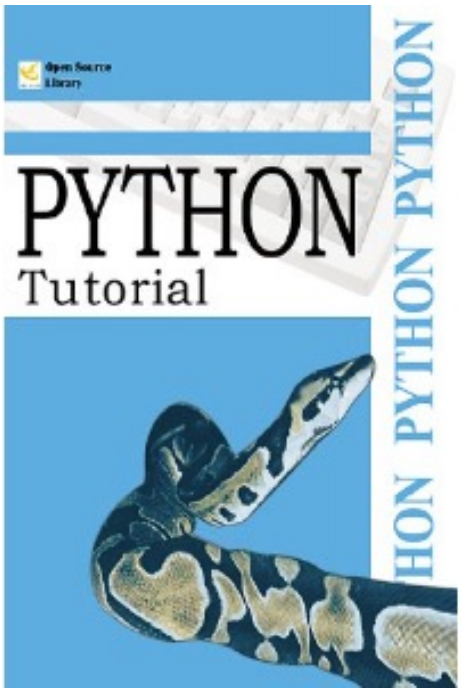
Files & BBDD



Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Understanding and using files in Python
- Understanding and using DDBB

Source

- https://docs.python.org/3/reference/
- https://ellibrodepython.com/
- Python Tutorial - Tapa blanda. GuidoVan Rossum (2012)

Outline

- **Introduction**
- Text vs binary file
- Operations
- Serialization
- Working with .json
- Working with .csv
- sqlite

Introduction

- Where did the programs get the data needed for execution?
- What happened to the data once the program finished running?
- How can we ensure data persists across multiple executions?
  - Instead of storing data in memory (e.g., dict or variables), what if we save it to a file on the disk?
  - How do we establish communication between the program and the file where data is stored?

- File Handling

Introduction: advantages

- Versatility: perform a wide range of operations: create, read, write, append, rename, and delete files
- Flexibility: supports multiple file types (e.g., text, binary, CSV, JSON), SQL
- Ease of Use: provides a user-friendly interface that simplifies file manipulation
- Cross-Platform: Windows, Mac & Linux, ensuring compatibility and integration

Introduction: disadvantages

- Error-Prone: may lead to errors if not handled carefully, especially with file permissions or file locks
- Security Risks: vulnerable to security issues if user inputs are not validated (e.g., unauthorized access or modifications).
- Complexity: advanced file formats or operations can be challenging and require meticulous attention to detail
- Performance: can be slower than other programming languages for handling large files or complex operations

Outline

- Introduction
- **Text vs binary file**
- Operations
- Serialization
- Working with .json
- Working with .csv
- sqlite

Text vs binary file

- Text file: each byte (every 8 bits) of the file corresponds to a character in the table of character codes used (ASCII, ANSI, UTF-8 etc.)

- Binary file: when the previous correspondence does not exist

Outline

- Introduction
- Text vs binary file
- **Operations**
- Serialization
- Working with .json
- Working with .csv
- sqlite

Operations Overview

- Open the File to connect your program with the file
- Perform Operations: read, write, or modify the file's content
- Close the File to release resources and ensure data integrity

Outline

- Introduction
- Text vs binary file
- Operations
  - **Open**
  - Read
  - Write
  - Random Access
  - Close
  - Rename
  - Delete
- Serialization

Open a file

- The open() function is used to open a file
- This function returns a file object, also called a ***handle***, which is used to read or modify the file

Open()

```
file_object = open(filename [, accessmode] [, buffering])
```

Open(): parameters

- filename: name of the file that we want to access
- accessmode: read, write, append, etc. (next slide)
- buffering:

  0 – No buffering

  1 – Line buffering

  Integer greater than 1 – Buffering is performed with the indicated buffer size

Access Mode

Mode	Description	Action
'r'	Read (default)	Opens the file for reading. Raises FileNotFoundError if the file does not exist
'w'	Write	Creates the file if it doesn’t exist. Overwrites the content if it exists
'a'	Append	Creates the file if it doesn’t exist. Adds content to the end if it exists
'w+'	Write and Read	Creates the file if it doesn’t exist. Overwrites the content if it exists
'r+'	Read and Write	Opens the file for reading and writing. Raises FileNotFoundError if the file doesn’t exist
'a+'	Append and Read	Creates the file if it doesn’t exist. Adds content to the end and allows reading
'b'	Binary	Opens the file in binary mode. Combine with other flags ('rb', 'wb', etc.) to set mode
't'	Text (default)	Opens the file in text mode. Combine with other flags to set mode
'x'	Exclusive Creation	Creates a file. Fails if the file already exists

Attributes

- Once a file is opened, we have a file object that contains various information related to the file

Open

```
file = open('ejemplo.txt')



print(file.closed)  # - Returns True if the file is closed, False otherwise -- FALSE

print(file.mode)    # - Returns the access mode in which the file was opened -- r

print(file.name)    # - Name of the file -- ejemplo.txt
```

Text vs binary mode

- Text Mode:
  - Handles character encoding (e.g., UTF-8)
  - Translates special characters (e.g., line endings)
  - Suitable for human-readable files
- Binary Mode:
  - Reads/Writes data byte by byte
  - No translation of characters
  - Useful for files like images, executables, etc

Text vs binary mode (I)

- Linux/UNIX: '\n' ←→Hex: 0a, Line Feed - LF).
- Windows: \n' (Hex: 0d 0a, -- Carriage Return + Line Feed (CR LF))
- Text Mode: automatically translates line endings to match the system

  Example: '\n' -> '\r\n' in Windows, '\n' remains as '\n' in Linux
- Binary Mode: the correspondence is kept (no translation)

  '\n' is written as 0a, regardless of the system

UNIX/Linux: \n ⇔ 0a (LF)
Windows: \n ⇔ 0d 0a (CR LF)

Outline

- Introduction
- Text vs binary file
- Operations
  - Open
  - **Read**
  - Write
  - Random Access
  - Close
  - Rename
  - Delete
- Serialization

Read

- A File object includes the following methods to read data from a file:
  - *read(chars)*: reads the specified number of characters starting from the current position (if none, until the EOF)
  - *readline()*: reads characters up to a newline character (\n)
  - *readlines()*: reads all lines until the end of the file and returns them as a **list**

Read(), readline, readlines

Example

```
file = open('ejemplo.txt')

print(file.read(5))

print("Una linea: ", file.readline())

print("Hasta el fina:l", file.readlines())

file.close()
```

ejemplo.txt

```
# Ejemplo.txt

Contenido primera línea

Contenido segunda línea

Contenido tercera línea

Contenido cuarta línea

Contenido quinta línea

Y final
```

Read(): Context Manager

Example

```
try:

    with open("ejemplo.txt", "r") as file:

        print(file.read(3))

        print("Una linea: ", file.readline())

        print("Otra linea: ", file.readline())

        print("Hasta el fina:l", file.readlines())

except FileNotFoundError:

    print("The file does not exist.")
```

Using iteration: *for*

Example

```
with open("ejemplo.txt", "r") as f:

    for line in f:

        print("Lineas", line, end=" ")
```

Output

```
# Ejemplo.txt

Contenido primera línea

Contenido segunda línea

Contenido tercera línea

Contenido cuarta línea

Contenido quinta línea

Y final
```

Specify path

Example: MAC

```
with open('/Users/mariadr-moreno/Documents/mifichero.txt', 'wt') as f:

    ...
```

Example: Windows (please check if correct)

```
with open ('C:\\myfile.txt','wt') as f:

    ...
```

With: advantages

- Automatically closes the file after the block of code
- Ensures the file is closed even if an exception occurs

Outline

- Introduction
- Text vs binary file
- Operations
  - Open
  - Read
  - **Write**
  - Random Access
  - Close
  - Rename
  - Delete

ChatGPT 4o

Co

Writing a File That Does Not Exist

- **Problem**: Using `'w'` mode to open a file that does not exist causes an error.
- **Solution**: Use `'x'` mode to create the file instead of `'w'`.

python

Copiar código

```python
# Using 'wt' mode (Error if the file doesn't exist)
with open('filename', 'wt') as f:
    f.write('Hello, This is sample content.\n')
# This will create an error that the file 'filename' doesn't exist.

# Using 'xt' mode (Creates the file)
with open('filename.txt', 'xt') as f:
    f.write('Hello, This is sample content.\n')
```

Envía un mensaje a ChatGPT

ChatGPT 4o

Co

Writing a File That Does Not Exist

- **Problem**: Using `'w'` mode to open a file that does not exist causes an error.
- **Solution**: Use `'x'` mode to create the file instead of `'w'`.

python

Copiar código

```python
# Using 'wt' mode (Error if the file doesn't exist)
with open('filename', 'wt') as f:
    f.write('Hello, This is sample content.\n')
# This will create an error that the file 'filename' doesn't exist.

# Using 'xt' mode (Creates the file)
with open('filename.txt', 'xt') as f:
    f.write('Hello, This is sample content.\n')
```

Envía un mensaje a ChatGPT

Write

- A File object includes the following methods to write data to a file:
  - *write(str)*: writes a string
  - writelines(list_of_str): write a list of strings

Write

Write

```
# Overwrites if the file exists, or creates a new one

file = open('example.txt', "w")

file.write("Machaco el fichero si tuviera algo.\n")

file.write("Otra linea.\n")

file.writelines(['Line 1\n', 'Line 2\n’])

file.close()
```

Write: *with*

Write

```
# Overwrites if the file exists, or creates a new one

with open("example_w.txt", "w") as file:

    file.write("This file is created or overwritten.")
```

Append

```
# Appends to the file if it exists, or creates a new one

with open("example_a.txt", "a") as file:

    file.write("Adding content to the end.\n")
```

Outline

- Introduction
- Text vs binary file
- Operations
  - Open
  - Read
  - Write
  - **Random Access**
  - Close
  - Rename
  - Delete

Random Access

- Allows moving to a specific position in a file without reading it sequentially
- Useful for large files where only specific parts need to be processed

seek()

- Returns the current position of the file pointer in bytes

Seek()

```
seek(offset, whence)
```

- Parameters:
  - offset: Number of bytes to move
  - whence: Reference position (default is 0)
    - 0: Beginning of the file
      1: Current position
      2: End of the file

seek(): example

Seek()

```
with open('example.txt', 'r') as file:

    file.seek(10)  # Move 10 bytes from the start

    print(file.read(5))  # Read the next 5 bytes
```

tell()

- Returns the current position of the file pointer in bytes

Tell(

```
Tell()
```

tell(): example

Tell()

```
with open('example.txt', 'r') as file:

    print(file.tell())  # Initial position

    file.seek(10)

    print(file.tell())  # Position after seeking
```

Outline

- Introduction
- Text vs binary file
- Operations
  - Open
  - Read
  - Write
  - Random Access
  - **Close**
  - Rename
  - Delete

Close()

- The close() function is used to close a file

Close

```
file_object.close()
```

Example

```
f = open("bin.txt", "wb")

print("Name of the file:", f.name)

f.close()
```

Outline

- Introduction
- Text vs binary file
- Operations
  - Open
  - Read
  - Write
  - Random Access
  - Close
  - **Rename**
  - Delete

Rename

- The *os* module provides methods that help to perform file-processing operations, such as renaming and deleting

Rename

```
os.rename(current_file_name, new_file_name)
```

Example

```
import os

os.rename('example.txt', 'NewExample.txt')

print('File renamed')
```

Outline

- Introduction
- Text vs binary file
- Operations
  - Open
  - Read
  - Write
  - Random Access
  - Close
  - Rename
  - **Delete**

Delete

- Use the os.remove() method to delete files

Rename

```
os.remove(current_file_name)
```

Example

```
import os

os.remove('example.txt')

print('File renamed')
```

Outline

- Introduction
- Text vs binary file
- Operations
- **Serialization**
- Working with .json
- Working with .csv
- sqlite

Serialization

- Is the process of converting a Python object into a format that can be:
  - Stored in a file or database
  - Transferred across a network
  - Reconstructed later into the original object (deserialization)
- Why?
  - Data Persistence: save objects to a file for future use (e.g., saving application state)
  - Data Exchange: share data between systems or applications
  - Caching: store pre-processed data to reduce computation time
  - Inter-Process Communication: pass objects between different parts of a program or systems

Serialization in Python

- Python provides several modules for serialization:
  - pickle: Python-specific, handles any Python object
  - json: human-readable, compatible with other languages
  - yaml: used for configuration files, more readable than JSON

Pickle module

- Pickling: the process of converting a Python object into a byte stream
  - pickle.dump(obj, file): serializes obj and saves it to a file
  - pickle.dumps(obj): serializes obj and returns it as a string

- Unpickling: the process of converting a byte stream back into a Python object
  - pickle.load(file): reads a file to deserialize its content into an object
  - pickle.loads(bytes_object): deserializes a string into an object

Example: List of objects

```
import pickle

class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

    def __str__(self):

        return f"Person(name={self.name}, age={self.age})"

# list of objets

people = [Person("Alice", 30), Person("Bob", 25), Person("Charlie", 35)]
```

Example: continue…

```
# Save in a file

with open("people.pkl", "wb") as file:

    pickle.dump(people, file)  # Serializa la lista de objetos y la guarda

# Read from a file

with open("people.pkl", "rb") as file:

    loaded_people = pickle.load(file)  # Carga los objetos serializados

    print(loaded_people)
```

Example I: Objects

```
import pickle

class Book:

    def __init__(self, title, author, year):

        self.title = title

        self.author = author

        self.year = year

books = [

    Book("The Great Gatsby", "F. Scott Fitzgerald", 1925),

    Book("To Kill a Mockingbird", "Harper Lee", 1960),

    Book("1984", "George Orwell", 1949)

]
```

Example I: continue

```
with open("books.pkl", "wb") as file:

    pickle.dump(books, file)

with open("books.pkl", "rb") as file:

    loaded_books = pickle.load(file)



# Print the books loaded

for book in loaded_books:

    print(f"Title: {book.title}, Author: {book.author}, Year: {book.year}")
```

Outline

- Introduction
- Text vs binary file
- Operations
- Serialization
- **Working with .json**
- Working with .csv
- sqlite

.json

- JSON (JavaScript Object Notation) is a lightweight data interchange format
- It is easy for:
  - Humans to read and write
  - Machines to parse and generate
- Commonly used for transmitting data between a server and a web application, as an alternative to XML
- As a summary:
  - Lightweight: minimal and efficient format for data exchange
  - Human-Readable: simple syntax, easy to understand
  - Language-Independent: supported by most programming languages, including Python

Summary

Feature	JSON	XML
Syntax	Simple and concise	Verbose
Readability	Easier to read/write	More complex
Data Interchange	Lightweight, efficient	Heavier, more formal
Use Cases	Modern web APIs, apps	Legacy systems, docs

.json data structures

1. Objects:
   - Unordered collections of key-value pairs
   - Enclosed in curly braces {}
   - Keys must be strings, and values can be any valid JSON data type

Example: objects

```
{
    "name": "Alice",
    "age": 30,
    "is_student": false
}
```

.json data structures

2. Arrays:
   - Ordered collections of values,
   - Enclosed in square brackets []

Example: arrays

```
["apple", "banana", "cherry"]
```

.json data structures

3. Values:
   - Numbers: 42, 3.14
   - Strings: "hello"
   - Booleans: true, false
   - null
   - Objects or Arrays

Example: JSON format

```
{
    "id": 101,
    "name": "John Doe",
    "skills": ["Python", "JavaScript", "SQL"],
    "active": true,
    "address": {
        "street": "123 Main St",
        "city": "Anytown",
        "zip": "12345"
    }
}
```

JSON module

- Serialization with JSON
  - json.dump(obj, file, [indent]): writes Python object as JSON to a file
  - json.dumps(obj, [indent]): converts Python object into JSON string
- Deserialization with JSON
  - json.load(file): reads JSON from a file and converts it into a Python object
  - json.loads(json_string): reads JSON string and deserializes into a Python object

Example: dump

```
import json

data = {

    "name": "Alice",

    "age": 30,

    "skills": ["Python", "Machine Learning",

               "Data Analysis"],

    "is_student": False

}

# Save JSON to a file

with open("data.json", "w") as file:

    json.dump(data, file, indent=4)
```

Ouput: file

```
{

    "name": "Alice",

    "age": 30,

    "skills": [

        "Python",

        "Machine Learning",

        "Data Analysis"

    ],

    "is_student": false

}
```

Example: load

```
import json

with open("data.json", "r") as file:

    existing_data = json.load(file)

print(existing_data)
```

Ouput:

```
{'name': 'Alice', 'age': 30, 'skills': ['Python', 'Machine Learning', 'Data Analysis'],
'is_student': False}
```

Exercise

- Añadir al fichero json más campos.
- Para ello, debe guardarlos en una LISTA
- Añade 3 veces data y 3 veces new_data

Example: new data

```
new_data = {
    "name": "Bob",
    "age": 25,
    "skills": ["Java", "C++"],
    "is_student": True
}
```

```
[
    {
        "name": "Alice",
        "age": 30,
        "skills": [
            "Python",
            "Machine Learning",
            "Data Analysis"
        ],
        "is_student": false
    },
    {
        "name": "Bob",
        "age": 25,
        "skills": [
            "Java",
            "C++"
        ],
        "is_student": true
    },
    {
        "name": "Bob",
        "age": 25,
        "skills": [
            "Java",
            "C++"
        ],
        "is_student": true
    },
    {
        "name": "Bob",
```

Outline

- Introduction
- Text vs binary file
- Operations
- Serialization
- Working with .json
- **Working with .csv**
- sqlite

.csv

- CSV (Comma Separated Values) is a commonly used data format used by spreadsheets
- A plain text file used to store tabular data (rows and columns).
- Each line is a row, and values are separated by a delimiter (commonly a comma)
- Why use CSV?
  - Portable and widely used for data exchange
  - Easy to read and write using Python

CSV module

- Write:
  - writer(): writes plain text to the file
  - writerow(): writes a single row as a list
  - writerows(): writes multiple rows at once (from a list of lists)
  - DictWriter(): writes dictionaries to a CSV file
- Read:
  - reader(): reads the file and returns rows as lists
  - DictReader(): reads the file and returns rows as dictionaries
  - next(): used to skip the header row in CSV files, allowing you to process only the data rows

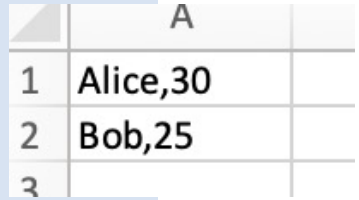
Example: writerows()

```
import csv

with open('file.csv', 'w', newline='') as file:

    writer = csv.writer(file)

    writer.writerows([['Alice', 30], ['Bob', 25]])
```



Example: DictWriter()

```
with open('output.csv', 'w', newline='') as file:

    fieldnames = ["Name", "Age", "Country"] #A header

    writer = csv.DictWriter(file, fieldnames=fieldnames)

    writer.writeheader()

    writer.writerow({"Name": "Alice", "Age": 30, "Country": "USA"})

    writer.writerow({"Name": "Bob", "Age": 25, "Country": "UK"})
```

Example: read()

```
import csv

with open('file.csv', 'r') as file:

    reader = csv.reader(file)

    for row in reader:

        print(row)
```

Example: DictReader ()

```
import csv

with open('output.csv', 'r') as file:

    reader = csv.DictReader(file)

    for row in reader:

        print(row)  # Each row is a dictionary

        print(row['age'] #for a specific column
```

pandas library

- It is a Python library designed for data manipulation and analysis
- Provides data structures like:
  - DataFrame: A 2D labeled, tabular structure (like an Excel sheet)
  - Series: A 1D labeled array (like a column in a table)
- Why use pandas?
  - Simplifies data operations (cleaning, filtering, aggregations)
  - Supports integration with multiple data formats (CSV, Excel, SQL, JSON)
  - Highly efficient for handling large datasets

Comparison

Feature	pandas	Csv module
Ease of Use	High	Moderate
Handles Large Files	Yes (chunking)	Limited
Data Filtering	Built-in	Manual implementation
Support missing data	Yes	No

pandas module for csv files

- pip install pandas

Example: pandas

```
import pandas as pd

# Read data from CSV

df = pd.read_csv('data.csv')

# Process data (e.g., filter rows where Age > 30)

filtered_df = df[df['Age'] > 30]

# Write the processed data to a new CSV

filtered_df.to_csv('filtered_data.csv', index=False)
```

Outline

- Introduction
- Text vs binary file
- Operations
- Serialization
- Working with .json
- Working with .csv
- **sqlite**

SQLite

- A lightweight, file-based database system
- Relational DDBB: we can query the data in **binary** representation
- Self-contained, serverless, and highly portable
- Suitable for small to medium-sized applications
- 4 operations for persistent storage: Create, Read, Update, Delete (CRUD)
- Why Use SQLite in Python?
  - Built-in support via the sqlite3 module
  - No setup or installation required
  - Ideal for prototyping and development

Create DDBB from CSV (interactive)

- Acceder al terminal: sqlite3
- .mode csv
- .import 'file_name' name_table
- .schema: to see the table
- SELECT row FROM name_table;
- .save file.db: save the table (in binary) into a file
- .quit: to leave the environment

Create DDBB from CSV

```
sqlite> .quit
(base) MacBook-Pro-de-Maria-2:Files mariadr-moreno$ sqlite3
SQLite version 3.32.3 2020-06-18 14:16:19
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .mode csv
sqlite> .import 'output.csv' prueba
sqlite> SELECT Name FROM prueba;
Alice
Bob
sqlite> .save prueba.db
sqlite> .quit
```

Create a DDBB with a program

1. Database Connection
2. Creating Tables
3. Inserting Data
4. Reading Data
5. Updating Data
6. Deleting Data
7. Closing the Connection

Connection

- Import the *sqlite3* module
- Connect to a database file (creates one if it doesn't exist)

Example: connection

```
import sqlite3

connection = sqlite3.connect(prueba.db')
```

Create Table

- Create a cursor object using *connection.cursor()*
- Use *execute()* method to run SQL commands (add IF NOT EXISTS to prevent error if the table exits)
- Then, commit changes in the DDBB using *connection.commit()*

Example: create table

```
cursor = connection.cursor()

cursor.execute('''

    CREATE TABLE IF NOT EXISTS prueba (

        Name TEXT,

        Age TEXT,

        Country TEXT

    )

''')

connection.commit()
```

Insert Data

- Use INSERT INTO to add rows to your table

Example: insert data

```
row = [("Charlie", "32", "Canada")]

cursor.execute('''

            INSERT INTO prueba (Name, Age, Country)

            VALUES (?, ?, ?)

        ''', (row['Name'], row['Age'], row['Country']))
```

Reading Data

- Use *fetchall()* or *fetchone()* to retrieve query results

Example: read data

```
cursor.execute('SELECT * FROM prueba')

rows = cursor.fetchall()

for row in rows:

    print(row)
```

Update data

- Modify existing records with UPDATE

Example: update

```
cursor.execute('''
    UPDATE prueba
    SET Name = ?
    WHERE Name = ?
''', ('Alicia', 'Alice'))
connection.commit()
```

Delete data

- Remove records with DELETE

Example: delete

```
cursor.execute('''

    DELETE FROM prueba

    WHERE Name = ?

''', ('Bob',))

connection.commit()
```

Closing connection

- Always close the connection when done to free resources

Example: close

```
connection.close()
```

- Visualize your DDBB: *https://sqlitebrowser.org/*