/sort/sorted/max/min/sum/len)

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2024/2025

Conversiones de tipos str a otros (parsear)

(esta diapositiva se vió en la sesión anterior -está para recordar-)

Hemos visto que los ficheros se leen en formato texto (str) y hay determinados campos que por su definición en el proyecto deben ser convertidos a un tipo apropiado. Para convertir:

- A *entero*: campo=*int*(nombre)
- A *real*: campo=*float*(campo)
- A *boolean* (*True* o *False*): campo=(campo==valor en el fichero).
  Por ejemplo, en el fichero viene "Si" o "No" para el campo es_repetidor, que debe transformarse en *True* o *False*: es_repetidor=(es_repetidor=="Si"). (*Transforma "Si" en True y en False cualquier otro valor.)*
- *Cambio de un carácter o una cadena por otra: campo.replace("cadena/carácter a cambiar", "cadena/carácter nuevo")*
  Por ejemplo: Si el campo importe contiene "12,36", se convierte en un número real Python con: importe=float(importe.*replace*("," , ".")) *(los reales separan la parte entera de la decimal por un punto).*

Conversiones de tipos str a otros (parsear)

(esta diapositiva se vió en la sesión anterior -está para recordar-)

Para convertir fecha y hora es necesario: from datetime import date, time, datetime
La conversión puede ser a fecha y hora, sólo a fecha o solo a hora

- *A datetime (fecha y hora): campo*=datetime.*strptime*(*campo*, *mascara de formato*).
  La máscara de formato tiene los siguientes especificadores: *%d* (para el día), *%m* (para el mes), *%Y* (para el año con 4 dígitos), *%y* (para el año con dos dígitos), *%H* (para la hora), *%M* (para los minutos), *%S* (para los segundos).
  - Si en el fichero viene "27/09/2024 – 16:30:15" la máscara sería : *"%d/%m/%Y – %H:%M:%S"*
  - Si en el fichero viene "27-09-24 16:30" la máscara sería : *"%d-%m-%y %H:%M"*
- A *date* (solo fecha): *campo*=datetime.*strptime*(*campo*, *mascara de formato*).*date()*
- A *time* (solo hora): *campo*=datetime.*strptime*(*campo*, *mascara de formato*).*time()*
- *Borrar espacios por la izquierda y/o derecha*: A veces, en los campos hay espacios por la izquierda y/o por la derecha. Podemos eliminarlos con el método *strip()*: campo=campo.*strip()*. *Por ejemplo*: si el campo "*nombre*" contiene " Antonio Manuel ", *nombre*=*nombre*.*strip*() hace que nombre contenga "Antonio Manuel"

Conversiones de tipos str a otros (“parsear”)

“parseo” de un_contenedor y método split()

En ocasiones es necesario separar trozos de una cadena que contiene algún carácter separador entre los trozos para almacenarlos en un contenedor (por ejemplo, una lista).

Supongamos que, al leer un archivo, determinado campo tiene el siguiente contenido: “Antonio#Ana#Paula#...” y se pretenda convertir y manejar como una lista: [“Antonio”, ”Ana”, ”Paula”,..].

El procedimiento consiste en realizar una función auxiliar *parsea_...* , utilizando el método *split()* que, invocado por una cadena y recibiendo como parámetro el carácter separador, devuelve una lista con los valores separados *que como ya sabemos se puede recorrer con un for.*

Ejemplo de construcción de la función auxiliar:

```
def parsea_nombres(cadena:str)->List[str]:
    res=list()
    for elemento in cadena.split(“#”)
        res.append(elemento)
    return res
```

En el ejemplo, se invocaría como: *nombres=parsea_nombres(nombres)*

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

Supongamos

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,’c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,’c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t’)}
```

- Método *clear()*:

  Permite borrar todos los elementos de una lista o de un conjunto. Las tuplas son inmutables:

  `tupla.clear()` → *Error*

  `lista.clear()` → *[ ]*

  `conjunto.clear()` → *set()*

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Metodo *sort()*:
  - *Modifica el orden de los elementos de una lista. Si la lista contiene, a su vez, otros contenedores se ordenan por el primero de los elementos de estos contendores.*

    `tupla.sort()` → *Error (las tuplas son inmutables: No se pueden modificar)*

    `lista.sort()` → `lista=[(1,'t'),(2,'a'),(4,'c'),(4,'c'),(5,'z'),(6,'w'),(9,'b')]`

    `conjunto.sort()` → *Error (a los conjuntos no se les puede inducir un orden)*

  - *Si se quiere ordenar en orden inverso se añade el parámetro reverse=True (por defecto es False)*

    `lista.sort(reverse=True)`→`[(9,'b'),(6, 'w'),(5,'z'),(4,'c'),(4,'c'),(2,'a'),(1,'t')]`

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Si se quiere ordenar *por alguno de los elementos* de los contenedores internos se especifica el parámetro *key=lambda e:e[i]* donde *i* es la posición de los elementos por los que ordenar (se recuerda que el primer elemento es el 0).

```
lista.sort(key=lambda e:e[1]) →
        lista=[(2,'a'),(9,'b'),(4,'c'),(4,'c'),(1,'t'),(6,'w'),(5,'z')]
```

- *Si se quiere ordenar en orden inverso se añade el parámetro reverse=True (por defecto es False)*

```
lista.sort(key=lambda e:e[1], reverse=True) →
        lista=[(5,'z'),(6, 'w'),(1,'t'),(4,'c'),(4,'c'),(9,'b'),(2,'a')]
```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Función *sorted()*:
  - *Devuelve una lista con los elementos del contenedor ordenados. Si el contenedor contiene, a su vez, otros contenedores se ordenan por el primero de los elementos de estos últimos*

```
t=sorted(tupla) → [(1,'t'),(2,'a'),(4,'c'),(4,'c'),(5,'z'),(6,'w'),(9,'b')]
l=sorted(lista) → [(1,'t'),(2,'a'),(4,'c'),(4,'c'),(5,'z'),(6,'w'),(9,'b')]
c=sorted(conjunto) → [(1,'t'),(2,'a'),(4,'c'),(5,'z'),(6,'w'),(9,'b')]
```

  - *Si se quiere ordenar en orden inverso se añade el parámetro reverse=True (por defecto es False)*

```
t=sorted(tupla,reverse=True) → [(9,'b'),(6,'w'),(5,'z'),(4,'c'),(4,'c'),(2,'a'),
                                (1,'t')]
l=sorted(lista,reverse=True) → [(9,'b'),(6,'w'),(5,'z'),(4,'c'),(4,'c'),(2,'a'),
                                (1,'t')]
c=sorted(conjunto,reverse=True) → [(9,'b'),(6,'w'),(5,'z'),(4,'c'),(2,'a'),(1,'t')]
```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Si se quiere ordenar *por otro de los elementos* de los contenedores internos se especifica el parámetro *key=lambda e:e[i]* donde *i* es la posición de los elementos por los que ordenar.

```
t=sorted(tupla,key=lambda e:e[1])→[(2,'a'),(9,'b'),(4,'c'),(4,'c'),(1,'t'),
                                   (6,'w'),(5,'z')]
l=sorted(lista,key=lambda e:e[1])→[(2,'a'),(9,'b'),(4,'c'),(4,'c'),(1,'t'),
                                   (6,'w'),(5,'z')]
c=sorted(conjunto,key=lambda e:e[1])→[(2,'a'),(9,'b'),(4,'c'),(1,'t'),
                                   (6,'w'),(5,'z')]
```

- *Si se quiere ordenar en orden inverso se añade el parámetro reverse=True (por defecto es False)*

```
t=sorted(tupla,key=lambda e:e[1],reverse=True)→[(5,'z'),(6,'w'),(1,'t'),
                                   (4,'c'),(4,'c'),(9,'b'),(2,'a')]
l=sorted(lista,key=lambda e:e[1],reverse=True)→[(5,'z'),(6,'w'),(1,'t'),
                                   (4,'c'),(4,'c'),(9,'b'),(2,'a')]
c=sorted(conjunto,key=lambda e:e[1],reverse=True)→[(5,'z'),(6,'w'),(1,'t'),
                                   (4,'c'),(9,'b'),(2,'a')]
```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=(2,4,4,7.2)
lista=[2,4,4,7.2]
conjunto={2,4,4,7.2}
```

- *Función **sum()**:*

  *Devuelve la suma de los elementos de un contenedor. ¡**OJO**! El contenedor debe ser de elementos numéricos:*

  ```
  sum(tupla) →17.2
  sum(lista) → 17.2
  sum(conjunto) → 13.2
  ```

- *Función **len()**:*

  Devuelve el número de elementos de un contenedor.

  ```
  len(tupla) →4
  len(lista) → 4
  len(conjunto) → 3
  ```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Función *max() / min()*:
  - *Devuelve el máximo/mínimo de los elementos de un contenedor. Si el contenedor contiene, a su vez otros contenedores, se devuelve el del mayor/menor según el primero de los elementos de estos contendores*

```
m=max(tupla)→(9,'b') / m=min(tupla)→(1,'t')
m=max(lista)→(9,'b') / m=min(tupla)→(1,'t')
m=max(conjunto)→(9,'b')     / m=min(tupla)→(1,'t')
```

  - Si se quiere *el máximo/mínimo por otro de los elementos* de los contenedores internos se especifica el parámetro *key=lambda e:e[i]* donde *i* es la posición del elemento a buscar.

```
m=max(tupla,key=lambda e:e[1])→(5,'z')/m=min(tupla,key=lambda e:e[1])→(2,'a')
m=max(lista,key=lambda e:e[1])→(5,'z')/m=min(lista,key=lambda e:e[1])→(2,'a')
m=max(conjunto,key=lambda e:e[1])→(5,'z')/m=min(conjunto,key=lambda e:e[1])→
                                                                    (2,'a')
```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

Si los contenedores tuviesen nombre (*NamedTuple*) toda referencia a posición se puede sustituir por el nombre del campo.

Por ejemplo: Si las tuplas hubiesen sido creadas con:

Par=*NamedTuple*('pareja', *[('número'*,int*)*, ('*letra*',str)]

```
tupla=(Par(2,'a'),Par(5,'z'),Par(4,'c'),Par(9,'b'),Par(4,'c'),Par(6,'w'),Par(1,'t'))
lista=[Par(2,'a'),Par(5,'z'),Par(4,'c'),Par(9,'b'),Par(4,'c'),Par(6,'w'),Par(1,'t')]
conjunto={Par(2,'a'),Par(5,'z'),Par(4,'c'),Par(9,'b'),Par(6,'w'),Par(1,'t')}
  t=sorted(tupla,key=lambda e:e.letra)→[Par(2,'a'),Par(9,'b'),Par(4,'c'),
                                        Par(4,'c'),Par(1,'t'),Par(6,'w'),Par(5,'z')]
  t=sorted(tupla,key=lambda e:e.letra, reverse=True)→[Par(5,'z'),Par(6,'w'),
                                        Par(1,'t'),Par(4,'c'),Par(4,'c'),Par(9,'b'),Par(2,'a')]
  m=max(lista,key=lambda e:e.letra)→ Par(5,'z')
  m=min(conjunto,key=lambda e:e.letra)→ Par(2,'a')
```

Ejercicio

Importe a VSC el proyecto *T08_VacunasCovidExtendido.zip,* mediante las siguientes pautas:

1. *Descargue* el proyecto de EV en una carpeta distinta a donde lleva sus proyectos
2. *Descomprímalo*
3. *Copie* la carpeta *T08_VacunasCovidExtendido* a la carpeta donde realiza sus proyectos Python
4. Lea el enunciado en el fichero *README.md y haga lo que se le pide.*