

Taller 2 - Complejidad Espacial

Hugo Alejandro Latorre Portela 506222722

Juan Esteban Pinzon Preciado 506222727

Amir Rodriguez Mejia 506222032

Fundación Universitaria Konrad Lorenz

I. INTRODUCCIÓN

Este proyecto realizó un análisis comparativo de dos scripts, uno en Python y otro en Java, para evaluar su rendimiento en tres máquinas diferentes así como su complejidad temporal y espacial. El objetivo principal es medir y comparar el tiempo y la memoria utilizados por los dos scripts en múltiples ejecuciones en diferentes entornos de hardware. Además, se calcula un factor de aceleración, que muestra cuánto más rápido funciona la versión mejorada en comparación con la versión anterior en un entorno determinado.

II. REQUISITOS DEL PROYECTO

Para llevar a cabo este proyecto, se realizaron las siguientes acciones:

Clonación del repositorio que contiene los proyectos en Python y Java. Instalación de las dependencias requeridas para el proyecto en Python utilizando el comando pip install -r requirements.txt. Ejecución del script en Python para generar datos simulados en formato CSV, generando un archivo con 500,000 líneas. Ejecución del proyecto en Java para procesar el archivo CSV generado por el script de Python. Medición de la complejidad espacial (uso de memoria) de ambos scripts en cada una de las tres máquinas. Cálculo de la complejidad temporal (tiempo de ejecución) de ambos scripts, considerando factores como bucles y llamadas a funciones. Cálculo del speedup para los dos mejores resultados obtenidos en diferentes máquinas.

III. CÓDIGOS USADOS EN LA PRÁCTICA

III-A. Código de python

Análisis: este código es un ejemplo de cómo se pueden utilizar bibliotecas de Python para generar datos ficticios y almacenarlos en un archivo CSV. Puede ser útil en situaciones donde se requiera un conjunto de datos de prueba o demostración para propósitos de desarrollo o análisis.

```

1 import csv
2 from Faker import Faker
3 import random
4 import time
5 import psutil
6
7 fake = Faker()
8
9     def generate_data():
10         data = []
11         for _ in range(500 * 2):
12             username = fake.user_name()
13             birthdate = fake.date_of_birth(
14                 minimum_age=18,
15                 maximum_age=70).strftime('%Y-%m-%d')
16             income = round(random.uniform(1000, 10000), 2)
17             debt = round(random.uniform(0, 5000), 2)
18             sex = random.choice(['Male', 'Female'])
19             num_children = random.randint(0, 5)
20             country = fake.country()
21             data.append([
22                 username,
23                 birthdate,
24                 income,
25                 debt,
26                 sex,
27             ])
28
29     return data
30
31
32     def save_to_csv(data, filename):
33         with open(filename, mode='w', newline='') as file:
34             writer = csv.writer(file)
35             writer.writerow([
36                 'Username',
37                 'Birthdate',
38                 'Income',
39                 'Debt',
40                 'Sex',
41                 'Number of Children',
42                 'Country'])
43             writer.writerows(data)
44
45     save_to_csv(data, 'dummy_data.csv')

```

Figura 1. Código Python primera parte

```

18     sex = random.choice(['Male', 'Female'])
19     num_children = random.randint(0, 5)
20     country = fake.country()
21     data.append([
22         username,
23         birthdate,
24         income,
25         debt,
26         sex,
27         num_children,
28         country])
29
30     return data
31
32     def save_to_csv(data, filename):
33         with open(filename, mode='w', newline='') as file:
34             writer = csv.writer(file)
35             writer.writerow([
36                 'Username',
37                 'Birthdate',
38                 'Income',
39                 'Debt',
40                 'Sex',
41                 'Number of Children',
42                 'Country'])
43             writer.writerows(data)
44
45     save_to_csv(data, 'dummy_data.csv')

```

Figura 2. Código Python segunda parte

```

46 if __name__ == "__main__":
47     start_time = time.time()
48     generated_data = generate_data()
49     save_to_csv(generated_data, filename='dummy_data.csv')
50     print("Data generation and CSV creation complete.")
51     end_time = time.time()
52     tiempo_resultado = end_time - start_time
53     print("Time taken:", tiempo_resultado)
54
55     memory_usage = psutil.virtual_memory().percent
56     print("Memory usage:", memory_usage)

```

Figura 3. Código Python tercera parte

III-B. Código de java

Análisis: En este código, el término "datos transformados" se refiere a información previamente extraída de un archivo CSV de entrada llamado "*dummydata.csv*" que ha sido sometido a una serie de procesos. Estos procesos incluyen calcular la edad de una persona a partir de su fecha de nacimiento y determinar la cantidad de dinero disponible después de pagar las deudas, lo que se conoce como "Ingresos Menos Deuda". Estos valores, junto con los datos originales, se escriben en un nuevo archivo llamado "*processeddata.csv*". Esta transformación de datos proporciona una visión más detallada y procesada de la información financiera de las personas, facilitando el análisis y la comprensión del conjunto de datos resultante.

```
① Main.java <
②
③ import java.io.*;
④ import java.text.ParseException;
⑤ import java.text.SimpleDateFormat;
⑥ import java.util.Date;
⑦
⑧ public class Main {
⑨     public static void main(String[] args) {
⑩         try {
⑪             BufferedReader reader = new BufferedReader(new FileReader("dummy_data.csv"));
⑫             BufferedWriter writer = new BufferedWriter(new FileWriter("processed_data.csv"));
⑬
⑭             String line = reader.readLine();
⑮             String[] headers = line.split(regex: ",");
⑯             writer.write("username,Birthdate,Age,Income,Debt,IncomeMinusDebt");
⑰
⑱             SimpleDateFormat dateFormat = new SimpleDateFormat(pattern: "yyyy-mm-dd");
⑲             Date currentdate = new Date();
⑳
⑳             while ((line = reader.readLine()) != null) {
⑳                 String[] values = line.split(regex: ",");
⑳                 String username = values[0];
⑳                 Date birthdate = dateFormat.parse(values[1]);
⑳                 double income = Double.parseDouble(values[2]);
⑳                 double debt = Double.parseDouble(values[3]);
⑳
⑳                 long ageInMillis = currentdate.getTime() - birthdate.getTime();
⑳                 int age = (int) (ageInMillis / (1000 * 60 * 60 * 24 * 365.25));
⑳
⑳             }
⑳         }
⑳     }
⑳ }
```

Figura 4. Código Java primera parte

```
    String[] values = line.split("\\s+");  
    String username = values[0];  
    Date birthdate = dateFormat.parse(values[1]);  
    double income = Double.parseDouble(values[2]);  
    double debt = Double.parseDouble(values[3]);  
  
    long ageInMillis = currentDate.getTime() - birthdate.getTime();  
    int age = (int) (ageInMillis / (1000 * 60 * 60 * 24 * 365.25));  
  
    double incomeMinusDebt = income - debt;  
  
    writer.write(username + " " + values[1] + "," + age + "," + income + "," + debt + "," + incomeMinusDebt + "\n");  
}  
  
reader.close();  
writer.close();  
  
System.out.println("ETL process completed.");  
} catch (IOException | ParseException e) {  
    e.printStackTrace();  
}
```

Figura 5. Código Java segunda parte

IV. EJECUCIÓN DE CÓDIGO

- Sistema Operativo: Windows 11
 - Procesador: Core I5 11va Gen
 - RAM: 16GB
 - Disco duro SSD: 512 GB

Intentos en Python:

Primer intento en Python:

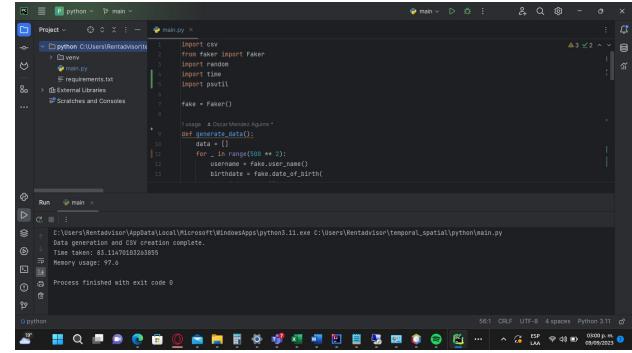


Figura 6. Primer intento en Python

Segundo intento en Python:

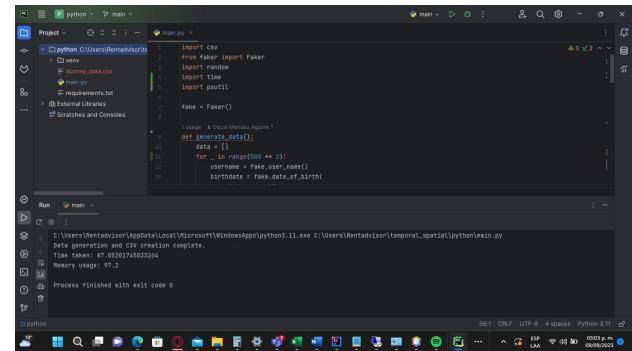


Figura 7. Segundo intento en Python

Tercer intento en Python:

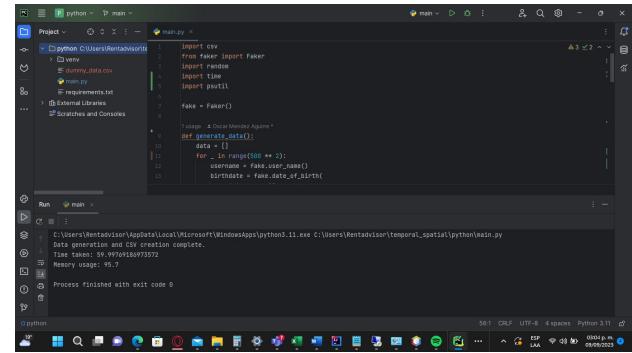


Figura 8. Tercer intento en Python

Intentos en Java:

Primer intento Cpu Time Java:

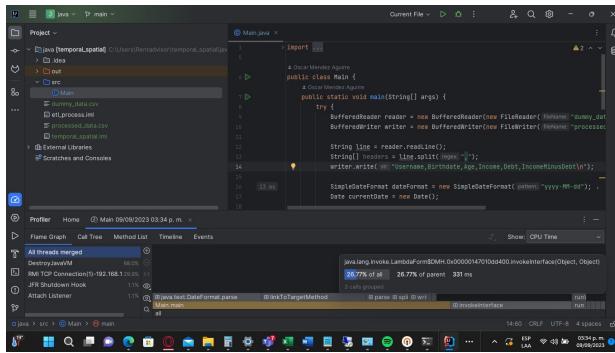


Figura 9. Primer intento Cpu Time Java

Primer intento Memory Allocations Java:

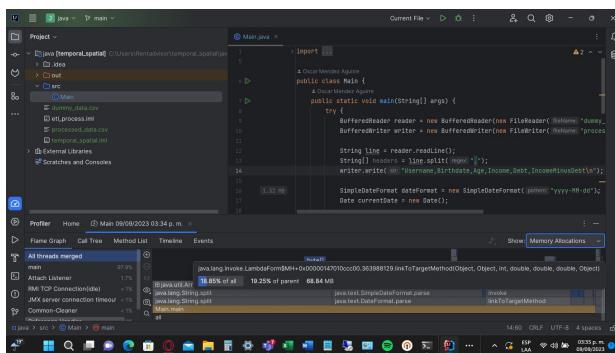


Figura 10. Primer intento Memory Allocations Java

Primer intento Total Time Java:

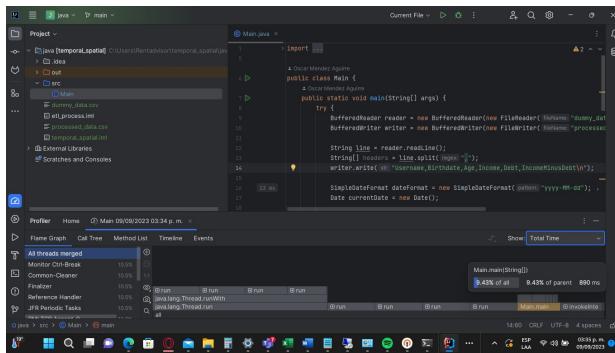


Figura 11. Primer intento Total Time Java

Segundo intento Cpu Time Java:

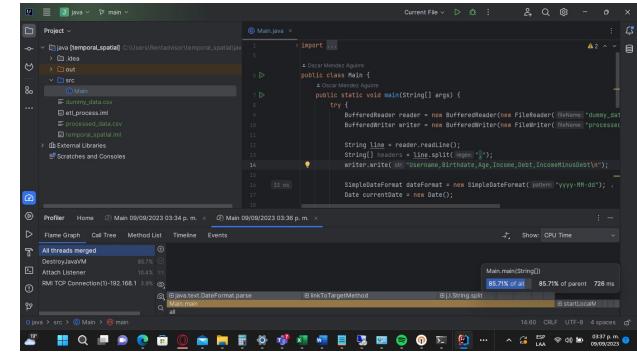


Figura 12. Segundo intento Cpu Time Java

Segundo intento Memory Allocations Java:

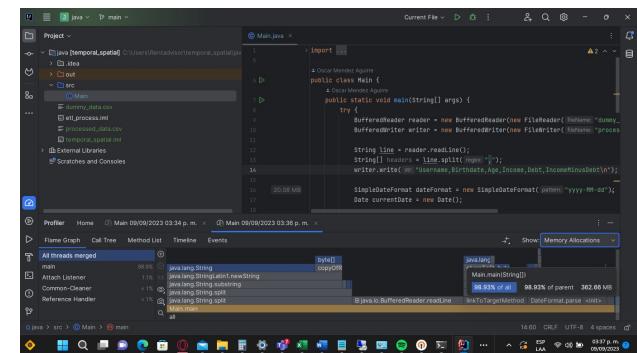


Figura 13. Segundo intento Memory Allocations Java

Segundo intento Total Time Java:

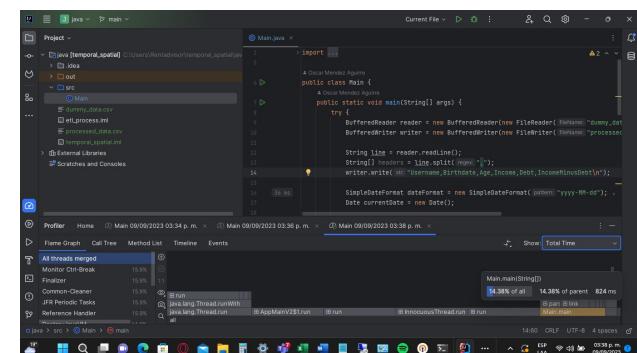


Figura 14. Segundo intento Total Time Java

Tercer intento Cpu Time Java:

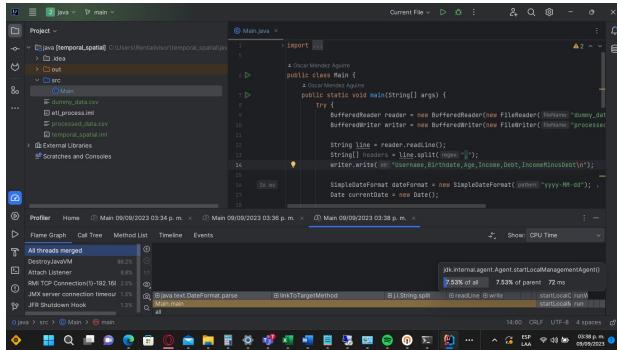


Figura 15. Tercer intento Cpu Time Java

Tercer intento Memory Allocations Java:

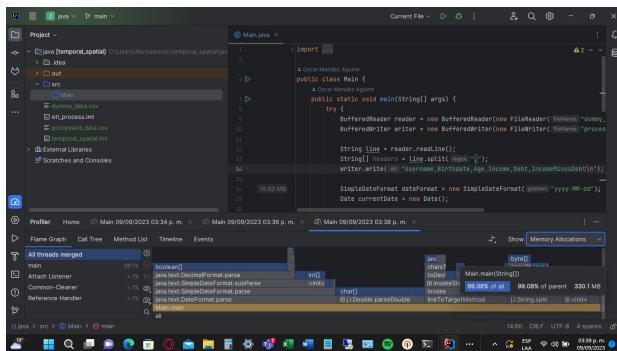


Figura 16. Tercer intento Memory Allocations Java

Tercer intento Total Time Java:

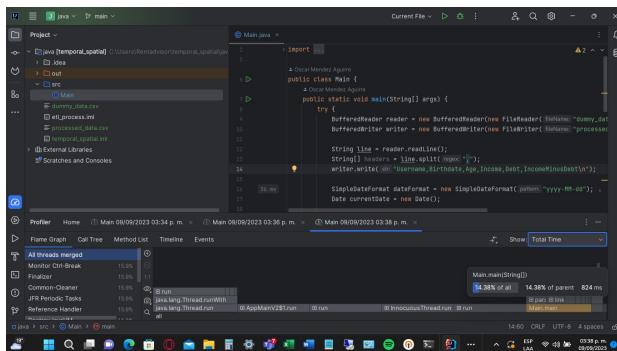


Figura 17. Tercer intento Total Time Java

- Sistema Operativo: Windows 11
- Procesador: Core I5 12va Gen
- RAM: 16 GB
- Disco duro SSD: 512 GB

Intentos en Python:

Primer intento en Python:

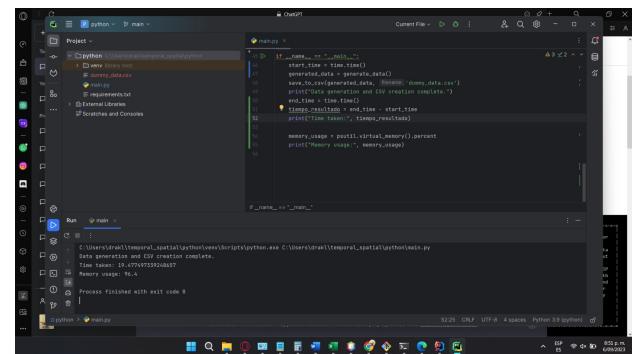


Figura 18. Primer intento en Python

Segundo intento en Python:

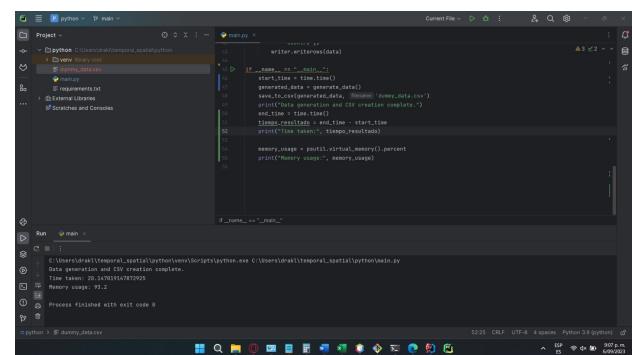


Figura 19. Segundo intento en Python

Tercer intento en Python:

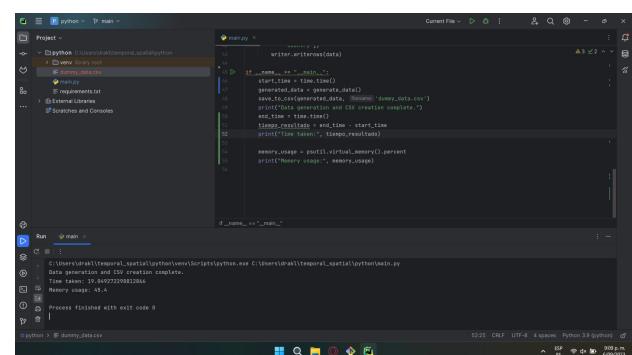


Figura 20. Tercer intento en Python

Intentos en Java:

Primer intento Cpu Time Java:

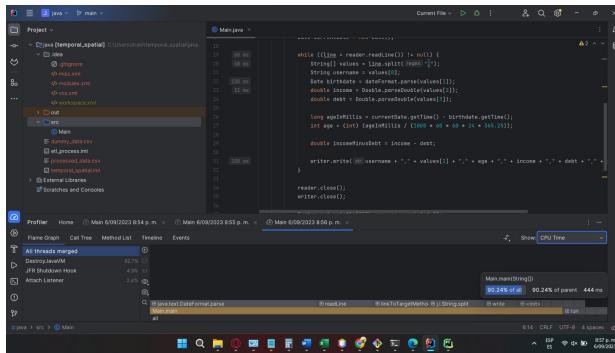


Figura 21. Primer intento Cpu Time Java

Primer intento Memory Allocations Java:

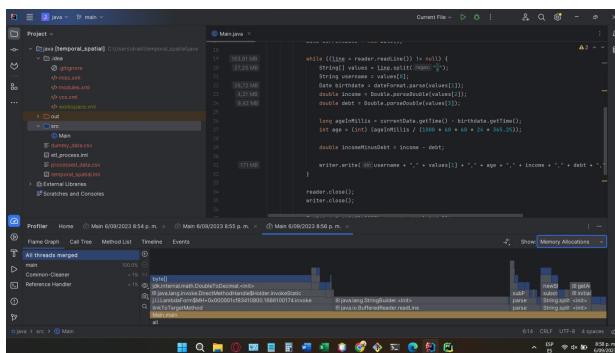


Figura 22. Primer intento Memory Allocations Java

Primer intento Total Time Java:

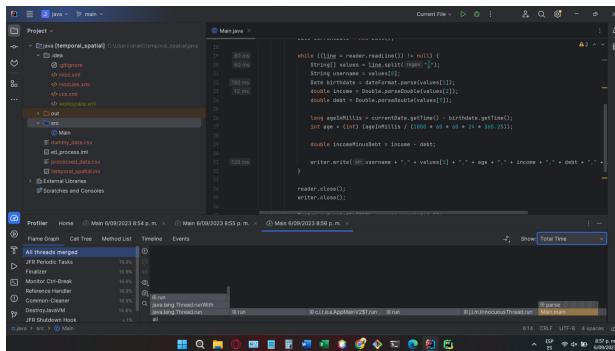


Figura 23. Primer intento Total Time Java

Segundo intento Cpu Time Java:

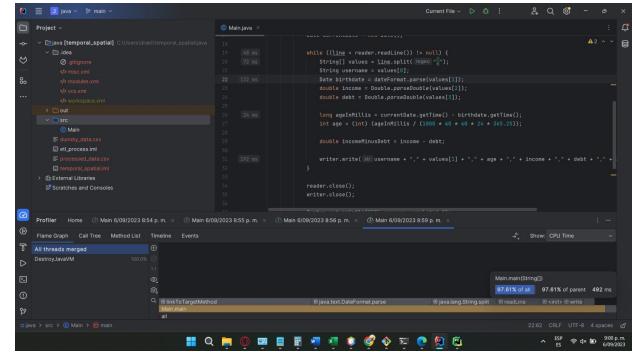


Figura 24. Segundo intento Cpu Time Java

Segundo intento Memory Allocations Java:

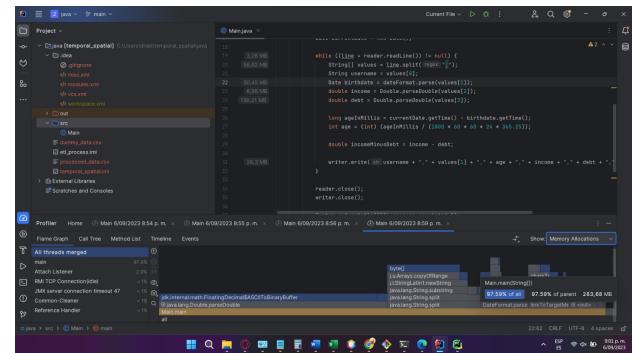


Figura 25. Segundo intento Memory Allocations Java

Segundo intento Total Time Java:

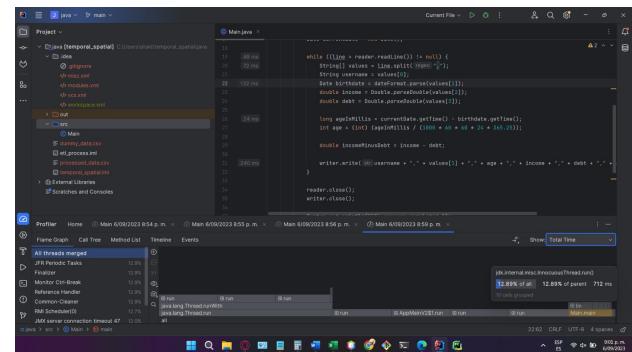


Figura 26. Segundo intento Total Time Java

Tercer intento Cpu Time Java:

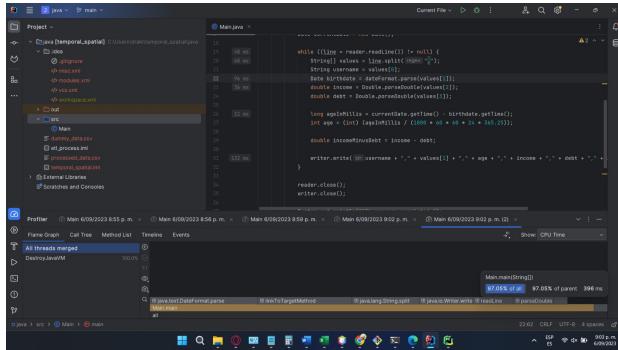


Figura 27. Tercer intento Cpu Time Java

Tercer intento Memory Allocations Java:

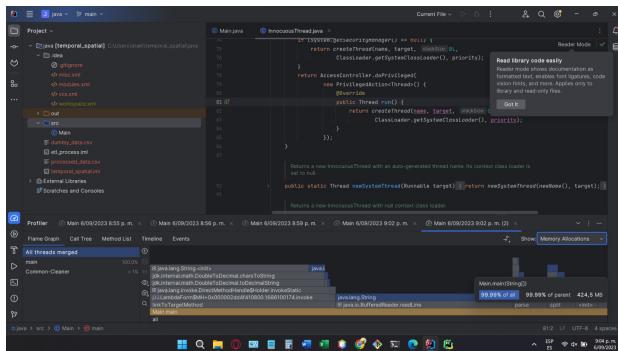


Figura 28. Tercer intento Memory Allocations Java

Tercer intento Total Time Java:

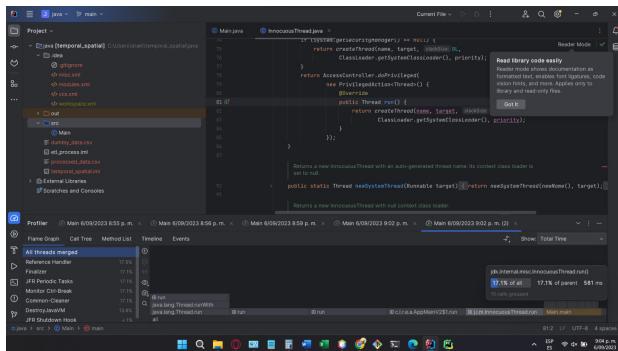


Figura 29. Tercer intento Total Time Java

- Sistema Operativo: Windows 10
- Procesador: Core I3 10va Gen
- RAM: 8 GB
- Disco duro SSD: 256 GB

Intentos en Python:

Primer intento en Python:

```

import csv
from faker import Faker
import random
import os
import string
import math

fake = Faker()

# Usage: >python Agencia.py
def generate_data():
    data = []
    for i in range(1000):
        username = fake.user_name()
        birthdate = fake.date_of_birth()
        location = fake.location()
        income = round(random.uniform(0, 100000), 2)
        debt = round(random.uniform(0, 90000), 2)
        sex = random.choice(['Male', 'Female'])
        num_children = random.randint(0, 5)

    generate_data()

```

Figura 30. Primer intento en Python

Segundo intento en Python:

```

import csv
from faker import Faker
import random
import os
import string
import math

fake = Faker()

# Usage: >python Agencia.py
def generate_data():
    data = []
    for i in range(1000):
        username = fake.user_name()
        birthdate = fake.date_of_birth()
        location = fake.location()
        income = round(random.uniform(0, 100000), 2)
        debt = round(random.uniform(0, 90000), 2)
        sex = random.choice(['Male', 'Female'])
        num_children = random.randint(0, 5)

    generate_data()

```

Figura 31. Segundo intento en Python

Tercer intento en Python:

```

import csv
from faker import Faker
import random
import os
import string
import math

fake = Faker()

# Usage: >python Agencia.py
def generate_data():
    data = []
    for i in range(1000):
        username = fake.user_name()
        birthdate = fake.date_of_birth()
        location = fake.location()
        income = round(random.uniform(0, 100000), 2)
        debt = round(random.uniform(0, 90000), 2)
        sex = random.choice(['Male', 'Female'])
        num_children = random.randint(0, 5)

    generate_data()

```

Figura 32. Tercer intento en Python

Intentos en Java:

Primer intento Cpu Time Java:

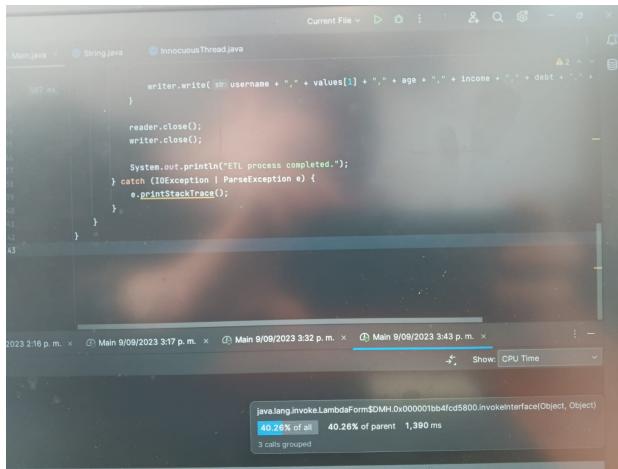


Figura 33. Primer intento Cpu Time Java

Primer intento Memory Allocations Java:

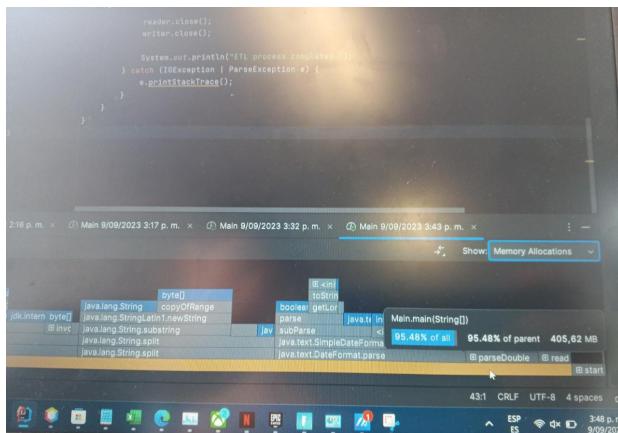


Figura 34. Primer intento Memory Allocations Java

Primer intento Total Time Java:

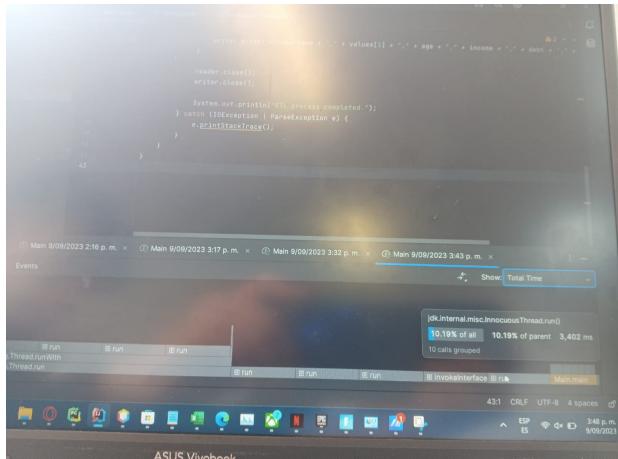


Figura 35. Primer intento Total Time Java

Segundo intento Cpu Time Java:

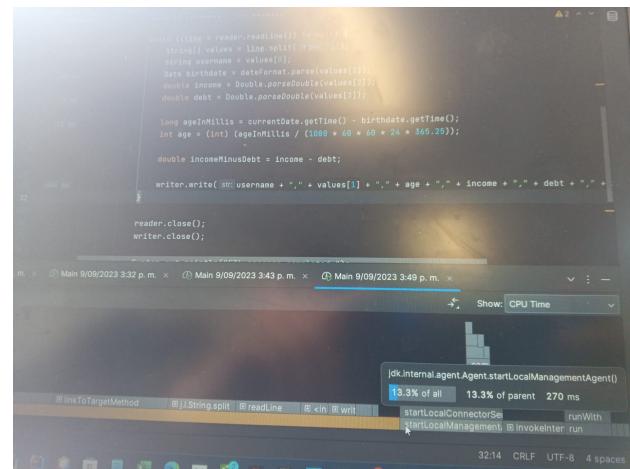


Figura 36. Segundo intento Cpu Time Java

Segundo intento Memory Allocations Java:

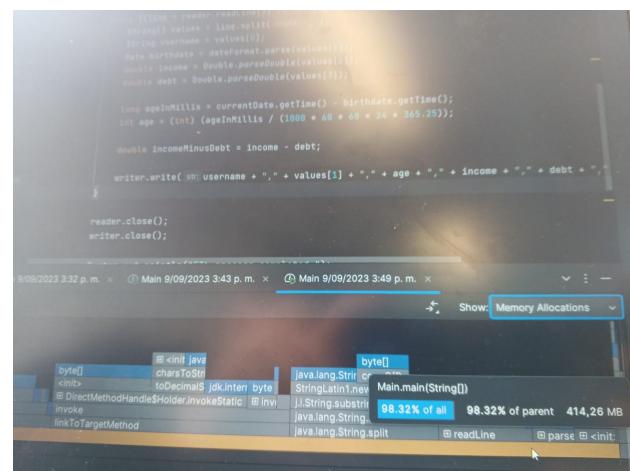


Figura 37. Segundo intento Memory Allocations Java

Segundo intento Total Time Java:

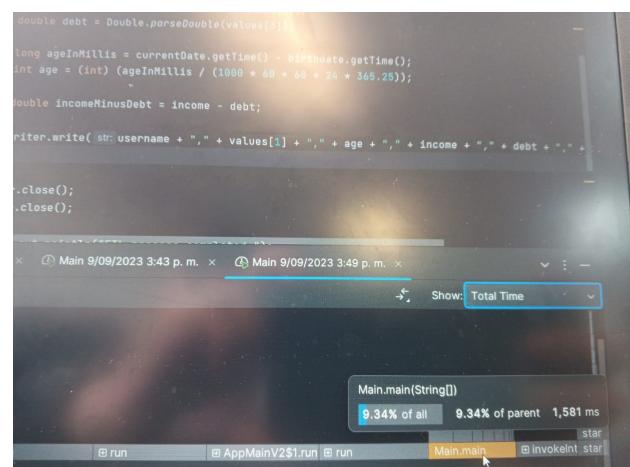


Figura 38. Segundo intento Total Time Java

Tercer intento Cpu Time Java:

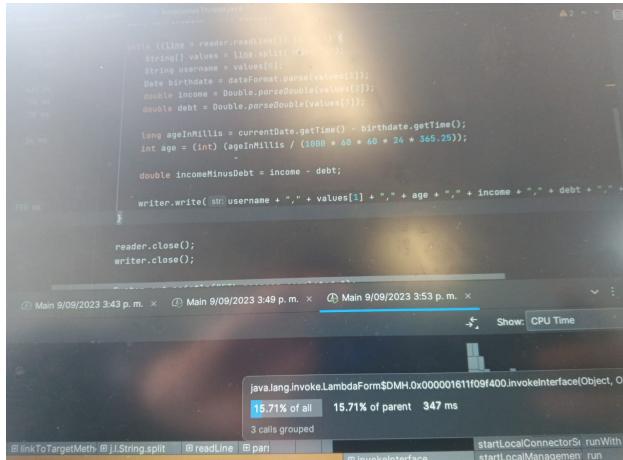


Figura 39. Tercer intento Cpu Time Java

Tercer intento Memory Allocations Java:

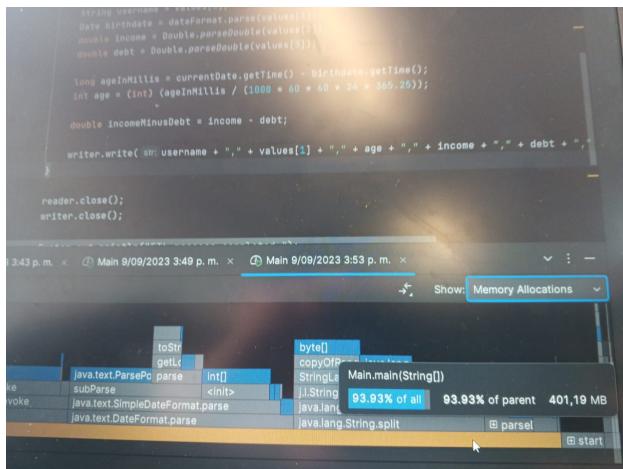


Figura 40. Tercer intento Memory Allocations Java

Tercer intento Total Time Java:

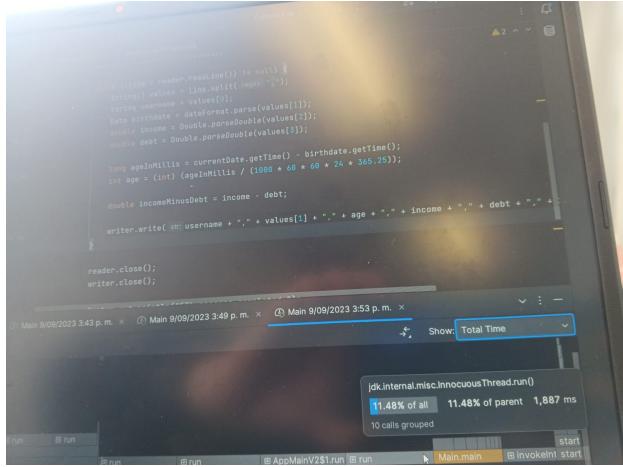


Figura 41. Tercer intento Total Time Java

V. CONCLUSIONES

Tras un análisis comparativo de tres máquinas diferentes, se extraen las siguientes conclusiones:

El rendimiento de los scripts de Python y Java varía según la máquina utilizada. Las diferencias en la capacidad de procesamiento y memoria pueden afectar significativamente el tiempo de ejecución y el consumo de recursos. La complejidad temporal de los guiones puede variar mucho. Los algoritmos y estructuras de control utilizados en cada script afectan el tiempo de ejecución. La complejidad de la sala también varía de una máquina a otra. El uso de la memoria depende de factores como la cantidad de datos procesados y las operaciones realizadas. Los cálculos acelerados revelan cuánto más rápido se ejecuta la versión mejorada del script en comparación con la versión anterior en un entorno determinado.

En general, este proyecto demuestra la importancia de considerar tanto el hardware como el software al evaluar el rendimiento y la complejidad del script. Los resultados obtenidos ayudan a comprender cómo diferentes factores afectan el comportamiento de las aplicaciones en diferentes entornos.