Serialización

La serialización es el proceso de convertir un objeto o una estructura de datos en un formato que pueda ser fácilmente almacenado o transmitido y luego reconstruido en su forma original. Este proceso consiste en transformar la representación interna de los datos en una secuencia de bytes o en un formato de texto que pueda ser almacenado en un archivo, enviado a través de una red o guardado en una base de datos. La deserialización es el proceso inverso, donde la secuencia de bytes o el formato de texto se convierte de nuevo en el objeto o estructura de datos original.

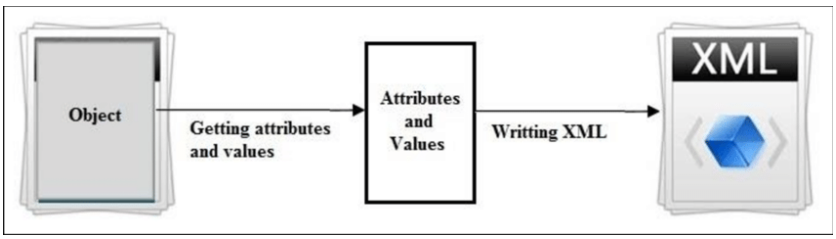
Las características más relevantes de la serialización son:

- **Compatibilidad:** Permite que datos estructurados sean leídos y escritos en diferentes lenguajes de programación.
- **Persistencia:** Permite guardar el estado de un objeto en un archivo o base de datos para ser recuperado más tarde.
- **Portabilidad:** Facilita el envío de datos entre diferentes sistemas y plataformas.
- **Eficiencia:** Optimiza el proceso de almacenamiento y transmisión de datos complejos.
- **Flexibilidad:** Soporta varios formatos de serialización, como binario y texto.

Las aplicaciones más comunes de la serialización se presentan a continuación:

- **Almacenamiento de datos:** Guardar el estado de objetos en bases de datos o archivos para su posterior recuperación.
- **Comunicación entre procesos:** Enviar datos entre procesos en la misma máquina o en diferentes máquinas.
- **Transmisión de datos a través de redes:** Enviar datos estructurados entre un cliente y un servidor en aplicaciones distribuidas.
- **Interoperabilidad entre lenguajes:** Facilitar el intercambio de datos entre programas escritos en diferentes lenguajes de programación.
- **Caching:** Almacenar objetos en memoria caché para mejorar el rendimiento de las aplicaciones.

El flujo de la serialización y deserialización comienza con un objeto o estructura de datos en su estado original. En el proceso de serialización, este objeto es transformado en un formato que puede ser fácilmente almacenado o transmitido, como una cadena JSON, un archivo binario o XML. Esta transformación implica convertir la representación interna del objeto en una secuencia de bytes o una cadena de texto, lo que permite su persistencia en medios de almacenamiento como archivos o bases de datos, o su envío a través de redes. Una vez que los datos serializados llegan a su destino o necesitan ser utilizados nuevamente, se lleva a cabo la deserialización. Durante la deserialización, la secuencia de bytes o cadena de texto es convertida de nuevo a su representación original en forma de objeto o estructura de datos, recuperando así toda la información y estado del objeto inicial.



Desarrollo:

Inicialmente creamos un proyecto en Visual Studio Code, basta con crear una carpeta y dos scripts con extensión .py para esta práctica se nombraron de la siguiente manera:

- P_serial_cliente.js
- P_serial_servidor.js

Ahora se buscará en la presente práctica enviar el contenido de un documento de tipo PDF al servidor con el apoyo de la serialización siguiendo un proceso de serializado y que el servidor se va a encargar de procesar ese contenido y deserializarlo para agregarlo en un documento PDF nuevo.

Para esta práctica se utilizará Flask como microframework para la facilidad de enviar peticiones de tipo POST y request es una biblioteca en Python que tiene como funcionalidad es facilitar el envío de peticiones.

Inicialmente se comenzará por instalar flask, para ello se debe de agregar lo siguiente en la terminal:

```
C:\Windows\system32>pip install flask
Collecting flask
  Downloading flask-3.0.3-py3-none-any.whl.metadata (3.2 kB)
Collecting Werkzeug>=3.0.0 (from flask)
  Downloading werkzeug-3.0.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading jinja2-3.1.4-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.1.2 (from flask)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from flask)
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Downloading blinker-1.8.2-py3-none-any.whl.metadata (1.6 kB)
Collecting colorama (from click>=8.1.3->flask)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Downloading MarkupSafe-2.1.5-cp312-cp312-win_amd64.whl.metadata (3.1 kB)
Downloading flask-3.0.3-py3-none-any.whl (101 kB)
   ---------------------------------------- 101.7/101.7 kB 983.8 kB/s eta 0:00:00
Downloading blinker-1.8.2-py3-none-any.whl (9.5 kB)
Downloading click-8.1.7-py3-none-any.whl (97 kB)
   ---------------------------------------- 97.9/97.9 kB 73.8 kB/s eta 0:00:00
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading jinja2-3.1.4-py3-none-any.whl (133 kB)
```

Posteriormente para instalar requests se hace de una forma similar como se muestra a continuación:

```
C:\Windows\system32>pip install requests
Collecting requests
  Obtaining dependency information for requests from https://files.pythonhosted.org/packages/f9/9b/335f9764261e915ed497fcdeb11df5dfd6f7bf257d4a6a2a686d80da4d54/requests-2.32.3-py3-none-any.whl.metadata
  Using cached requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\python312\lib\site-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\python312\lib\site-packages (from requests) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\python312\lib\site-packages (from requests) (2.2.2)
Collecting certifi>=2017.4.17 (from requests)
  Obtaining dependency information for certifi>=2017.4.17 from https://files.pythonhosted.org/packages/1c/d5/c84e1a17bf61d4df64ca866a1c9a913874b4e9bdc131ec689a0ad013fb36/certifi-2024.7.4-py3-none-any.whl.metadata
  Using cached certifi-2024.7.4-py3-none-any.whl.metadata (2.2 kB)
Using cached requests-2.32.3-py3-none-any.whl (64 kB)
Using cached certifi-2024.7.4-py3-none-any.whl (162 kB)
Installing collected packages: certifi, requests
Successfully installed certifi-2024.7.4 requests-2.32.3

[notice] A new release of pip is available: 23.2.1 -> 24.1.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Una vez que se haya instalado el microframework y la biblioteca de Python se comenzará con el desarrollo de la práctica primero agregando la lógica necesaria para nuestro cliente que será el encargado de mandar el documento PDF, se agrega el siguiente fragmento de código al archivo P_serial_cliente.py:

```
import pickle
import requests

# Leer el archivo PDF
with open('documento.pdf', 'rb') as f:
    pdf_data = f.read()

# Serializar el contenido del PDF
serialized_data = pickle.dumps(pdf_data)

# Enviar el contenido serializado al servidor
response = requests.post('http://localhost:8080/prueba_serializacion',
files={'file': ('documento.pdf', serialized_data)})
print(response.text)
```

Este fragmento de código lee un archivo PDF, serializa su contenido utilizando la biblioteca pickle, y envía los datos serializados a un servidor mediante una solicitud POST usando la biblioteca requests. La biblioteca pickle permite convertir objetos de Python en una secuencia de bytes para su almacenamiento o transmisión, mientras que requests es una biblioteca popular para hacer solicitudes HTTP en Python de manera sencilla.

Ahora procedemos a definir la lógica que se utilizará en el servidor, se agregará el siguiente fragmento de código en el archivo P_serial_servidor.py como se muestra a continuación:

```
from flask import Flask, request
import pickle
import json

app = Flask(__name__)

@app.route('/prueba_serializacion', methods=['POST'])
def upload():
    # Obtener el archivo serializado
    serialized_data = request.files['file'].read()
    print(serialized_data)

    # Deserializar el contenido
    pdf_data = pickle.loads(serialized_data)

    # Guardar el archivo PDF
    with open('received_document.pdf', 'wb') as f:
        f.write(pdf_data)

    # Imprimir los headers
    print("Headers: ", request.headers)

    # Crear un diccionario con la información relevante del request
    request_info = {
```

```
        'headers': dict(request.headers),
        'form': request.form.to_dict(),
        'files': {file_name: file.filename for file_name, file in
request.files.items()}
    }

    # Imprimir el mensaje en formato JSON
    print("Request Info (JSON):", json.dumps(request_info, indent=4))

    return 'Archivo recibido y guardado', 200

if __name__ == '__main__':
    app.run(port=8080)
```

Este código define un servidor web usando Flask que maneja solicitudes POST en la ruta /prueba_serializacion. Cuando recibe una solicitud, lee un archivo serializado enviado en el campo file, deserializa su contenido con pickle.loads para obtener los datos originales del PDF y guarda estos datos en un archivo llamado received_document.pdf. Adicionalmente, imprime los encabezados de la solicitud y otra información relevante del request en formato JSON para propósitos de depuración. Finalmente, responde con un mensaje indicando que el archivo ha sido recibido y guardado.

Una vez comprendido el funcionamiento de la lógica del lado del cliente y del lado del servidor se va a ejecutar la práctica y ver su funcionamiento:

Primero se tiene el siguiente contenido dentro de un documento de tipo PDF:

"La serialización es el proceso de convertir un objeto en un formato que se puede almacenar o transmitir y luego reconstruirlo en su forma original. En otras palabras, se trata de transformar una estructura de datos o un objeto en una secuencia de bytes que se puede guardar en un archivo, enviar a través de una red, o almacenar en una base de datos."

Entonces una vez teniendo dicho documento se procede a ejecutar nuestro servidor como se muestra a continuación:

```
                                                                    python
.\P_serial_servidor.py
 * Serving Flask app 'P_serial_servidor'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production
 WSGI server instead.
 * Running on http://127.0.0.1:8080
Press CTRL+C to quit
```

Mientras está en ejecución el servidor se va a ejecutar el cliente como se muestra a continuación:

```
 python .\P_serial_cliente.py
Archivo recibido y guardado
```

Una vez que se haya completado la ejecución del lado del cliente el servidor va a recibir una respuesta como la que se ve a continuación en la terminal y en Wireshark:

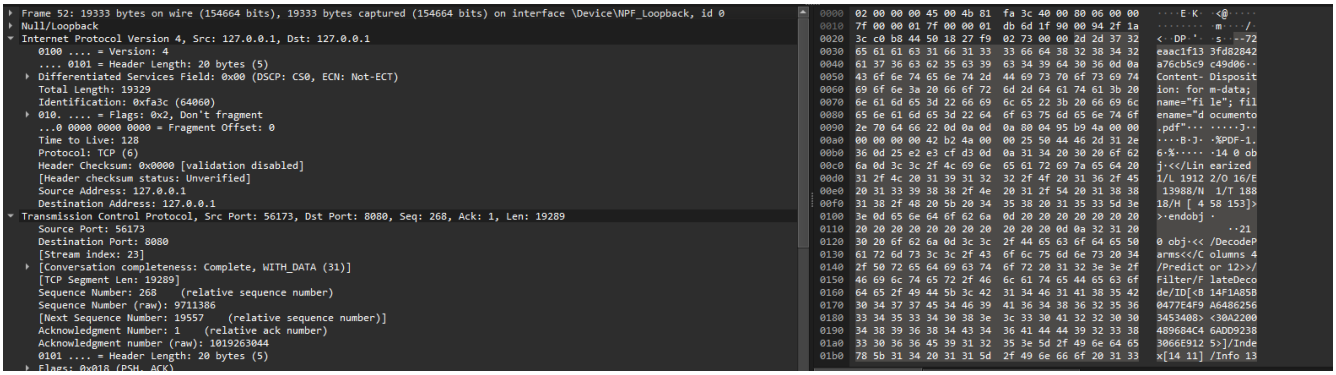
```
    \n
        \n
            \n                                  \n<?xpacket end="w"?>\r\nendstream\rendobj\r3
<</Filter/FlateDecode/First 5/Length 50/N 1/Type/ObjStm>>stream\r\nh\xde24R0P\xb0\xb1\xd1
cd+Q0\xd4\xf7\xceL)\x8e64\x03\n\x06\xc5\xea\x87T\x16\xa4\xea\x07$\xa6\xa7\x16\xdb\xd9\x01
\x00\xe70\x0b\xdc\r\nendstream\rendobj\r4 0 obj\r<</Filter/FlateDecode/First 5/Length 206
e/ObjStm>>stream\r\nh\xde|\xcdAK\xc3@\x10\x86\xe1\xbf2\xb7&\x07\x9b\xc96\xa4\xa5\x94B1xP\
\x9e\'\xd9\x11W\xdb\x9d2\xbb\x8b\xc4_\xefF<x\xf2\xf6\x1e>\x9e\xaf^\x01\xc2nW\x1dR|\x13-\x
\x80{\x0e)\xc0\x89|\x94\x00={\xcb_eu+\x97\x0b\xfb\x18\x8a\x9f\xbc\x92\x9f\xe6R\xa6\xe8\xc
b9\xe8\xb6\x06M\x83k\xb3\xa9[\xc4\xa6\xbe\xc1v\x81\xb8\xf8]e\xfe0\xaa\x0c\x14\xe1\xd8\xdd
\xc1\n\x06\xe1JJ\xf0"j\xcb\xea\x81\xa7\xcf\x1c\xf3C/\xf6\x7f\xf2\xa8b\xd3\xc8\xd9\xb42\xf
a3\x1b\x94t\xca\xe8\xb2]\xae\x9b\xb2:I\xd2\x913\xe5^\x1d\xdb?\x961\x88\xabM\x1e\xa4\xe1\x
98\xff\x9e\\<sQ\xee\xf7\xdf\x02\x0c\x00\t\xa2Q\xf3\r\nendstream\rendobj\r5 0 obj\r<</Deco
</Columns 4/Predictor 12>>/Filter/FlateDecode/ID[<B14F1A85B0477E4F9A64862563453408><30A22
C46ADD92383066E9125>]/Info 13 0 R/Length 49/Root 15 0 R/Size 14/Type/XRef/W[1 2 1]>>strea
debb\x00\x02&F\xb3%\x0cL\x0cL\xd5@\x82\x7f5\x90`\xec\x03\x11\xda@\x89\xedu@\x16\x03\x03#\
xe97\x90`d\x00\x080\x00\xd0\xad\x05#\r\nendstream\rendobj\rstartxref\r\n116\r\n%%EOF\r\n\
Headers:  Host: localhost:8080
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 19289
Content-Type: multipart/form-data; boundary=72eaac1f133fd82842a76cb5c9c49d06


Request Info (JSON): {
    "headers": {
        "Host": "localhost:8080",
        "User-Agent": "python-requests/2.32.3",
        "Accept-Encoding": "gzip, deflate",
        "Accept": "*/*",
        "Connection": "keep-alive",
        "Content-Length": "19289",
        "Content-Type": "multipart/form-data; boundary=72eaac1f133fd82842a76cb5c9c49d06"
    },
    "form": {},
    "files": {
        "file": "documento.pdf"
    }
}
127.0.0.1 - -                    "POST /prueba_serializacion HTTP/1.1" 200 -
```

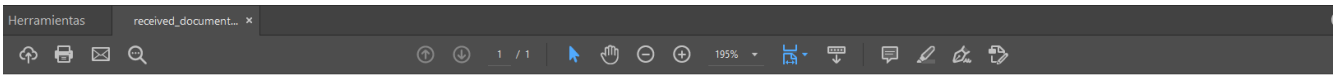
En Wireshark se puede observar que se tiene la respuesta del endpoint:

```
HTTP    19333 POST /prueba_serializacion HTTP/1.1
TCP        44 8080 → 56173 [ACK] Seq=1 Ack=19557 Win=2600448 Len=0
TCP       217 8080 → 56173 [PSH, ACK] Seq=1 Ack=19557 Win=2600448 Len=173 [TCP segment of a reassembled PDU]
TCP        44 56173 → 8080 [ACK] Seq=19557 Ack=174 Win=2619392 Len=0
HTTP       71 HTTP/1.1 200 OK  (text/html)
```

También se obtiene información sobre el paquete enviado:



Finalmente se obtiene un archivo de tipo PDF como se muestra a continuación:



La serialización es el proceso de convertir un objeto en un formato que se puede almacenar o transmitir y luego reconstruirlo en su forma original. En otras palabras, se trata de transformar una estructura de datos o un objeto en una secuencia de bytes que se puede guardar en un archivo, enviar a través de una red, o almacenar en una base de datos.