Lectura de Ficheros

Para poder leer o escribir en un fichero es necesario abrir el fichero. De las diversas funciones Python que permiten la lectura utilizaremos:

with open(“nombre_fichero”, modo, codificación) as f :,

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 (*nuestros ficheros*)

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

1. *Una cabecera con los nombres de los campos*
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, por lo que para "trocear" la línea en campos independientes se utiliza el método:*

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

   este método csv.reader está en la librería estándar de Python "csv" (del inglés) "valores separados por coma". Por ello, antes de usar el método hay que importar la librería 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 (*Ejemplo 1 de lectura -campos separados por (,)-)*

Supongamos un archivo con las siguientes 4 primeras líneas

1. *Nombre de país,Código de país,año,número de habitantes*
2. *'Angola','AGO',1980,8929900*
3. *'Portugal','PRT',2000,10289898*
4. *'Spain','ESP',1996,39889852*
5. *…*

La siguiente namedtuple: Población=namedtuple('población','país, código, año, num_habitantes')

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

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

Supongamos un archivo con las siguientes 4 primeras líneas

1. *Nombre de país#Código de país#año#número de habitantes*
2. *'Angola'#'AGO'#1980#8929900*
3. *'Portugal'#'PRT'#2000#10289898*
4. *'Spain'#'ESP'#1996#39889852*
5. *…*

La misma namedtuple: Población=namedtuple('población','país, código, año, num_habitantes')

```
def lee_población (nombre_fichero:str)->list[Población]:
    res=list()
    with open(nombre_fichero,'rt',encoding='utf-8') as f:
        lector=csv.reader(f, delimiter="#")
        next(lector)
        for r in lector:
            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 Test básico para lectura de fichero

```
def test_lee_población(poblaciones:list[Población]):
    print("Número de registros leídos:",len(poblaciones))
    print("Los 2 primeros registros son:",poblaciones[:2])
    print("Los 3 últimos registros son:",poblaciones[-3:])

def test_....(poblaciones:list[Población]):

…

if __name__==‘__main__’:
    poblaciones=lee_población(“TNN_Poblaciones/data/population.csv”)
    test_lee_población(poblaciones)
    test_....(poblaciones)
    ….
```

Ruta desde la carpeta en donde se ha abierto VSC”

Ejercicio:

Copie y pegue el proyecto *T08_Datos_Personales* con el nombre *T09_Datos_Personales*

- Cree una carpeta "*data*" y copie en ella el fichero "*datos_personales.csv*". Este fichero tiene los siguientes campos por cada línea:

dni;nombre;apellidos;edad;estatura;peso;localidad;provincia

Todas string salvo: edad de tipo int, estatura y peso de tipo float

- En el módulo *datos_personales.py*:
  - Modifique la namedtuple *Persona* con la nueva descripción de los campos del fichero.
  - Añada una función: *lee_datos_personales* que, recibiendo el nombre de un fichero, devuelva una lista los registros leídos.
- Cree un nuevo módulo *test_datos_personales.py* (borre el que ahora hay en el proyecto)
  - Implemente un test que permita probar la función *lee_datos_personales* visualizando:
    a) El número de registros leídos
    b) El tercer registro (sin contar el registro de cabecera)
    c) Los tres primeros registros leídos
    d) Los tres últimos registros leídos

Conversiones de tipos (replace)

replace( )

En ocasiones es necesario cambiar en una cadena determinado/s carácter/es por otro/s. Por ejemplo, un archivo en el que los valores reales tiene una coma (,) para separar la parte entera de la parte decimal y Python necesita que estén separadas por punto(.) para poder tratarlos como un valor real.

Usamos el método *replace()*.

Sintaxis:

cadena.*replace*("*texto/carácter a cambiar*","*nuevo texto/carácter*")

Devuelve una nueva cadena con los caracteres cambiados.

Por ejemplo:

Si la variable str *trozo* contiene "12,36", se convierte en un número real Python con:

float(*trozo*.*replace*(",","."))

Conversiones de tipos (a boolean)

parsea_...( )

En ocasiones es necesario cambiar determinados valores (normalmente cadena de caracteres o carácter, por los valores *True* o *False*. *Por ejemplo*, al leer un archivo, determinado campo puede tomar los valores "*cierto*" o "*falso*"; o los valores "*SI*" o "*NO*" que se quiere transformar en *True* o *False.*

Sintaxis en la misma línea

es_repetidor=(es_repetidor=='cierto') (*o en el segundo ejemplo es_repetidor=="SI"*)

Sintaxis llamando a una función auxiliar:

a) es_repetidor=parsea_es_repetidor(es_repetidor)

b) y realizamos la función auxiliar *parsea_...()*.

```
def parsea_es_repetidor (cadena:str)->bool:
    res=False
    if cadena=="cierto":                (o en el segundo ejemplo cadena=="SI")
        res=True
    return res
```

Ejercicio:

Modifique el proyecto *T09_Datos_Personales*

- En la carpeta "*data*" copie el fichero "*datos_personales2.csv*". Este fichero tiene los siguientes campos por cada línea :

dni;nombre;apellidos;edad;estatura;peso;localidad;provincia;esmujer

Todas string salvo: edad de tipo int, estatura y peso de tipo float y esmujer de tipo bool

- En el módulo *datos_personales.py*:
  - Cree una nueva namedtuple *Persona2* a partir de una copia de *Persona*
  - Añada una función: *lee_datos_personales2* que, recibiendo el nombre de un fichero, devuelva una lista los registros leídos.
- En el módulo *test_datos_personales.py*
  - Añada un test que permita probar la función *lee_datos_personales2* visualizando:
    a) El número de registros leídos
    b) El tercer registro (sin contar el registro de cabecera)
    c) Los tres primeros registros leídos
    d) Los tres últimos registros leídos

Conversiones de tipos (una cadena que representa una lista en una lista)

parsea_...( ) y método split()

En ocasiones es necesario separar una cadena que contiene determinados valores en una lista. Por ejemplo, al leer un archivo, determinado campo tomar los valores: “Antonio#Ana#Paula#…” que claramente representan una lista y que se pretende convertir y manejar como tal lista: [“Antonio”, ”Ana”, ”Paula”,..]. Es necesario usar el método *split()* que invocado por el campo y recibiendo como parámetro el carácter separador, devuelve una lista con los valores separados.

Realizamos un método auxiliar *parsea_...()*.

Sintaxis de construcción de la función:

```
def parsea_...(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)*

Ejercicio:

Modifique el proyecto *T09_Datos_Personales*

- En la carpeta "*data*" copie el fichero "*datos_personales3.csv*". Este fichero tiene los siguientes campos por cada línea :

dni;nombre;apellidos;edad;estatura;peso;localidad;provincia;esmujer;hobbies

Todas string salvo: edad de tipo int, estatura y peso de tipo float, esmujer de tipo bool y bobbies de tipo list[str]

- En el módulo *datos_personales.py*:
  - Cree una nueva namedtuple *Persona3* a partir de una copia de *Persona2*
  - Añada una función: *lee_datos_personales3* que, recibiendo el nombre de un fichero, devuelva una lista los registros leídos.
- En el módulo *test_datos_personales.py*
  - Añada un test que permita probar la función *lee_datos_personales3* visualizando:
    a) El número de registros leídos
    b) El tercer registro (sin contar el registro de cabecera)
    c) Los tres primeros registros leídos
    d) Los tres últimos registros leídos