

INFORME PA2

LEIVA DAMIAN ANDRES DANIEL

CAPÍTULO 1: ANÁLISIS DEL PROBLEMA

1. Descripción del problema

El sistema busca administrar un conjunto de procesos simulados por el usuario. Cada proceso tiene atributos como nombre, prioridad, duración y estado. El objetivo es permitir agregar procesos, modificar su estado, listarlos y eliminarlos de manera ordenada. El sistema debe facilitar la administración básica de procesos mediante una interfaz interactiva basada en menús, que guíe al usuario paso a paso en cada acción.

2. Estructuras de datos propuestas

Se utilizará una lista dinámica para almacenar los procesos. Cada proceso será representado por un objeto que contiene sus atributos principales: identificador, nombre, duración, prioridad y estado. La lista permite insertar, buscar, modificar y eliminar elementos de forma sencilla, lo cual es ideal para este tipo de aplicación en consola.

3. Justificación de la elección

La lista dinámica es adecuada porque:

Permite almacenar una cantidad variable de procesos.

Ofrece acceso rápido para recorrer todos los elementos.

Facilita la implementación de operaciones como agregar, cambiar estados y eliminar procesos.

No requiere memoria fija desde el inicio, lo que la hace flexible.

CAPÍTULO 2: DISEÑO DE LA SOLUCIÓN

1. Descripción de estructuras de datos y operaciones

Estructura principal: Lista de procesos.

Cada proceso contiene:

- id: número único generado automáticamente.
- nombre: identificación escrita por el usuario.
- prioridad: valor numérico asignado.
- duracion: tiempo estimado.

- estado: valores posibles como pendiente, ejecutando o finalizado.

Operaciones implementadas:

- Agregar proceso.
- Cambiar estado de un proceso.
- Mostrar lista de procesos.
- Eliminar proceso.
- Buscar proceso por id.

2. Algoritmos principales

Pseudocódigo para agregar un proceso:

Inicio

Solicitar nombre del proceso

Solicitar duración

Solicitar prioridad

Crear objeto proceso con datos ingresados

Asignar id automático

Asignar estado inicial "pendiente"

Agregar proceso a la lista

Fin

Pseudocódigo para cambiar estado de un proceso:

Inicio

Solicitar id del proceso a modificar

Buscar proceso en la lista

Si no existe, mostrar mensaje y terminar

Mostrar estados disponibles

Solicitar nuevo estado

Actualizar valor del estado en el proceso

Fin

3. Diagramas de flujo:

Diagrama para agregar proceso:

Inicio → Ingresar datos → Crear proceso → Insertar en lista → Mensaje de éxito → Fin

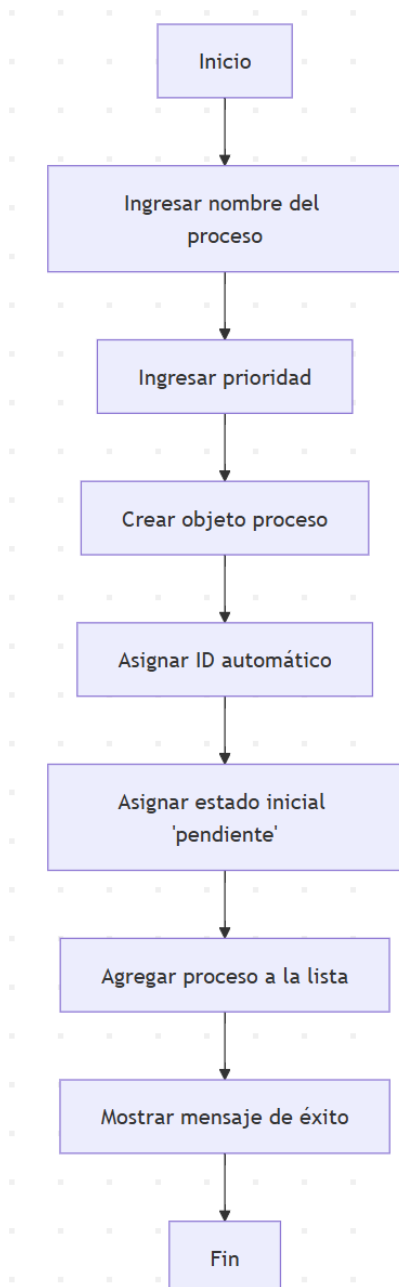
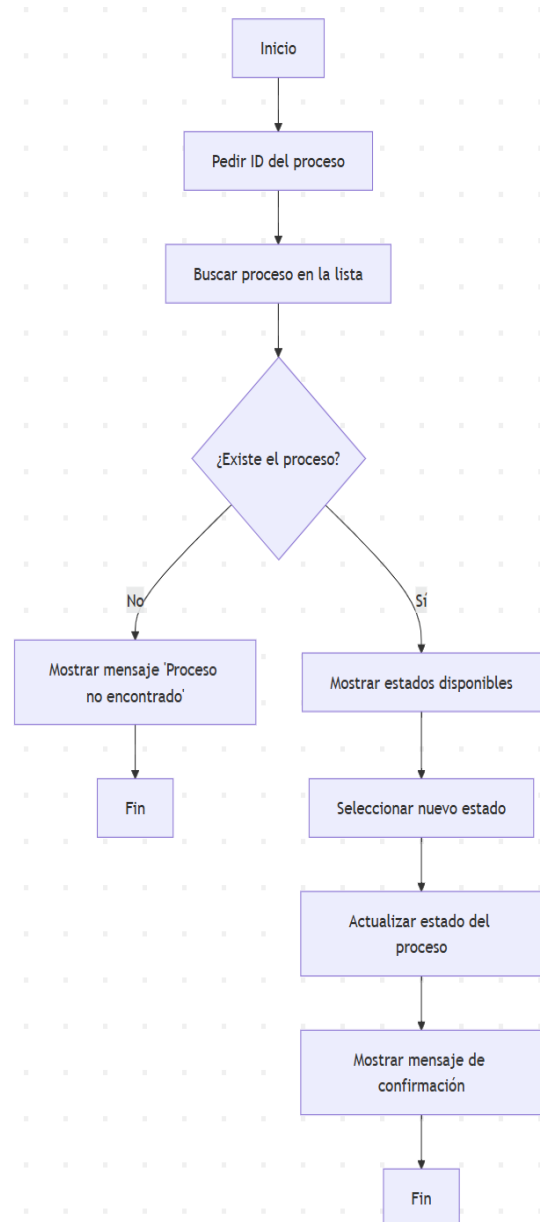


Diagrama para cambiar estado:

Inicio → Pedir id → Buscar proceso → ¿Existe? → No: mensaje y fin / Sí: mostrar opciones → seleccionar estado → actualizar valor → Fin



4. Justificación del diseño

El diseño basado en una lista dinámica permite una administración flexible y eficiente. La estructura es fácil de recorrer y manipular, y su simplicidad facilita el mantenimiento. Los algoritmos seleccionados son directos, con pasos claros

que permiten una ejecución rápida y estructurada. La interfaz mediante menú mejora la interacción, ya que guía al usuario sin necesidad de comandos complejos.

CAPÍTULO 3: SOLUCIÓN FINAL

1. Código limpio y estructurado

El programa final está implementado en un archivo principal con un menú interactivo. Incluye clases separadas para manejar los procesos, mejorar el orden del código y facilitar modificaciones futuras. Se incluyen validaciones de entrada para evitar errores.

2. Capturas de pantalla

Las capturas deben mostrar:

- El menú principal.

```
=====
SISTEMA DE GESTIÓN DE TAREAS
=====

1. Agregar tarea
2. Listar tareas
3. Cambiar estado
4. Marcar como urgente
5. Sacar tarea urgente (pila)
6. Mostrar tabla 2D
0. Salir
```

- El ingreso de nuevos procesos.

```
=====
SISTEMA DE GESTIÓN DE TAREAS
=====

1. Agregar tarea
2. Listar tareas
3. Cambiar estado
4. Marcar como urgente
5. Sacar tarea urgente (pila)
6. Mostrar tabla 2D
0. Salir

Elige una opción: 1

--- AGREGAR TAREA ---
Título: leer
Prioridad (alta, media, baja): alta
Tarea agregada con ID 1.
```

- La modificación del estado.

```
=====
SISTEMA DE GESTIÓN DE TAREAS
=====

1. Agregar tarea
2. Listar tareas
3. Cambiar estado
4. Marcar como urgente
5. Sacar tarea urgente (pila)
6. Mostrar tabla 2D
0. Salir

Elige una opción: 3

--- CAMBIAR ESTADO ---
ID de tarea: 1
Nuevo estado (pendiente, en_progreso, completada): completada
Estado de la tarea 1 actualizado.
```

- Extraer proceso urgente

```
=====
SISTEMA DE GESTIÓN DE TAREAS
=====

1. Agregar tarea
2. Listar tareas
3. Cambiar estado
4. Marcar como urgente
5. Sacar tarea urgente (pila)
6. Mostrar tabla 2D
0. Salir

Elige una opción: 5

--- SACAR URGENTE (PILA) ---
Tarea urgente extraída: estudiar (ID 3)
```

- La visualización de la lista completa.

```
=====
SISTEMA DE GESTIÓN DE TAREAS
=====

1. Agregar tarea
2. Listar tareas
3. Cambiar estado
4. Marcar como urgente
5. Sacar tarea urgente (pila)
6. Mostrar tabla 2D
0. Salir

Elige una opción: 2

--- LISTA DE TAREAS ---
[1] leer - Estado: completada - Prioridad: alta
[2] jugar - Estado: pendiente - Prioridad: alta
[3] estudiar - Estado: pendiente - Prioridad: media
[4] ir a la u - Estado: pendiente - Prioridad: alta
```

3. Manual de usuario

MANUAL DE USUARIO – Sistema de Gestión de Tareas

- El archivo principal del programa: demo.py

1. Introducción

El Sistema de Gestión de Tareas es una aplicación en consola desarrollada en Python.

Su propósito es permitir al usuario registrar, actualizar, clasificar y consultar tareas

utilizando estructuras de datos como diccionarios, pilas y matrices bidimensionales.

Este manual explica cómo ejecutar el programa y cómo utilizar cada una de sus funciones.

2. Requisitos del Sistema

Para utilizar esta aplicación se necesita:

- Python 3.10 o superior
- Un editor o terminal (CMD, PowerShell o VS Code)

3. Cómo ejecutar el programa

1. Abrir la terminal o PowerShell.

2. Navegar a la carpeta donde se encuentra el archivo demo.py. Ejemplo:

```
cd
C:\Users\Usuario\Documentos\Proyecto
```

3. Ejecutar el programa con el siguiente comando:

```
python demo.py
```

4. Al ejecutar, aparecerá el menú principal con las opciones disponibles.

4. Menú Principal

El programa muestra el siguiente menú:

1. Agregar tarea
2. Listar tareas

3. Cambiar estado
4. Marcar como urgente
5. Sacar tarea urgente (pila)
6. Mostrar tabla 2D
0. Salir

Cada opción realiza una acción específica dentro del sistema.

5. Descripción de Opciones

5.1 Agregar tarea

Permite registrar una nueva tarea.

El sistema solicitará:

- Título
- Prioridad (alta, media o baja)

La tarea se guarda con un ID asignado automáticamente.

5.2 Listar tareas

Muestra todas las tareas registradas, con:

- ID
- Título
- Estado
- Prioridad
- Indicación de si es urgente

5.3 Cambiar estado de una tarea

El usuario debe ingresar:

- ID de la tarea
- Nuevo estado (pendiente, en_progreso o completada)

5.4 Marcar tarea como urgente

El usuario ingresa un ID.

La tarea se agrega a la pila de urgentes.

Los urgentes siguen un orden LIFO (último en entrar, primero en salir).

5.5 Sacar tarea urgente (pila)

Extrae la última tarea marcada como urgente.

La operación usa pop() y muestra la tarea extraída.

5.6 Mostrar tabla 2D

Genera e imprime una matriz con el formato:

[ID, título, estado, prioridad, urgente]

Ideal para exportar o visualizar el sistema como si fuera una tabla.

5.7 Salir

Cierra el programa de forma segura.

6. Mensajes de error comunes

- "Error: el título no puede estar vacío."
- "Error: prioridad inválida."
- "No existe una tarea con ese ID."
- "Estado inválido."
- "No hay tareas urgentes."

Estos mensajes indican entradas incorrectas y guían al usuario a corregirlas.

7. Recomendaciones de Uso

- Mantener prioridades correctas (alta, media, baja).
- No ingresar letras cuando se pida un ID.
- Revisar la lista antes de cambiar un estado.
- Usar la pila de urgentes solo para tareas críticas.

CAPÍTULO 4: EVIDENCIAS DE TRABAJO EN EQUIPO

Errores:

```
1 tareas = {}
2 pila_urgentes = []
3 next_id = 1
4
5 def agregar_tarea():
6     global next_id
7
8     print("\n--- AGREGAR TAREA ---")
9     titulo = input("Titulo: ").strip()
10    prioridad = input("Prioridad (alta, media, baja): ").strip().lower()
11
12    if titulo == "":
13        print("Error: el titulo no puede estar vacío.")
14        return
15
16    if prioridad not in ["alta", "media", "baja"]:
17        print("Error: prioridad inválida.")
18        return
19
20    tareas[next_id] = {
21        "titulo": titulo,
22        "estado": "pendiente",
23        "prioridad": prioridad,
24        "urgente": False
25    }
26
27    print(f"Tarea agregada con ID {next_id}.")
28    next_id += 1
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

File "c:\Users\andre\OneDrive\Documentos\VS STUDIO CODE\porgramalistadetares\pa2estructuradados\veripa2.py", line 5
def agregar_tarea()
^
SyntaxError: expected ':'
PS C:\Users\andre\OneDrive\Documentos\VS STUDIO CODE> & C:/Users/andre/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/andre/OneDrive/Documentos/VS STUDIO CODE/porgramalistadetares/pa2estructuradados/veripa2.py"
File "c:\Users\andre\OneDrive\Documentos\VS STUDIO CODE\porgramalistadetares\pa2estructuradados\veripa2.py", line 5
def agregar_tarea()
^
SyntaxError: expected ':'
PS C:\Users\andre\OneDrive\Documentos\VS STUDIO CODE> & C:/Users/andre/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/andre/OneDrive/Documentos/VS STUDIO CODE/porgramalistadetares/pa2estructuradados/veripa2.py"
File "c:\Users\andre\OneDrive\Documentos\VS STUDIO CODE\porgramalistadetares\pa2estructuradados\veripa2.py", line 5
def agregar_tarea()
^
SyntaxError: expected ':'
PS C:\Users\andre\OneDrive\Documentos\VS STUDIO CODE>

```
1 tareas = {}
2 pila_urgentes = []
3 next_id = 1
4
5 def agregar_tarea():
6     global next_id
7
8     print("\n--- AGREGAR TAREA ---")
9     titulo = input("Titulo: ").strip()
10    prioridad = input("Prioridad (alta, media, baja): ").strip().lower()
11
12    if titulo == "":
13        print("Error: el titulo no puede estar vacío.")
14        return
15
16    if prioridad not in ["alta", "media", "baja"]:
17        print("Error: prioridad inválida.")
18        return
19
20    tareas[next_id] = {
21        "titulo": titulo,
22        "estado": "pendiente",
23        "prioridad": prioridad,
24        "urgente": False
25    }
26
27    print(f"Tarea agregada con ID {next_id}.")
28    next_id += 1
29
30
31 def listar_tareas():
32     print("\n--- LISTA DE TAREAS ---")
33
34     if not tareas:
35         print("No hay tareas registradas.")
36         return
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

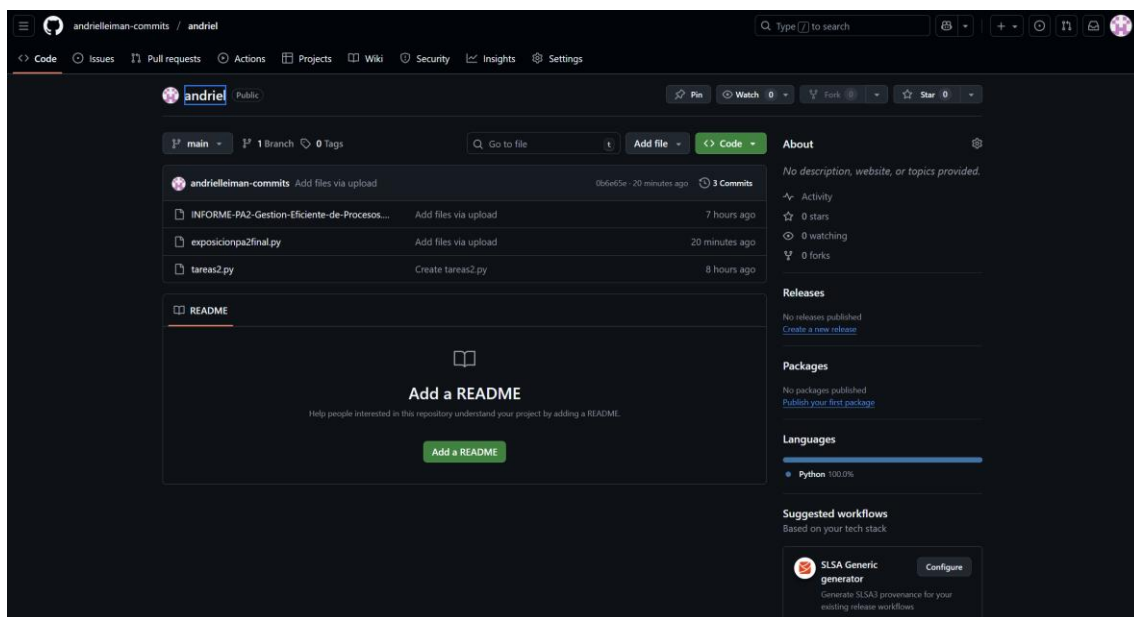
PS C:\Users\andre\OneDrive\Documentos\VS STUDIO CODE> & C:/Users/andre/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/andre/OneDrive/Documentos/VS STUDIO CODE/porgramalistadetares/pa2estructuradados/veripa2.py"
File "c:\Users\andre\OneDrive\Documentos\VS STUDIO CODE\porgramalistadetares\pa2estructuradados\veripa2.py", line 22
"estado": "pendiente",
^
SyntaxError: unterminated string literal (detected at line 22)
PS C:\Users\andre\OneDrive\Documentos\VS STUDIO CODE>

```
121 def menu():
122     registrar_uno_nuevo()
123
124     elif opcion == "2":
125         listar_tareas()
126
127     elif opcion == "3":
128         cambiar_estado()
129
130     elif opcion == "4":
131         marcar_urgente() Expected indented block
132
133     elif opcion == "5": Expected expression
134         sacar_urgente() Unexpected indentation
135
136     elif opcion == "6": Expected expression
137         generar_tabla() Unexpected indentation
138
139     elif opcion == "0": Expected expression
140         print("Saliendo del programa...") Unexpected indentation
141         break
142
143     else: Expected expression
144         print("Opción inválida.") Unexpected indentation
145
146
147 menu()
```

PROBLEMAS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python +v

IndentationError: expected an indented block after 'elif' statement on line 130
PS C:\Users\andre\OneDrive\Documentos\VS STUDIO CODE> & C:/Users/andre/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/andre/OneDrive/Documentos/VS STUDIO CODE/pogramalistadetaareas/pa2estructuradedatos/eripa2.py"
File "c:/Users/andre/OneDrive/Documentos/VS STUDIO CODE/pogramalistadetaareas/pa2estructuradedatos/eripa2.py", line 131
marcar_urgente()
~~~~~  
IndentationError: expected an indented block after 'elif' statement on line 130  
PS C:\Users\andre\OneDrive\Documentos\VS STUDIO CODE> & C:/Users/andre/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/andre/OneDrive/Documentos/VS STUDIO CODE/pogramalistadetaareas/pa2estructuradedatos/eripa2.py"  
File "c:/Users/andre/OneDrive/Documentos/VS STUDIO CODE/pogramalistadetaareas/pa2estructuradedatos/eripa2.py", line 131  
marcar\_urgente()  
~~~~~  
IndentationError: expected an indented block after 'elif' statement on line 130
PS C:\Users\andre\OneDrive\Documentos\VS STUDIO CODE>]

- Registro de commits claros y significativos que evidencien aportes individuales (proactividad).



Enlace:
<https://github.com/andrielleiman-commits/andriel>