Tema 4. Almacenamiento persistente de datos.

Vicente Benjumea García

Programación-II
Departamento de Lenguajes y Ciencias de la Computación.
E.T.S.I. Informática. Univ. de Málaga.

Tema 4. Almacenamiento persistente de datos.

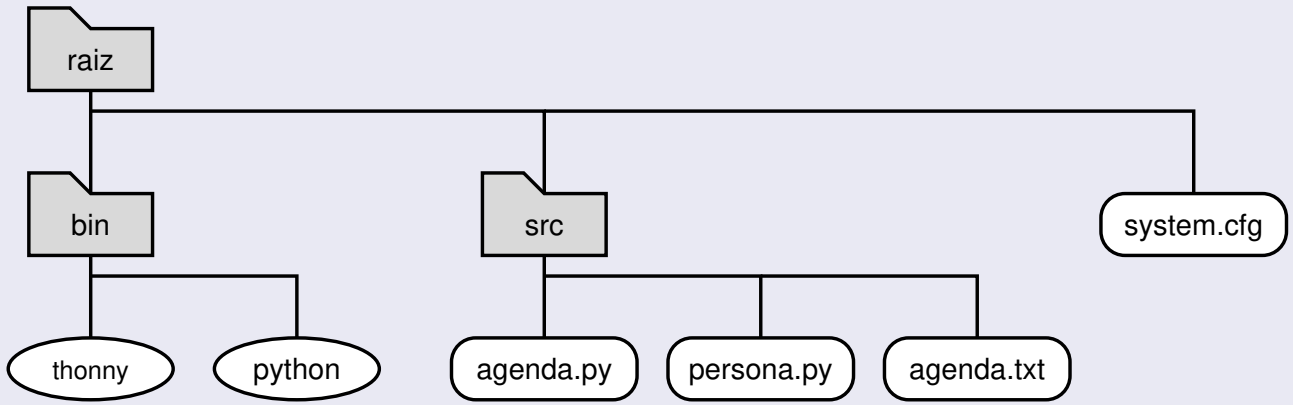
- Introducción.
- La sentencia with y los gestores de contexto.
- Lectura de datos de ficheros de texto.
- Escritura de datos en ficheros de texto.
- Formato de ficheros CSV (comma-separated values).
- Lectura de datos de ficheros CSV.
- Escritura de datos en ficheros CSV.

Introducción

- Almacenamiento de Datos en el Ordenador
  - Memoria Principal: (*acceso directo de la CPU*).
    - Tiempo de acceso muy rápido.
    - Almacenamiento **no persistente**: **volátil**.
    - Capacidad de almacenamiento limitada.
  - Memoria Secundaria: (*discos duros*, *discos ópticos*, *memorias USB*, *etc.*)
    - Tiempo de acceso lento.
    - Almacenamiento **persistente**.
    - Gran capacidad de almacenamiento.

Introducción

- Organización de la Memoria Secundaria (*gran capacidad de almacenamiento*)
  - Sistema de Ficheros
    - Jerarquía de Directorios (Carpetas) y Ficheros
    - Directorios: organizan jerarquicamente el sistema de ficheros
      Directorios, subdirectorios y ficheros
    - Ficheros: almacenamiento persistente de información
      Datos: información, configuraciones, código fuente
      Software: bibliotecas y programas ejecutables



Introducción

- La **entrada de datos** (lectura/cargar) se refiere a los datos que recibe el programa.
- La **salida de datos** (escritura/guardar) se refiere a los datos que el programa envía.
  - Ya hemos visto la entrada de teclado y la salida a pantalla.
  - Ahora vamos a tratar la entrada/salida con **ficheros**, almacenados en memoria secundaria, para el **almacenamiento de datos de forma persistente**.

- Almacenamiento de la Información. Tipos de Ficheros:
  - Ficheros de Texto
    - Codificación textual: secuencia de caracteres (ASCII/UTF-8/etc) *(Conversión)*
    - Procesamiento orientado a ordenador (también humano)
    - Representan información muy diversa (editor de textos)
  - Ficheros Binarios
    - Codificación binaria: secuencia de bytes (rep. interna del ordenador)
    - Procesamiento orientado a ordenador (problemas de compatibilidad)
    - Representan información binaria (programas, imágenes, música, etc.)

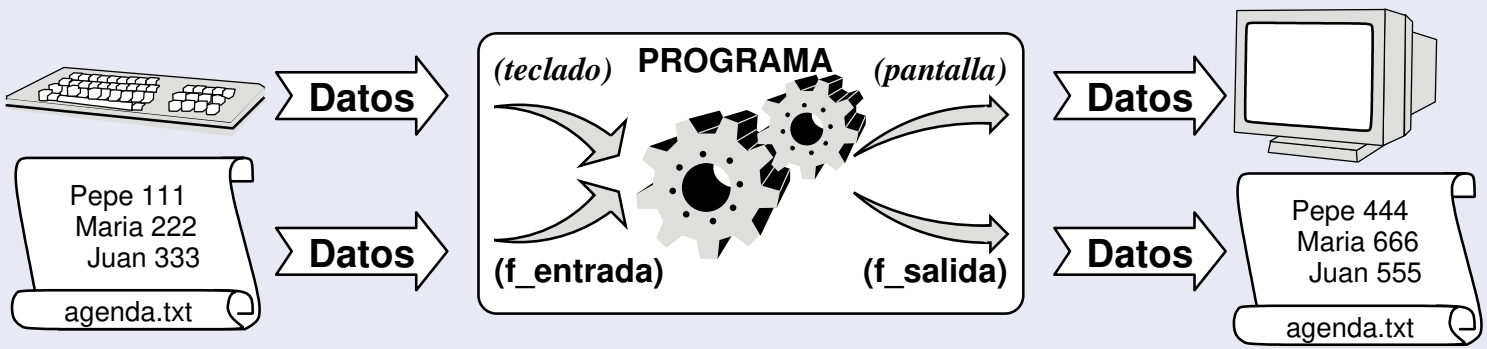
Introducción

- Los ficheros (archivos) permiten **almacenar la información** de forma **permanente** en el sistema de almacenamiento.
- Un fichero contiene cierta información **codificada**, que se almacena en memoria como una **secuencia de bytes**.
- Cada fichero recibe un **nombre** (posiblemente con una extensión) y se ubica dentro de un **directorio** (carpeta) que forma parte de una cierta **jerarquía** (ruta o camino).
- El **nombre** y la **ruta**, o secuencia de directorios, que hay que atravesar para llegar a la ubicación de un fichero, **identifican** a dicho fichero de forma única dentro del sistema de ficheros.
  - En **Windows** se utiliza la *barra invertida* \ como carácter separador en cadenas de caracteres, En el programa se debe duplicar \\, ya que representa el símbolo de escape de los caracteres.
  - En **Unix**, **Linux** y **MacOSX**, se utiliza el símbolo / para separar los componentes de la ruta del fichero.
  - Para identificar los **ficheros en el directorio de trabajo**, solo se debe especificar el nombre del fichero, no es necesario utilizar el caracter separador.

- En **Python**, utilizaremos el símbolo / para separar los componentes de la ruta del fichero, pero se invocará a la función `os.path.normpath("ruta/del/fichero.txt")` para que se genere el nombre del fichero con los separadores adecuados, según el S.O.

Introducción

- Entrada y Salida de Datos a Través de Flujos (Streams)
  - Flujos estándares de entrada (*teclado*) y salida (*pantalla*).
  - **Flujo de entrada**: *fuente* que proporciona una secuencia de caracteres
  - **Flujo de salida**: *sumidero* que recibe una secuencia de caracteres
  - Un **manejador de fichero**: *variable* que vincula un flujo de datos (de entrada o salida) con un determinado fichero.
  - Toda la transferencia de información se realiza a través de los manejadores de ficheros (a través de *buffers*).



Operaciones con ficheros

- **Apertura**: establece una *conexión* entre un *manejador del fichero* dentro del programa y un determinado fichero dentro del sistema de almacenamiento. En caso de apertura para lectura (entrada) (**"r"**), el fichero debe existir. En caso de apertura para escritura (salida) (**"w"**), se borrará o creará un nuevo fichero.
- **Escritura**: para poder almacenar información en un fichero, una vez abierto en modo de escritura, hay que transferir la información, **organizada** de alguna forma, mediante operaciones de escritura.
- **Lectura**: para poder utilizar la información contenida en un fichero, debe estar abierto en modo de lectura, y hay que utilizar las operaciones de lectura adecuadas a la **organización** de la información contenida en dicho fichero.
- **Cierre**: cuando se ha terminado de transferir la información a/desde el fichero, se debe **cerrar** la conexión previamente establecida entre la variable manejador del fichero y el fichero en el sistema de almacenamiento. Esta operación se ocupa, además, de mantener la **integridad** del fichero, escribiendo previamente la información que se encuentre en algún buffer intermedio en espera de pasar al fichero. En caso de **no cerrar adecuadamente** el manejador de fichero, entonces es posible que se **pierdan** *recursos del sistema*, que el fichero **no** guarde correctamente todos los datos enviados, y que no se pueda garantizar la **integridad** del fichero.

La sentencia with y los gestores de contexto

- Cuando se ha terminado de transferir la información a/desde el fichero, se debe **cerrar** el manejador del fichero.
- Durante la transferencia y el procesamiento de información con el fichero, podrían **lanzarse excepciones** que podrían hacer que el fichero no se cerrase adecuadamente, con los problemas que ello traería consigo.
- Aunque la cláusula **finally** podría garantizar el cierre del fichero, Python proporciona un sistema más adecuado para la **gestión de recursos**, los gestores de contexto y la sentencia **with**.
  - La función `open(os.path.normpath(nombre_fich), modo, encoding="utf-8")`, donde el *modo* puede ser `"r"` para lectura (*read*), y `"w"` para escritura (*write*), establece una conexión entre un fichero y el manejador correspondiente.
    - La función `open` lanza la excepción `OSError` si se produce error al abrir el fichero.
  - Cuando la sentencia **with** termina, el fichero se **cierra automáticamente**. La sentencia **with garantiza** que el fichero se **cerrará**, independientemente de los errores y excepciones que hayan podido surgir en su procesamiento.

La sentencia with y los gestores de contexto

Apertura de fichero para entrada (lectura)

- Utilizaremos la sentencia **with** para abrir el fichero (y cerrarlo automáticamente), y dentro del contexto leeremos líneas de texto, que procesaremos de la forma adecuada.

```
import os.path

def cargar_de_fichero(nombre_fich: str) -> None:
    with open(os.path.normpath(nombre_fich), "r", encoding="utf-8") as fich:
        ... # lectura y procesamiento de los datos del fichero (fich)

    # cuando se termina la sentencia with, el fichero se cierra automáticamente
```

Apertura de fichero para salida (escritura)

- Utilizaremos la sentencia **with** para abrir el fichero (y cerrarlo automáticamente), y dentro del contexto escribiremos líneas de texto, organizadas adecuadamente para que se pueda extraer su información posteriormente.

```
import os.path

def guardar_en_fichero(nombre_fich: str) -> None:
    with open(os.path.normpath(nombre_fich), "w", encoding="utf-8") as fich:
        ... # escritura de los datos al fichero (fich)

    # cuando se termina la sentencia with, el fichero se cierra automáticamente
```

Lectura de datos de ficheros de texto

- Utilizaremos la sentencia **with** para abrir el fichero (y cerrarlo automáticamente), y dentro del contexto leeremos líneas de texto, que procesaremos de la forma adecuada.
  - Leer un fichero de texto, y mostrar las líneas que contiene.

```
import logging
import os.path

def mostrar_fichero(nombre_fich: str) -> None:
    with open(os.path.normpath(nombre_fich), "r", encoding="utf-8") as fich:
        for linea in fich:          # lee cada línea del fichero
            print(linea, end="")    # linea ya tiene \n al final

def main() -> None:
    try:
        nombre_fich = input("Introduce nombre de fichero: ")
        mostrar_fichero(nombre_fich)
    except OSError as exc:
        logging.error(f"Fichero no encontrado: [{nombre_fich}]")
    except ValueError as exc:
        logging.error(f"Codificación de fichero errónea: [{nombre_fich}]")

if __name__ == "__main__":
    main()
```

Lectura de datos de ficheros de texto

- Leer un fichero de alumnos, donde cada línea almacena los datos de un alumno. Para cada alumno, se almacena el *dni*, el *nombre*, y varias *notas* (se calculará la nota *media*). Si alguna línea tiene un formato incorrecto, se desechará esa línea y se continuará procesando el resto de líneas.

```python
# Módulo: cargar_alumnos.py
from typing import NamedTuple, TextIO
import logging
import os.path

class Alumno(NamedTuple):
    dni: str
    nombre: str
    nota: float

def cargar_alumnos(nombre_fich: str) -> list[Alumno]:
    lista_alumnos: list[Alumno] = list()
    with open(os.path.normpath(nombre_fich), "r", encoding="utf-8") as fich:
        for linea in fich:
            try:
                alumno = procesar_linea(linea)
            except ValueError as exc:
                logging.warning(f"Formato erróneo: [{linea.strip()}]")
            else:
                lista_alumnos.append(alumno)
    return lista_alumnos
```

Lectura de datos de ficheros de texto

```python
# Módulo: cargar_alumnos.py (continuación)
def procesar_linea(linea: str) -> Alumno:
    datos = linea.split(";")
    if len(datos) < 3:
        raise ValueError("Datos insuficientes")
    suma = sum(float(x) for x in datos[2:])
    return Alumno(dni=datos[0].strip(), nombre=datos[1].strip(), nota=(suma/(len(datos)-2)))

def main() -> None:
    try:
        nombre_fich = input("Introduce nombre de fichero: ")
        lista_alumnos = cargar_alumnos(nombre_fich)
    except OSError as exc:
        logging.error(f"Fichero no encontrado: [{nombre_fich}]")
    except ValueError as exc:
        logging.error(f"Codificación de fichero errónea: [{nombre_fich}]")
    else:
        print(lista_alumnos)

if __name__ == "__main__":
    main()
```

```
1111; pepe luis; 5.5; 7.5
2222; juan luis; xxx
3333; maria luisa; 8.8
4444; xxx
5555 ; ana luisa ; 6.7 ; 8.9 ; 9.9
```

```
WARNING:root:Formato erróneo: [2222; juan luis; xxx]
WARNING:root:Formato erróneo: [4444; xxx]
[Alumno(dni='1111', nombre='pepe luis', nota=6.5),
 Alumno(dni='3333', nombre='maria luisa', nota=8.8),
 Alumno(dni='5555', nombre='ana luisa', nota=8.5)]
```

Escritura de datos en ficheros de texto

Escritura de datos a ficheros de texto

- Utilizaremos la sentencia **with** para abrir el fichero (y cerrarlo automáticamente), y escribiremos líneas de texto, organizadas adecuadamente para que se pueda extraer su información posteriormente.
  - El método `write(string)` permite escribir al fichero el *string* recibido como parámetro. **Nota:** es necesario escribir explícitamente un salto de línea (`"\n"`) al final de cada línea de texto.

```
import logging
import os.path
def guardar_texto(nombre_fich: str, lineas: list[str]) -> None:
    with open(os.path.normpath(nombre_fich), "w", encoding="utf-8") as fich:
        for linea in lineas:
            fich.write(linea)
            fich.write("\n")    # salto de línea al final de la línea

def main() -> None:
    try:
        lineas = ["pepe luis", "maria luisa", "juan luis", "ana luisa"]
        nombre_fich = input("Introduce nombre de fichero: ")
        guardar_texto(nombre_fich, lineas)
    except OSError as exc:
        logging.error(f"No se puede crear el fichero: [{nombre_fich}]")

if __name__ == "__main__":
    main()
```

Escritura de datos en ficheros de texto

Escritura de datos a ficheros de texto

- Escribir a un fichero de alumnos, donde cada línea almacena los datos de un alumno. Para cada alumno, se almacena el *dni*, el *nombre*, y la *nota media*.

```
from typing import NamedTuple, TextIO
import logging
import os.path
class Alumno(NamedTuple):
    dni: str
    nombre: str
    nota: float

def guardar_alumnos(nombre_fich: str, lista_alumnos: list[Alumno]) -> None:
    with open(os.path.normpath(nombre_fich), "w", encoding="utf-8") as fich:
        for alumno in lista_alumnos:
            try:
                fich.write(f"{alumno.dni}; {alumno.nombre}; {alumno.nota}\n")
            except ValueError as exc:
                logging.warning(f"Error de escritura: [{alumno}]")

def main() -> None:
    try:
        lista_alumnos = [Alumno(dni="1111", nombre="pepe luis", nota=6.5),
                         Alumno(dni="3333", nombre="maria luisa", nota=8.8),
                         Alumno(dni="5555", nombre="ana luisa", nota=8.5)]
        nombre_fich = input("Introduce nombre de fichero: ")
        guardar_alumnos(nombre_fich, lista_alumnos)
    except OSError as exc:
        logging.error(f"No se puede crear el fichero: [{nombre_fich}]")

if __name__ == "__main__":
    main()
```

Formato de ficheros CSV (*comma-separated values*)

- El formato **csv** es un formato de ficheros de **texto** muy habitual:
  - Es un formato de texto, que puede ser importado por muchas hojas de cálculo (*excel*, *libre-office*, etc.), es muy flexible, y se utiliza en numerosas aplicaciones de procesamiento de datos.
  - Se suele representar de forma **tabulada**.
  - Cada **fila** almacena los datos de un **registro**.
  - Los **campos** de un registro aparecen separados por **comas** (a veces por *punto-y-coma*).
  - Aquellos campos que pueden tener comas incluidas (string) se escriben entre **comillas dobles**.
  - Usualmente, la primera fila contiene la **descripción** de cada campo (el nombre de cada columna).

```
"dni","nombre","nota"
"1111","pepe luis",6.5
"3333","maria luisa",8.8
"5555","ana luisa",8.5
```

```
dni       nombre          nota
-------   -------------   ------
1111      pepe luis       6.5
3333      maria luisa     8.8
5555      ana luisa       8.5
-------   -------------   ------
```

Lectura de datos de ficheros CSV

- Para leer ficheros de texto en formato **csv**, se deben realizar las siguientes acciones:
  - Se debe importar el módulo **csv**.
  - Se debe **abrir** el fichero de texto en modo lectura (*"r"*) con la función `open` dentro de la sentencia `with`, especificando el caracter de *nueva-línea* como el string vacío (*""*), y la codificación *"utf-8"*.

```
with open(os.path.normpath(nombre_fich), "r", newline="", encoding="utf-8") as csvfile:
```

  - Dentro del contexto de la sentencia `with`, se debe crear un objeto **lector-de-CSV** asociado al fichero que se abrió en el paso anterior, especificando que se deben *entre-comillar* todos los valores que no sean numéricos:

```
csv_reader = csv.reader(csvfile, quoting=csv.QUOTE_NONNUMERIC)
```

    - `QUOTE_NONNUMERIC`: convierte a `float` los valores no entre-comillados, resto `str`.
  - A continuación, leeremos cada fila, **iterando** (en un bucle `for`) sobre el objeto **lector-de-CSV**, donde cada fila es una lista que contiene los valores de cada elemento de esa fila, que usualmente serán de tipo *string* o *float*.

```
lista_filas: list[list] = list()
for fila in csv_reader:
    lista_filas.append(fila)  # fila es una lista de valores
```

```
"dni","nombre","nota"
"1111","pepe luis",6.5
"3333","maria luisa",8.8
"5555","ana luisa",8.5
```

dni	nombre	nota
1111	pepe luis	6.5
3333	maria luisa	8.8
5555	ana luisa	8.5

Lectura de datos de ficheros Csv

```python
import logging
import os.path
import csv

def cargar_csv(nombre_fich: str) -> list[list]:
    with open(os.path.normpath(nombre_fich), "r", newline="", encoding="utf-8") as csvfile:
        csv_reader = csv.reader(csvfile, quoting=csv.QUOTE_NONNUMERIC)
        lista_filas: list[list] = list()        # Los datos del fichero CSV
        for fila in csv_reader:                 # se almacenan como LISTA DE LISTAS
            lista_filas.append(fila)            # Cada fila es una lista
    return lista_filas                          # con los valores de los campos (columnas)
                                                # La primera fila contiene los nombres de
def main() -> None:                             # los campos (columnas)
    try:
        nombre_fich = input("Introduce nombre de fichero: ")
        lista_filas = cargar_csv(nombre_fich)
    except OSError as exc:
        logging.error(f"Fichero no encontrado: [{nombre_fich}]")
    except ValueError as exc:
        logging.error(f"Codificación de fichero errónea: [{nombre_fich}]")
    else:
        print(lista_filas)

if __name__ == "__main__":
    main()
```

```
"dni","nombre","nota"
"1111","pepe luis",6.5
"3333","maria luisa",8.8
"5555","ana luisa",8.5
```

```
[['dni', 'nombre', 'nota'],
 ['1111', 'pepe luis', 6.5],
 ['3333', 'maria luisa', 8.8],
 ['5555', 'ana luisa', 8.5]]
```

Escritura de datos en ficheros Csv

- Para escribir ficheros de texto en formato **csv**, se deben realizar las siguientes acciones:
  - Se debe importar el módulo **csv**.
  - Se debe **abrir** el fichero de texto en modo escritura (`"w"`) con la función `open` dentro de la sentencia `with`, especificando el caracter de *nueva-línea* como el string vacío (`""`), y la codificación `"utf-8"`.

    ```
    with open(os.path.normpath(nombre_fich), "w", newline="", encoding="utf-8") as csvfile:
    ```

  - Dentro del contexto de la sentencia `with`, se debe crear un objeto **escritor-de-CSV** asociado al fichero que se abrió en el paso anterior, especificando que se deben *entre-comillar* todos los valores que no sean numéricos:

    ```
    csv_writer = csv.writer(csvfile, quoting=csv.QUOTE_NONNUMERIC)
    ```

    - `QUOTE_NONNUMERIC`: entre-comilla los *strings* (`str`), números (`int`, `float`) sin entre-comillar.
  - A continuación, si todas las filas a escribir ya se encuentran en una lista de listas de valores, entonces se puede utilizar el método `writerows` del objeto **escritor-de-CSV** para escribir **todas** las filas al fichero en formato *CSV*.

    ```
    csv_writer.writerows(lista_filas) # lista_filas: list[list[str|float|int]]
    ```

  - También se puede utilizar el método `writerow` del objeto **escritor-de-CSV** para escribir **cada fila** de forma individualizada, como una lista de valores (`list[str|float|int]`), al fichero en formato *CSV*.

    ```
    for lista_valores in lista_filas:
        csv_writer.writerow(lista_valores) # lista_valores: list[str|float|int]
    ```

Escritura de datos en ficheros Csv

```python
import logging
import os.path
import csv

def guardar_csv(nombre_fich: str, lista_filas: list[list]) -> None:
    with open(os.path.normpath(nombre_fich), "w", newline="", encoding="utf-8") as csvfile:
        csv_writer = csv.writer(csvfile, quoting=csv.QUOTE_NONNUMERIC)
        csv_writer.writerows(lista_filas)

def main() -> None:                                    # Los datos del fichero CSV
    try:                                               # se almacenan como LISTA DE LISTAS
        lista_filas = [["dni", "nombre", "nota"],      # Cada fila es una lista
                       ["1111", "pepe luis", 6.5],     # con los valores de los campos (columnas)
                       ["3333", "maria luisa", 8.8],   # La primera fila contiene los nombres de
                       ["5555", "ana luisa", 8.5]]     # los campos (columnas)
        nombre_fich = input("Introduce nombre de fichero: ")
        guardar_csv(nombre_fich, lista_filas)
    except OSError as exc:
        logging.error(f"No se puede crear el fichero: [{nombre_fich}]")
    except ValueError as exc:
        logging.error(f"Codificación de fichero errónea: [{nombre_fich}]")

if __name__ == "__main__":
    main()
```

```
[["dni", "nombre", "nota"],
 ["1111", "pepe luis", 6.5],
 ["3333", "maria luisa", 8.8],
 ["5555", "ana luisa", 8.5]]
```

```
"dni","nombre","nota"
"1111","pepe luis",6.5
"3333","maria luisa",8.8
"5555","ana luisa",8.5
```

Escritura de datos a Cadenas de Caracteres (*string*)

- A veces, es necesario generar una *cadena de caracteres* en un formato complejo. En estos casos, podemos crear un objeto `io.StringIO`, que permite hacer operaciones de *escritura a fichero*, cuyo resultado se puede extraer después como una cadena de caracteres.

```
import io

def dict2str(dcc: dict) -> str:
    resultado = ""
    with io.StringIO() as strio:
        for (clave, valor) in dcc.items():
            strio.write(f"{clave}: {valor}\n")
        resultado = strio.getvalue()  # dentro de WITH
    return resultado

def main() -> None:
    dcc = { "a": 1, "b": 2, "c": 3 }
    salida = dict2str(dcc)
    print(salida)

if __name__ == "__main__":
    main()
```

```
def dict2str(dcc: dict) -> str:
    return "\n".join(f"{clave}: {valor}" for (clave, valor) in dcc.items())
```