Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de Ciencias Computacionales.

Ingeniería en Computación

Actividad 3.

Programa 1. Simular el Procesamiento por Lotes

Materia: Sistemas Operativos ***Sección:*** D04

Profesor: Becerra Velázquez Violeta del Rocío.

Equipo:

- Carriola Navarro Aldo Fernando | 221978167
- Cervantes Araujo Maria Dolores | 217782452

Fecha: 10/09/2023

Índice

Tabla de Imágenes

Objetivos

El procesamiento por lotes fue de los primeros métodos que encontraron los programadores para gestionar una cantidad grande de datos de manera secuencial; con el paso del tiempo esto fue mejorando, integrándose en los equipos de hoy en día, algunos de manera interna y otros ejecutados por el mismo usuario.

Es por ello, que este primer programa tiene gran importancia dado que nos permite comprender de mejor manera el proceso largo que se llevaba a cabo en aquel entonces; contando con los siguientes objetivos:

1. Comprender la importancia y utilidad de los procesamientos por lotes.
2. Crear un programa que permita el desarrollo de habilidades necesarias para lograr que funcione cada proceso de manera secuencial.
3. Análisis de algoritmos para dar soluciones a los distintos problemas que surjan con la capacidad de programación de cada persona, sin perder el enfoque del proyecto.
4. Aprender a trabajar en equipo, no cerrarse a una única solución o lógica de programación, estar abierto al diálogo para poder avanzar.

Notas Sobre el Lenguaje

El lenguaje utilizado para llevar a cabo el programa fue *Python*, dado que nos permite experimentar con diversas funciones que apoyan en la creación del programa, permitiéndonos que sea orientado a objetos, estructural, con interfaz, a consola, con color e incluso animación.

Dado que es un lenguaje de nuestro total conocimiento y dominio para ambos, optamos por crear el programa de forma estructural, es decir, mediante funciones. Buscando siempre mediante acuerdos, que ambos nos sintiéramos familiarizados con el lenguaje y lógica del programa, de esta manera optimizamos tiempos y nos fue más sencillo modelar y visualizar el programar conforme lo esperado.

TDA o Estructuras Utilizadas en el Programa

Todo el código de nuestro programa se implementa en un solo archivo con programación estructural o secuencial, se utilizó el TDA lista de listas, que nos permitió una mejor manipulación con los datos al momento de reflejarlos en las tablas.

Primero definimos todas las funciones que necesitaríamos a lo largo del programa, que son un total de 8 funciones.

```python
import os
from tabulate import tabulate
import time as t

IDs = []

def cantidad_procesos():···


def ID():···


def nombre():···


def isOperador():···


def tiempo_estimado():···


def lotes_de_procesos():···


def imprimir():···


def main():···

main()
```

Ilustración 1. Definición de funciones

Las primeras 5 funciones, son las encargadas de capturar por el usuario los datos necesarios para almacenarlos en una lista encargada de seccionar según el número de procesos que ingrese el usuario.

Se piden datos como cantidad de procesos, id (así como sus respectivas validaciones para que no se repitan los mismos), nombre, validación para el tipo de operación que realizaría y almacenado de información mediante *f"{num1}{operador}{num2}"* que posteriormente nos será de utilidad para mostrar la operación en la tabla de procesos en ejecución, y por último capturar el tiempo estimado.

La función "***lotes_de_procesos()"*** es la que se ejecuta cuando el usuario ingresa al programa, aquí implementamos una estructura de control repetitiva, haciendo uso de un ciclo *while* para iterar las 5 funciones previas n veces de procesos y crear la lista de listas.

En la función ***"imprimir()"*** creamos el diseño que se mostrará en consola para el usuario, aquí utilizamos el atributo de la librería *tabulate* para crear un formato organizado mediante tablas, haciéndolo visualmente más agradable para el usuario.

```python
while contProcess != numProcess:

    contPendiente -= 1
    lote_actual = []
    proceso_actual = []
    procesos_terminados = []
    contLote += 1

    for _ in range(procesosMax):
        if not procesos_pendientes:
            break       #rompe el programa

        proceso = procesos_pendientes.pop(0)
        lote_actual.append(proceso)

    while len(lote_actual) != 0:
        contTiempo = 0

        actual = lote_actual.pop(0)
        proceso_actual.append(actual)

        while True:
            os.system("cls")
            contTiempo += 1
            contadorGeneral += 1
            imprimir()
            t.sleep(1)

            if tiempo_restante == 0:
                t.sleep(1)
                break

        finalizado = proceso_actual.pop(0)
        procesos_terminados.append(finalizado)
        contProcess += 1

    if len(procesos_terminados) != len(lote_actual):
        os.system("cls")
        imprimir()
        t.sleep(2)
```

Ilustración 2. Estructura de función main

En nuestra función principal ***"main()"*** se encuentra la estructura de la *ilustración 2*, que es la encargada de simular el procesamiento por lotes, una vez que almacena la información en listas, se crea el algoritmo para limitar cada lote con 5 procesos como máximo, creando una simulación secuencial para cada proceso por lote y creando excepciones en todo momento previniendo errores; una vez que termina con el último proceso del último lote, simplemente realiza un congelado de pantalla que nos permite apreciar el último lote.

Funcionamiento

A continuación, se presentan algunas ilustraciones que reflejan el funcionamiento del programa de simulación con un ejemplo de 8 lotes:

1. Primero el programa pregunta las n veces que capturará datos:

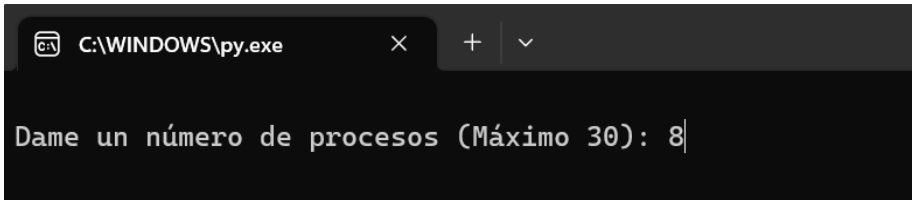

Ilustración 3. Captura de datos 1.

2. Captura de datos y comprobación de validaciones establecidas en clase para la entrada de datos:

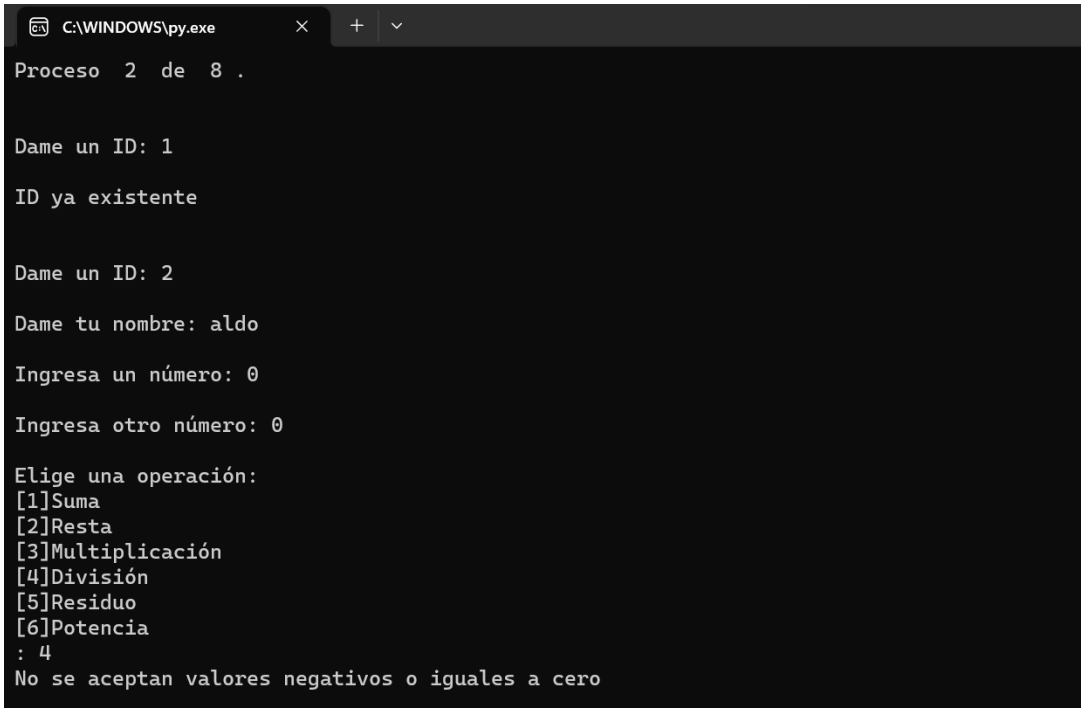

Ilustración 4. Captura y validación de datos 1.1.

```
No se aceptan valores negativos o iguales a cero

Ingresa un número: 4

Ingresa otro número: 2

Elige una operación:
[1]Suma
[2]Resta
[3]Multiplicación
[4]División
[5]Residuo
[6]Potencia
: 3

Dame el tiempo estimado: -1

El tiempo estimado debe ser un número mayor que 1.


Dame el tiempo estimado: 0

El tiempo estimado debe ser un número mayor que 1.


Dame el tiempo estimado: 6
```

Ilustración 5. Captura y validación de dato 1.2

```
C:\WINDOWS\py.exe

Proceso  6  de  8 .


Dame un ID: lalo

Por favor, ingresa un número válido.
```

Ilustración 6. Captura y validación de datos 1.3

3. Resultado del funcionamiento de la simulación por lotes:

```
C:\WINDOWS\py.exe

No. Lotes Pendientes:  1

Lote Actual:  1          Proceso en Ejecución          Procesos Terminados

| ID | TME |
|---|---|
| 2 | 6 |
| 3 | 4 |
| 4 | 8 |
| 5 | 5 |

| Nom | lola |
|---|---|
| Id | 1 |
| Ope | 21+23 |
| TME | 4 |
| TT | 2 |
| TR | 2 |

| ID | Ope | Res |
|---|---|---|

Contador:  2
```

Ilustración 7. Resultado 1.

Simular el Procesamiento por Lotes

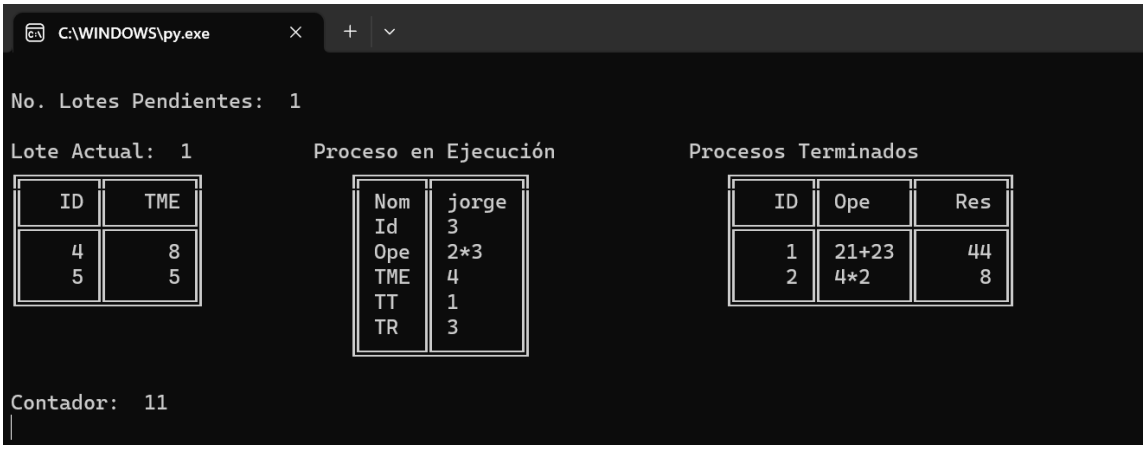

Ilustración 8. Resultado 1.1

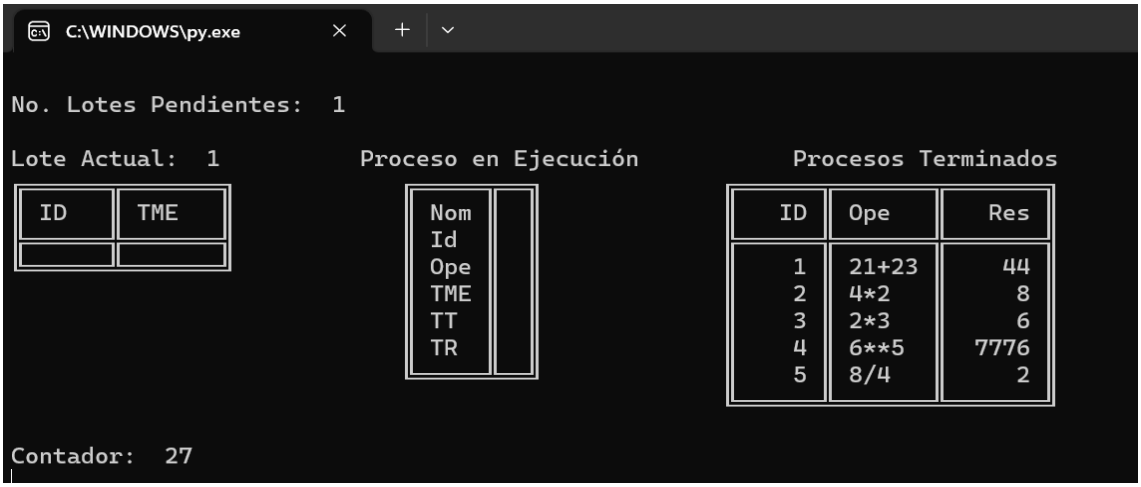

Ilustración 9. Resultado 1.2

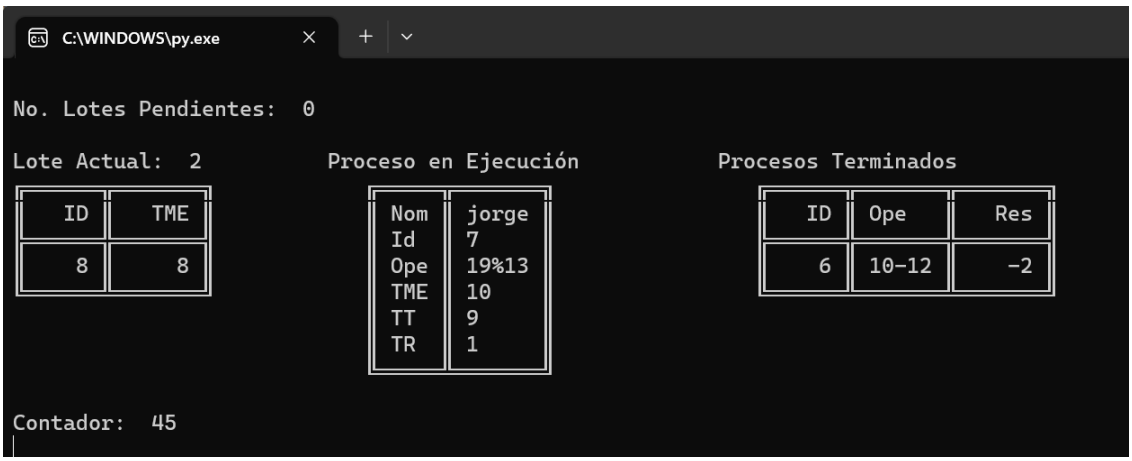

Ilustración 10. Resultado 1.3

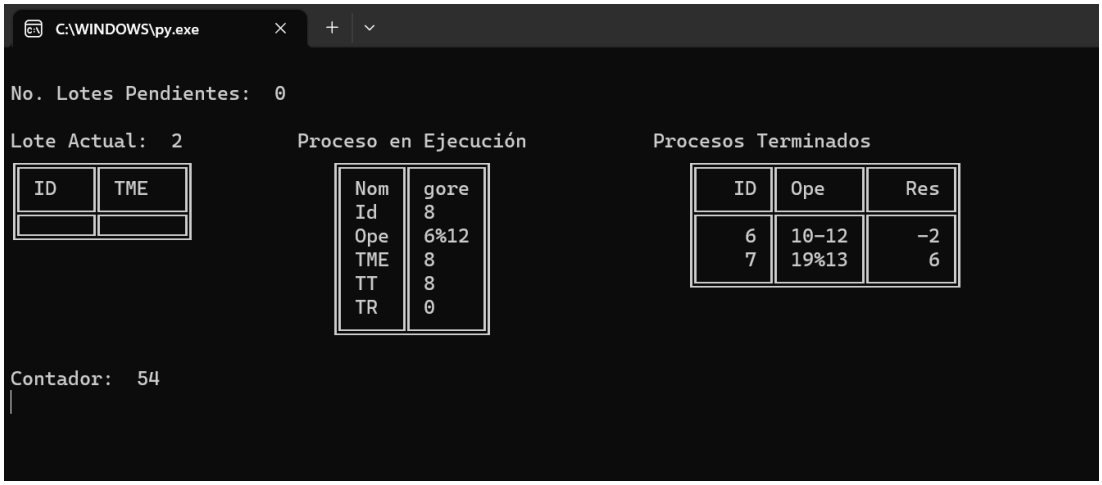

Ilustración 11. Resultado 1.4

Errores y Soluciones

Mostrar las tablas de forma horizontal: Al momento de querer hacer la simulación con las tablas de manera que se imprimieran en consola horizontalmente, me mostraba este error:

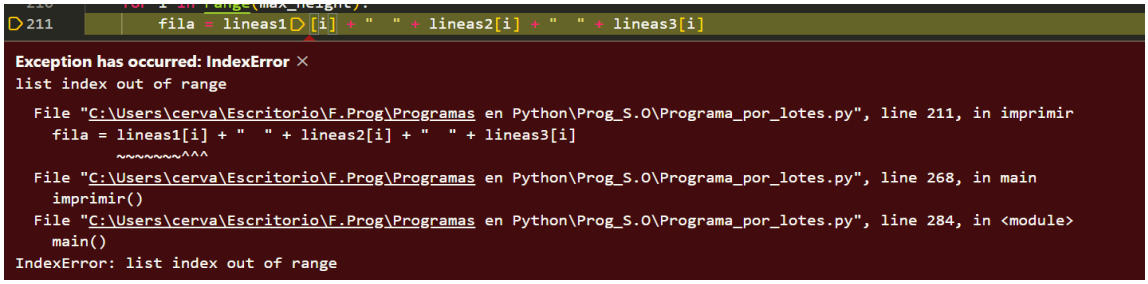

Ilustración 12. Error 1.

Después de analizar el código por arduas horas, nos percatamos que el error provenía desde que separábamos la tabla en líneas y se trataba de ajustarlas pese a sus diferentes tamaños:

```
# Obtener el número de líneas en cada tabla
num_lines1 = len(tabla1.splitlines())
num_lines2 = len(tabla2.splitlines())
num_lines3 = len(tabla3.splitlines())

# Determinar la altura máxima entre las tres tablas
max_height = max(num_lines1, num_lines2, num_lines3)

# Rellenar las tablas más cortas con filas en blanco si es necesario
if num_lines1 < max_height:
    tabla1 += "\n" * (max_height - num_lines1)
if num_lines2 < max_height:
    tabla2 += "\n" * (max_height - num_lines2)
if num_lines3 < max_height:
    tabla3 += "\n" * (max_height - num_lines3)

# Dividir cada tabla en líneas y combinarlas horizontalmente
lineas1 = tabla1.splitlines()
lineas2 = tabla2.splitlines()
lineas3 = tabla3.splitlines()
```

Ilustración 13. Causa del error

La manera en que lo solucionamos fue separando primero cada tabla en líneas y después de cada una sacar la cantidad de total de líneas, nos dimos cuenta de que estábamos siendo redundantes con el proceso anterior, por último, restamos el número total de líneas correspondiente a la tabla con más información y se lo disminuimos a cada tabla, para adaptar lo sobrante, manipulando la tabla con un arreglo, quedando el algoritmo de la siguiente manera:

```
# ----------------- PARA MOSTRAR EN HORIZONTAL
# Dividir cada tabla en líneas
lineas1 = tabla1.splitlines()
lineas2 = tabla2.splitlines()
lineas3 = tabla3.splitlines()

# Determinar el número máximo de líneas entre las tres tablas
max_lines = max(len(lineas1), len(lineas2), len(lineas3))

# Rellenar las tablas más cortas con líneas en blanco si es necesario
lineas1 += ['\t\t'] * (max_lines - len(lineas1))
lineas2 += ['\t'] * (max_lines - len(lineas2))
lineas3 += [''] * (max_lines - len(lineas3))

# Alinear horizontalmente las líneas de las tablas
tabla_final = ''
for i in range(max_lines):
    fila = lineas1[i] + "\t    " + lineas2[i] + " \t\t   " + lineas3[i]
    tabla_final += fila + '\n'

print(tabla_final)
print("Contador: ", contadorGeneral)
```

Ilustración 14. Solución 1.

Excepción para que haga una pausa una vez que termine un lote: El algoritmo al principio creaba una pausa solo cuando estaba en ejecución un proceso, por el tiempo transcurrido y tiempo restante, por lo que se realizó un apartado para que no cerrara el programa sin antes visualizar la comprobación de los procesos terminados, creamos la siguiente condicional:

```
if len(procesos_terminados) != len(lote_actual):
    os.system("cls")
    imprimir()
    t.sleep(2)
```

Ilustración 15. Error 2

Pero presentamos error con la tabla de "*procesos en ejecución",* dado que al ejecutarlo daba un error idéntico al anterior por los datos que requiere la tabla:

```
178     tiempo_estimado = proceso_actual[0][4]

Exception has occurred: IndexError ×
list index out of range
  File "C:\Users\cerva\Escritorio\F.Prog\Programas en Python\Prog_S.O\Proyecto_lotes.py", line 178, in imprimir
    tiempo_estimado = proceso_actual[0][4]
                      ~~~~~~~~~~~~~~~^^^
  File "C:\Users\cerva\Escritorio\F.Prog\Programas en Python\Prog_S.O\Proyecto_lotes.py", line 278, in main
    imprimir()
  File "C:\Users\cerva\Escritorio\F.Prog\Programas en Python\Prog_S.O\Proyecto_lotes.py", line 281, in <module>
    main()
IndexError: list index out of range
```

Ilustración 16. Error 2.1

Para solucionarlo se creó la siguiente condicional, para poder visualizar los procesos terminados por lote:

```
# ---------------------- PROCESO DE EJECUCIÓN
headersE = ["Nom", "Id", "Ope", "TME", "TT", "TR"]
if proceso_actual:  # Verifica si proceso_actual no está vacío
    tiempo_estimado = proceso_actual[0][4]
    tiempo_restante = tiempo_estimado - contTiempo
    proceso_actual_tabla = [proceso_actual[0][1], proceso_actual[0][0]
else:
    # Si proceso_actual está vacío, establece los valores en blanco
    proceso_actual_tabla = ["", "", "", "", "", ""]
```

Ilustración 17. Solución 2.

Excepciones para la división y potencia: Por otro lado, como sabíamos que no se puede dividir un número entre ‘0’ o dividir ‘0’ entre cualquier otro número, además, de que dicho número no puede ser elevado con ningún otro número, como se observa en la Ilustración 18.

```
95          elif opc == 4:
96              operador = '/'
97              resultado = num1/num2

Se produjo una excepción: ZeroDivisionError ×
division by zero
```

Ilustración 18. Error 3

Inicialmente, pensamos en poner una condicional en la función “*isOperador()”*donde le arrojaríamos un error al usuario, siempre que ingresará el número 0 en cualquiera de los dos enteros solicitados. Sin embargo, nos percatamos que esto sería un gran error lógico en nuestro programa, ya que, ciertamente si se puede hacer una suma, resta o multiplicación con números ceros. Así que, nuestra solución fue colocar las siguientes estructuras condicionales, para cada caso distinto, tal como se muestra en la Ilustración 19.

```
elif opc == 4:
    if num1 > 0 or num2 > 0:
        operador = '/'
        resultado = num1/num2
        break
    else:
        print("No se aceptan valores negativos o iguales a cero")

elif opc == 5:
    if num1 > 0 or num2 > 0:
        operador = '%'
        resultado = num1%num2
        break
    else:
        print("No se aceptan valores negativos o iguales a cero")

elif opc == 6:
    if num1 > 0 and num2 > 0:
        operador = '**'
        resultado = num1**num2
        break
    else:
        print("Cero no se puede elevar a cero")
```

Ilustración 19. Solución 3

Procesamiento solo del primer proceso por lote: Al estar programando finalmente la lógica dentro de nuestra función *'main()'* se había logrado que se separará la impresión en pantalla por lotes, es decir, en segmentos de cinco procesos o menos en caso de no alcanzar el número máximo de procesos. Tal como se muestra en la Ilustración 20.

```
for _ in range(procesosMax):
    if not procesos_pendientes:
        break       #rompe el programa

    proceso = procesos_pendientes.pop(0)
    lote_actual.append(proceso)

    contTiempo = 0

    actual = lote_actual.pop(0)
    proceso_actual.append(actual)
```

Ilustración 20. Error 4

Solucionando el error anterior, creando una nueva condicional donde se esté comparando la longitud de nuestro lote actual sea diferente a cero, para que así se termine por ejecutar todo nuestro lote actual en su totalidad antes de pasar al siguiente:

```
while len(lote_actual) != 0:
    contTiempo = 0

    actual = lote_actual.pop(0)
    proceso_actual.append(actual)
```

Ilustración 21. Solución 4

Conclusiones

Maria Dolores.

Al termino de este primer programa, puedo concluir que el procesamiento por lotes es importante dado que internamente ejecuta procesos que requiere el computador para realizar todas las tareas que nosotros determinamos o para mantenerse actualizado y optimizado, sin la intervención manual del usuario. Fue una actividad interesante para entender cómo fue que comenzaron los avances que conocemos hoy en día y particularmente para no perder las habilidades en programación.

Personalmente lo más complicado al principio de crear el programar, fue poder organizarme con mi compañero, en tiempos y actividades, poder estar los dos involucrados en todo momento para saber de qué trata cada línea de código y poder continuar uno tras otro en la lógica y depuración; al final, nos complementamos bastante bien y nos permitió aprender un poco más de los alcances del lenguaje con ciertas librerías.

Aldo Fernando.

En lo personal, la creación de este primer programa me ha servido como una gran retroalimentación tanto para practicar la programación estructurada en Python como para poder comprender como es el funcionamiento del procesamiento por lotes. Ya que, a pesar de estar usando programación avanzada para simular una actividad de procesos de hace muchos años, supone un gran reto el buscar la forma de poder adaptar estructuras y lógicas complejas para realizar simplemente el resultado de una operación cada cierto tiempo por procesos ya su vez por lotes.

En adición, la realización de este código me ha permitido comprender como poder hacer una buena impresión en la consola, sin necesidad de una interfaz gráfica, además, desarrollar mi lógica sobre cómo hacer funcionar varias estructuras dentro de una función de forma secuencial, con sus respectivas excepciones y tiempos respectivamente.

Finalmente, puedo concluir que me ha resultado muy genuino trabajar con mi compañera para la creación de este primer programa satisfactoriamente. Esperando y analizando las próximas mejoras que le haremos a nuestro código para el programa siguiente.