Bloque 2

Lectura de ficheros

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2024/2025

Lectura de Ficheros

Para poder leer o escribir en un fichero en cualquier lenguaje de programación es necesario "abrir" el fichero, indicado la "dirección" donde se ubica.

De las diversas funciones que ofrece Python para abril un fichero utilizaremos *open* dentro de un bloque *with-as*:

```
with open(nombre_fichero, modo, codificación) as f :
```

nombre_fichero : es el nombre (incluyendo la ruta donde se ubica el fichero en nuestro ordenador)

f: es un descriptor para hacer referencia al fichero en el resto del programa.

modo: es una cadena con dos caracteres "XY" con los siguientes valores:

- Primer carácter (X): r, w o a (r=lectura / w=sobreescribir / a=añadir detrás)
- Segundo carácter(Y): t o b (t=archivo de texto / b=archivo binario)

Si se omite el modo, el valor por defecto es "*rt*" (*modo lectura de un fichero de texto*)

Codificación: Python toma la codificación del sistema operativo. No obstante, la codificación se puede indicar (nuestros archivos serán codificados con utf-8 con lo que podremos *encoding='utf-8'*)

Lectura de Ficheros (*estructura de nuestros ficheros en FP*)

Los ficheros de textos que vamos a leer tienen en general la siguiente estructura:

1. *Una cabecera con los nombres de los campos (aunque puede no existir)*
2. *Están formados por líneas (se ven en un editor de texto una debajo de otra)*
3. *Cada línea tiene separadores para diferenciar un campo de otro. Normalmente un coma (,) o (;) pero pueden tener otros como: # - / _,...etc, Ello permitirá "trocear" la línea en campos independientes mediante el método reader de la biblioteca csv:*

   `csv.reader`(descriptor del fichero, [`delimiter`="separador"])

4. *Devuelve un iterable (un contenedor)*
5. *Es necesario importar la librería csv" (del inglés) "valores separados por coma".:import csv.*

   Según esto, un fichero .csv debería de tener lo campos separados por coma (,) aunque se abusa de su nombre y también se usan otros separadores como (;)

   Si los campos del fichero están separados efectivamente por una coma (,) se puede omitir el parámetro delimiter

Lectura de Ficheros (*estructura de nuestros ficheros en FP*)

Ejemplo de fichero de Datos Personales (con cabecera y campos separados por punto y coma)

```
DNI;NOMBRE;APELLIDOS;EDAD;ESTATURA;PESO;LOCALIDAD;PROVINCIA
12345678A;JUAN;AFAN POSTIGO;22;1.78;69.9;SEVILLA;SEVILLA
12345678B;NICOLAS;AGUILAR SAUCEDO;20;1.59;76.7;DOS HERMANAS;SEVILLA
12345678C;LUCAS;ACEJO GARCÍA;20;1.99;65.4;UTRERA;SEVILLA
12345678D;CLAUDIA;ÁLVAREZ GARCÍA;21;1.94;57.9;VISO DEL ALCOR;SEVILLA
12345678E;PAULA;ALBENDÍN CAMINO;19;1.73;83;TOMARES;SEVILLA
12345678F;ANA;LOBATO ÁLVAREZ;18;1.65;78.9;PUNTA UMBRÍA;HUELVA
12345678G;ANTONIO;DÍAZ NARANJO;18;1.77;51.3;CHIPIONA;CÁDIZ
12345678H;SOFÍA;GUERRERO CANTARERO;20;1.56;72.7;CHIPIONA;CÁDIZ
12345678I;JOSÉ;FIERRO ÁLVAREZ;22;1.74;78.6;SEVILLA;SEVILLA
12345678J;CARLOS;GUERRERO PAREDES;22;1.69;59.7;DOS HERMANAS;SEVILLA
```

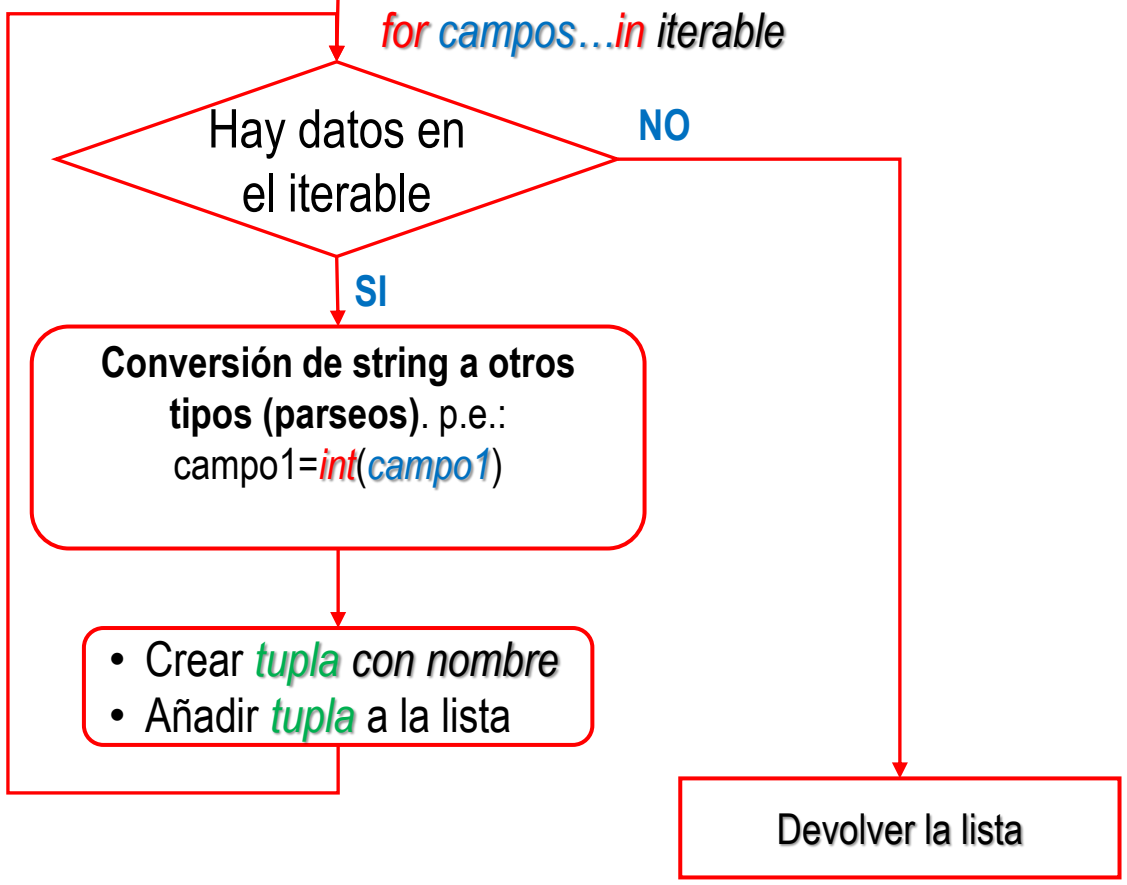
Ejemplo de fichero de Datos de Censo de poblaciones (sin cabecera y campos separados por coma)

```
Arab World,ARB,1960,92490932
Arab World,ARB,1961,95044497
Arab World,ARB,1962,97682294
Arab World,ARB,1963,100411076
Arab World,ARB,1964,103239902
Arab World,ARB,1965,106174988
Arab World,ARB,1966,109230593
Arab World,ARB,1967,112406932
Arab World,ARB,1968,115680165
Arab World,ARB,1969,119016542
```

Lectura de Ficheros (*esquema de lectura*)

- Crear una lista vacía: *list()*
- Crear un bloque *with* y abrir el archivo con *open*
- Obtener un *iterable*: *csv.reader*
- Si hay cabecera, saltar con *next*

fichero.csv



Lectura de Ficheros (*Ejemplo 1 de lectura -campos separados por (,)-)*

Supongamos un archivo con las siguientes 4 primeras líneas

```
1   País,Código,año,censo
2   Austria,AUT,2000,8011566
3   Austria,AUT,2001,8042293
4   Austria,AUT,2002,8081957
```

El siguiente tipo: Población = NamedTuple("población",[('país',str),('código',str),('año',int),('censo',int)])

```
def lee_poblaciones (nombre_fichero:str)->List[Población]:
    res=list()                          #Se crea una lista vacía
    with open(nombre_fichero,'rt',encoding='utf-8') as f:
        iterable=csv.reader(f) #trocea las líneas en 4 campos y se guardan en un contenedor
        next(iterable)              #salta la 1ª línea 1 del contenedor (la de cabecera)
        for nombre, código, año, censo in iterable: #Se va leyendo cada línea del contenedor
            res.append(Población(nombre, código, int(año), int(censo)))
    return res
```

Lectura de Ficheros (*Ejemplo 2 de lectura -campos separados por (#)-)*

Supongamos un archivo con las siguientes 4 primeras líneas

```
1  País#Código#año#censo
2  Austria#AUT#2000#8011566
3  Austria#AUT#2001#8042293
4  Austria#AUT#2002#8081957
```

El siguiente tipo: Población = NamedTuple("población",[('país',str),('código',str),('año',int),('censo',int)])

```
def lee_poblaciones (nombre_fichero:str)->List[Población]:
    res=list()                          #Se crea una lista vacía
    with open(nombre_fichero,'rt',encoding='utf-8') as f:
        iterable=csv.reader(f, delimiter="#")) #trocea las líneas en 4 campos y se guardan en un contenedor
        next(iterable)          #salta la 1ª línea 1 del contenedor (la de cabecera)
        for r in iterable: #Se va leyendo cada línea del contenedor
            res.append(Población( r[0], r[1], int(r[2]), int(r[3])))
    return res
```

Usando una tupla genérica "**r**" y accediendo a cada campo por su posición r[i]

Ejemplo de estructura de test con lectura de fichero

```
def test_lee_poblaciones(datos:List[Población])->None:
  print("\nTotal poblaciones leídas:",len(datos))
  print("Las dos primeras: ",datos[:2])
  print("Las dos últimas:",datos[-2:])

def test_filtra_por_país(datos:List[Población])->None:
  ...

def test_filtra_por_año_y_umbral_de_censo(datos:List[Población])->None:
  ...
if __name__=='__main__':
  poblaciones=lee_poblaciones("TNN_Población/data/poblaciones1.csv")
  test_lee_poblaciones(poblaciones)
  test_filtra_por_país(poblaciones)
  test_filtra_por_año_y_umbral_de_censo(poblaciones)
```

La variable poblaciones guarda la lista que devuelve lee_poblaciones y se pasa como parámetros a los test

Ruta desde la carpeta en donde se ha abierto VSC

Ejercicio:

- Copie y pegue el proyecto *T07_Población* con el nombre *T07_Población_con_fichero*
- Cree una carpeta *data* y copie en ella el fichero *poblaciones1.csv* cuyos primeros registros son:

```
1   País,Código,año,censo
2   Austria,AUT,2000,8011566
3   Austria,AUT,2001,8042293
4   Austria,AUT,2002,8081957
```

- En el módulo *poblaciones.py*:
  - Modifique la función *lee_poblaciones* para que reciba como parámetro el nombre del fichero, (incluyendo la ruta) y en lugar de devolver una lista con los datos predefinidos, devuelva la lista con los registros leídos del fichero.
- En el módulo *test_poblaciones.py*
  - Modifique la llamada a *lee_poblaciones* para pasar como parámetro el nombre (incluyendo la ruta) del fichero *poblaciones1.csv*.
  - Ejecute el test y debe funcionar exactamente tal y como funciona T07_Población.
- Por último, copie en la carpeta *data* *poblaciones2.csv* y pruebe el test nuevamente. Debe funcionar igual, pero con más datos.

Conversiones de tipos str a otros (parsear)

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)

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"

Ejercicio

Importe a VSC el proyecto *T08_VacunasCovid,* mediante las siguientes pautas:

1. *Descargue* el proyecto de EV
2. *Descomprímalo*
3. *Cópielo* a la carpeta donde realiza sus proyectos Python
4. Lea el enunciado en el fichero *README.md*

¡¡¡Animo que se puede!!!