Actividad 8: Gestión de Datos con Diccionarios y Menú Interactivo

Nombre: Alejandro Ramirez Cruz

Matricula: 379551

Fecha: Lunes 6 octubre de 2025

Ejercicio 1: Menú Interactivo para Gestión de Trabajadores de una Frabrica

Descripción:

Sistema de gestión de empleados que permite administrar una lista de trabajadores mediante un menú interactivo, con opciones para agregar, buscar, ordenar, eliminar y visualizar empleados en diferentes formatos.

Explicación del problema:

Desarrollar un programa que maneje información de empleados de forma eficiente, garantizando la integridad de los datos mediante validaciones. El sistema debe permitir tanto la entrada manual como automática de datos, asegurando que cada empleado tenga un ID único. Además, debe ofrecer diferentes formas de visualizar la información y operaciones básicas de gestión como ordenamiento y eliminación.

Solución:

Se implementó una solución diviendo el programa en bloques de funciones y un menu princiapl.

1. Uso de librerias como pandas, yaml, json y random para sus respectivas tareas como impresion y generacion de numeros aleatorios.
2. Validaciones robustas para todos los campos (enteros, cadenas, edades, sexo, IDs únicos)
3. Generación automática de datos usando listas predefinidas y selección aleatoria
4. Múltiples formatos de salida (Pandas, YAML, JSON) seleccionados aleatoriamente
5. Gestión de lista con funciones para ordenar, buscar y eliminar elementos
6. Menú interactivo con control de flujo que mantiene la lista en memoria durante la sesión
7. Manejo de errores para entradas inválidas y confirmaciones para operaciones destructivas

El programa utiliza diccionarios para almacenar cada trabajador y listas para mantener la colección completa, con funciones especializadas para cada operación del menú.

```
!pip install pyyaml
!pip install pandas
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages (6.0.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17
```

```
# IMPORTAR LIBRERIAS QUE SE USARAN PARA GENERAR NUMEROS ALEATORIOS Y PARA IMPRIMIR
import random
import pandas as pd
import yaml
import json
```

```
# VALIDACIÓN DE DATOS Y ENTRADAS DEL USUARIO

def verificar_lista_vacia(lista):    # Verifica si la lista esta vacia  o con datos.
    if not lista:
        print("La lista de trabajadores está vacia.\n")
        return True
    return False
```

```
def validar_entero(mensaje):    # Valida la entrada de numeros enteros, en campos como ID y sueldo
    while True:
        try:
            entero = int(input(mensaje))
            return entero
```

```
        except ValueError:
            print("Error: Debe ingresar un número entero...")
```

```
def validar_cadena(mensaje):   # Valida que se ingresa una cadena, o que esta no tenga número o este vacia
    while True:
        try:
            cadena = input(mensaje).strip()
            if cadena:
                return cadena.upper()   # Regrese la cadena convertida a mayúsculas
            else:
                print("No se ha ingresado nada...")
        except ValueError:
            print("Error: No se permiten números...")
```

```
def validar_edad(mensaje):    # Valida que la edad del trabajador sea mayor o igual a 18 años según lo establecido por la
    while True:
        try:
            edad = int(input(mensaje))
            if edad >= 18:
                return edad
            else:
                print("La edad debe ser mayor o igual a 18...")
        except ValueError:
            print("Error: Debe ingresar un número entero")
```

```
def validar_sexo(mensaje):    # Valida que el usario ingrese bien su tipo de sexo.
     while True:
        sexo = input(mensaje).strip().upper()    # Se usa upper() para convertir la cadena a mayúsculas
        if sexo in ['H', 'M']:
            return sexo
        print("Error: El sexo debe ser 'H' (Hombre) o 'M' (Mujer).")
```

```
def validar_id_unico(mensaje, lista_trabajadores):  # Valida que el id ingresado sea unico y no repetido
    while True:
        id_empleado = validar_entero(mensaje)
        # Extrae todos los IDs existentes para verificar duplicados
        ids_existentes = [trabajador['No. Empleado'] for trabajador in lista_trabajadores]

        if id_empleado in ids_existentes:
            print("Error: Este número de empleado ya existe. Ingrese uno diferente.")
        else:
            return id_empleado  # Regresa el ID unico
```

```
# FUNCION GENERACIÓN Y AGREGACIÓN DE TRABAJADORES

def generar_datos_automaticos():
    nom_hombres = ["GOKU", "VEGETA", "GOHAN", "TRUNKS", "KRILIN", "PICCOLO", "TEN SHIN HAN", "YAMCHA", "ROSHI", "MR. SATAN"
    nom_mujeres = ["CHI-CHI", "BULMA", "VIDEL", "PAN", "ANDROID 18", "ANDROID 21", "CAULIFLA", "KALE", "MAI", "LAUNCH", "MA
    apellidos = ["GOMEZ", "HERNANDEZ", "LOPEZ", "PEREZ", "GARCÍA", "RODRIGUEZ", "MARTINEZ", "SANCHEZ", "DIAZ", "RAMIREZ", "
    puestos = ["GERENTE DE PRODUCCION","TECNICO DE ENSAMBLAJE", "OPERARIO DE PRODUCCION", "INGENIERO DE CALIDAD", "TECNICO
               "OPERADOR DE MAQUINAS", "TECNICO EN MANTENIMIENTO","ANALISTA DE INVENTARIOS","INGENIERO DE DISEÑO", "JEFE DE
               "PERSONAL DE LIMPIEZA","TECNICO EN SISTEMAS", "GERENTE DE CALIDAD"]
    sueldos = ["$1500","$2000","$2500","$3000","$3500","$4000","$4500","$5000","$5500","$6000","$6500","$9500","$10000"]
    # Estructura de datos del trabajador
    claves = ["No. Empleado", "Nombre", "A. Paterno", "A. Materno", "Edad", "Sexo", "Puesto", "Sueldo"]

    sexo = random.choice(['H','M'])
    if sexo == 'H':       # Si sexo es igual a H agrega nombre de hombre, y viceversa con M
        nombre = random.choice(nom_hombres)
    else:
        nombre = random.choice(nom_mujeres)

    valores = [
        random.randint(1000, 9999),      # ID único dentro del rango
        nombre,                          # Nombre según sexo
        random.choice(apellidos),        # Apellido paterno
        random.choice(apellidos),        # Apellido materno
        random.randint(18, 80),          # Edad entre 18 y 80 años
        sexo,                            # Sexo (H o M)
        random.choice(puestos),          # Puesto asignado aleatoriamente
        random.choice(sueldos)           # Sueldo como cadena
    ]

    # Combina claves y valores para crear el diccionario
    trabajador = dict(zip(claves, valores))
    return trabajador
```

```python
# FUNCION PARA AGREGAR DATOS AUTOMATICOS

def agregar_automatico(lista_trabajadores):  # Permite agregar varios trabajadores, con IDS unicos que no se repiten en la
    print("AGREGAR TRABAJADORES AUTOMÁTICAMENTE\n")
    cantidad = validar_entero("¿Cuántos trabajadores desea agregar?: ")

    for i in range(cantidad):
        # Genera los datos automáticos
        nuevo_trabajador = generar_datos_automaticos()

        # Verificar que el ID no exista en la lista actual
        ids_existentes = [trabajador['No. Empleado'] for trabajador in lista_trabajadores]
        while nuevo_trabajador['No. Empleado'] in ids_existentes:
            nuevo_trabajador = generar_datos_automaticos()

        lista_trabajadores.append(nuevo_trabajador)

    print(f"Se agregaron {cantidad} trabajadores automáticamente\n")
    return lista_trabajadores
```

```python
# FUNCION AGREGAR DATOS MANUALMENTE

def agregar_manual(lista_trabajadores):  # Solicita los datos de cada campo al usario para añadirlo a la lista de trabajado
    print("AGREGAR TRABAJADOR DE FORMA MANUAL\n")

    claves = ["No. Empleado", "Nombre", "A. Paterno", "A. Materno", "Edad", "Sexo", "Puesto", "Sueldo"]

    # Solicita y valida cada campo individualmente
    id_empleado = validar_id_unico("Ingrese el número de empleado: ", lista_trabajadores)
    nombre = validar_cadena("Ingrese nombre: ")
    apellido_paterno = validar_cadena("Ingrese apellido paterno: ")
    apellido_materno = validar_cadena("Ingrese apellido materno: ")
    edad = validar_edad("Ingrese edad: ")
    sexo = validar_sexo("Ingrese sexo (H/M): ")
    puesto = validar_cadena("Ingrese puesto: ")
    sueldo = validar_entero("Ingrese sueldo: $")

    valores = [id_empleado, nombre, apellido_paterno, apellido_materno, edad, sexo, puesto, f"${sueldo}"]
    # Crea el diccionario del trabajador y lo agrega a la lista
    trabajador = dict(zip(claves, valores))
    lista_trabajadores.append(trabajador)

    print(f"Trabajador {nombre} agregado correctamente.")
    return lista_trabajadores
```

```python
# FUNCIONES PARA IMPRIMIR EN 3 DIFERENTES FORMATOS

def imprimir_pandas(lista_trabajadores):  # Imprime la lista en formato Pandas
    # Verifica si la lista esta vacia
    if verificar_lista_vacia(lista_trabajadores):
        return

    print("\n--- LISTA DE TRABAJADORES (PANDAS) ---")
    imp_panda = pd.DataFrame(lista_trabajadores)
    print(imp_panda)
```

```python
def imprimir_yaml(lista_trabajadores):   # Imprime la lista en formato Yaml
    # Verifica si la lista esta vacia
    if verificar_lista_vacia(lista_trabajadores):
        return

    print("\n--- LISTA DE TRABAJADORES (YAML) ---")
    print(yaml.dump(lista_trabajadores, sort_keys=False, default_flow_style=False))
```

```python
def imprimir_json(lista_trabajadores):   # Imprime la lista en formato Json
    # Verifica si la lista esta vacia
    if verificar_lista_vacia(lista_trabajadores):
        return

    print("\n--- LISTA DE TRABAJADORES (JSON) ---")
    print(json.dumps(lista_trabajadores, sort_keys=False, indent=2))
```

```python
def imprimir_lista(lista_trabajadores):  # Imprime aleatoriamente la lista
    # Verifica si la lista esta vacia
    if verificar_lista_vacia(lista_trabajadores):
        return

    metodos = [imprimir_pandas, imprimir_yaml, imprimir_json]
```

```
    imprimir = random.choice(metodos)  # Selección aleatoria del formato
    imprimir(lista_trabajadores)
```

```
# FUNCION PARA BUSCAR ID

def buscar_id_trabajador(lista_trabajadores):  # Permite buscar un trabajador por su número de empleado.
    if verificar_lista_vacia(lista_trabajadores):
        return

    id_buscar = validar_entero("Ingrese el ID a buscar: ")

    # Recorre la lista buscando el ID
    for trabajador in lista_trabajadores:
        if trabajador['No. Empleado'] == id_buscar:
            print("TRABAJADOR ENCONTRADO")
            for clave, valor in trabajador.items():
                print(f"{clave}: {valor}")
                return

    print(f" No se encontró ningún trabajador con ID: {id_buscar}")
```

```
# FUNCION PARA ORDENAR LISTA

def ordenar_lista(lista_trabajadores):  # Ordena la lista de trabajadores por su 'No. Empleado
    if verificar_lista_vacia(lista_trabajadores):
        return lista_trabajadores

    ids = [t['No. Empleado'] for t in lista_trabajadores]

    if ids == sorted(ids):   # Verifica si ya está ordenada antes de realizar el cambio.
        print("La lista ya está ordenada por ID.")
        return lista_trabajadores

    # Ordena usando lambda para acceder al campo 'No. Empleado'
    lista_trabajadores.sort(key=lambda x: x['No. Empleado'])
    print("Lista ordenada por ID.")
    return lista_trabajadores
```

```
# FUNCION PARA ELIMINAR ID

def eliminar_id(lista_trabajadores):  # Elimina un trabajador de la lista por su número de empleado.

    if verificar_lista_vacia(lista_trabajadores):
        return lista_trabajadores

    id_eliminar = validar_entero("Ingrese el ID del trabajador a eliminar: ")

    # Busca el trabajador por ID
    for i, trabajador in enumerate(lista_trabajadores):
        if trabajador['No. Empleado'] == id_eliminar:
            confirmar = input(f"¿Seguro desea eliminar a {trabajador['Nombre']}? (s/n): ").lower()
            if confirmar == 's':
                lista_trabajadores.pop(i) # Elimina por índice
                print(f"Trabajador con ID {id_eliminar} eliminado correctamente.")
            else:
                print("Eliminación cancelada.")
            return lista_trabajadores

    print("No se encontró ningún trabajador con ese ID.")
    return lista_trabajadores
```

```
# FUNCION PARA BORRAR TODA LA LISTA

def borrar_toda_lista(lista_trabajadores):  # Elimina todos los trabajadores de la lista si lo decide el usuario

    if verificar_lista_vacia(lista_trabajadores):
        return lista_trabajadores

    confirmar = input("¿Está seguro de borrar toda la lista? (s/n): ").lower()
    if confirmar == 's':
        lista_trabajadores.clear()  # Limpia toda la lista
        print("Toda la lista ha sido eliminada.")
    else:
        print("Operación cancelada.")
    return lista_trabajadores
```

```
# FUNCION PARA MOSTRAR MENU

```

```python
def mostrar_menu():  # Menu del programa
    lista_trabajadores = []

    while True:
        print("\n=== MENÚ DEL SISTEMA DE GESTIÓN DE EMPLEADOS ===")
        print("1) Agregar (automático)")
        print("2) Agregar (manual)")
        print("3) Imprimir lista")
        print("4) Buscar {ID}")
        print("5) Ordenar")
        print("6) Eliminar {ID}")
        print("7) Borrar toda la lista")
        print("0) Salir")

        opcion = validar_entero("Seleccione una opción: ")

        if opcion == 0:
            print("Programa finalizado.")
            break
        elif opcion == 1:
            lista_trabajadores = agregar_automatico(lista_trabajadores)
        elif opcion == 2:
            lista_trabajadores = agregar_manual(lista_trabajadores)
        elif opcion == 3:
            imprimir_lista(lista_trabajadores)
        elif opcion == 4:
            buscar_id_trabajador(lista_trabajadores)
        elif opcion == 5:
            lista_trabajadores = ordenar_lista(lista_trabajadores)
        elif opcion == 6:
            lista_trabajadores = eliminar_id(lista_trabajadores)
        elif opcion == 7:
            lista_trabajadores = borrar_toda_lista(lista_trabajadores)
        else:
            print("Opción no válida. Intente nuevamente.")
```

```python
# EJECUCIÓN DEL CODIGO
if __name__ == "__main__":
    mostrar_menu()
```

```
=== MENÚ DEL SISTEMA DE GESTIÓN DE EMPLEADOS ===
1) Agregar (automático)
2) Agregar (manual)
3) Imprimir lista
4) Buscar {ID}
5) Ordenar
6) Eliminar {ID}
7) Borrar toda la lista
0) Salir
Seleccione una opción: 7
¿Está seguro de borrar toda la lista? (s/n): s
Toda la lista ha sido eliminada
```