UNIVERSIDAD
DEL QUINDÍO

Laboratorio 1: **Mini Sistema de Informacion para EPS**

Integrantes:
Miguel Angel Gutierrez
Juan Manuel Vasquez Nieto
Santiago Noriega Cifuentes

RESUMEN

En esta práctica se diseñó un Mini Sistema de Información para una EPS y las estructuras básicas vistas en clase. El sistema funciona en consola y permite registrar afiliados con validación de datos, listarlos por apellidos, buscarlos por número de identificación y consultar estadísticas rápidas como el total de afiliados por plan, el promedio de edad por género y la identificación del más joven y el más veterano. Además, los datos se pueden exportar a un archivo CSV para asegurar su persistencia.

La práctica permitió aplicar de manera práctica el uso de condicionales, ciclos, funciones, listas, diccionarios y pruebas unitarias. El resultado fue un programa funcional que cumple con los requerimientos establecidos y demuestra cómo se puede organizar y manipular información de manera estructurada. Este ejercicio sirve como base para proyectos más avanzados de gestión de datos.

PALABRAS CLAVE

- Afiliados
- Listas y diccionarios
- Validación de datos.
- CSV
- lambda

1. INTRODUCCIÓN

El sistema de salud colombiano depende de la adecuada gestión de información de los millones de afiliados que hacen parte de las EPS (Entidades Promotoras de Salud). Los datos deben mantenerse únicos, organizados y disponibles, pues de ellos dependen procesos como la asignación de citas, el cálculo de copagos, la autorización de procedimientos y la generación de reportes regulatorios. En este contexto, la práctica se orientó a resolver el problema de construir un sistema básico de administración de afiliados en un entorno de programación estructurada.

UNIVERSIDAD DEL QUINDÍO

El objetivo del laboratorio fue crear un **Mini EPS**, que integrará funciones clave: registro de afiliados, listado y búsqueda, estadísticas rápidas, exportación de datos y encuesta de satisfacción.

El ejercicio permitió aplicar estructuras de programación fundamentales, validación de datos de entrada y almacenamiento persistente en archivos CSV. Con ello, se buscó no solo afianzar los conocimientos teóricos, sino también producir un sistema aplicable en la práctica del sector salud.

Técnicamente, en la práctica el uso de programación estructurada, con énfasis en estructuras de datos como listas, diccionarios y conjuntos, el manejo de ciclos y condicionales, y el uso de pruebas unitarias.[1]

2. REQUERIMIENTOS

1. **Interfaz en consola:**
   El programa debía ejecutarse desde la terminal, mostrando un menú con opciones numeradas que permitieran al usuario interactuar con las funciones disponibles.

2. **Registro de afiliados:**

   - Solicitar al usuario la información básica: número de identificación, nombres, apellidos, fecha de nacimiento, plan, género y correo.
   - Validar que el número de identificación tenga exactamente 10 dígitos y que sea único.
   - Validar que la fecha de nacimiento tenga el formato correcto (dd/mm/aaaa).
   - Ingresar el plan únicamente con las opciones mostradas **A, B** o **C**.
   - Validar que el género sea **M** (masculino) o **F** (femenino).

3. **Almacenamiento temporal de datos:**
   Los afiliados deben guardarse en una lista de diccionarios, mientras que los números de identificación registrados deben almacenarse en un conjunto (set) para garantizar unicidad.

4. **Listar afiliados:**
   Mostrar en pantalla todos los registros existentes, ordenados alfabéticamente por sus

UNIVERSIDAD DEL QUINDÍO

apellidos.

5. **Búsqueda de afiliados:**
   Permitir al usuario consultar la información completa de un afiliado a partir de su número de identificación.

6. **Estadísticas rápidas:**
   - Calcular el total de afiliados por plan.
   - Calcular el promedio de edad por género.
   - Determinar la edad del afiliado más joven y del más veterano.

7. **Exportar datos:**
   Guardar la información en un archivo **`afiliados.csv`**, mostrando el contenido con los datos actualizados al momento de exportar.

8. **Finalización del programa:**
   Al seleccionar la opción “Salir”, exportar automáticamente los datos y mostrar un mensaje indicando el total de afiliados almacenados.

3. PROCEDIMIENTO

1. **Validación de identificación:**

   Se creó una función para leer el número de identificación y asegurar que fuera numérico, de 10 dígitos y sin repeticiones. Si el dato no cumple, el programa vuelve a pedirlo.

```
# Identificacion
def leer_identificacion(mensaje):
    while True:
        valor = input(mensaje).strip()
        if valor.lower() == "salir":
            return None
        if not valor.isdigit():
            print("Debes ingresar solo numeros.")
            continue
```

UNIVERSIDAD DEL QUINDÍO

```
        if len(valor) != 10:
            print("La identificación debe tener exactamente 10 dígitos.")
            continue
        return valor
```

2. **Validación de fecha:**

   Se implementó una función para leer la fecha de nacimiento y verificar que esté escrita en el formato correcto **dd/mm/aaaa**. Si el formato no es válido, se solicita nuevamente.

```
# Validar Fecha

def leer_fecha(mensaje):

    while True:

        valor = input(mensaje).strip()

        if valor.lower() == "salir":

            return None

        try:

            return datetime.strptime(valor, "%d/%m/%Y").date()

        except ValueError:

            print("Formato invalido. Usar dd/mm/aaaa.")
```

3. **Validacion de plan de afiliacion:**

   Este programa solo acepta una de las tres opciones.

   - Pide al usuario que ingrese el plan.

UNIVERSIDAD DEL QUINDÍO

- Convierte lo que escribe a mayúscula y elimina espacios.
- Si el valor ingresado es **A**, **B** o **C**, lo acepta y lo devuelve.
- Si el valor no corresponde, muestra el mensaje “ Debe ser A, B o C ”y vuelve a pedir la entrada hasta que sea correcta.

```
# Plan Afiliados

def leer_plan(mensaje):

    while True:

        valor = input(mensaje).upper().strip()

        if valor.lower() == "salir":

            return None

        if valor in ["A", "B", "C"]:

            return valor

        print("Debe ser A, B o C.")
```

4. **Validación de género:**

   Selección de género entre M y F:

- El usuario escribe el género.
- El programa convierte la entrada a mayúscula y quita espacios.
- Si la opción ingresada es **M** o **F**, la acepta y la devuelve.

UNIVERSIDAD DEL QUINDÍO

- Si no corresponde, muestra el mensaje “ El género debe ser M o F ” y vuelve a pedir el dato hasta que sea válido.

De esta manera, el sistema asegura que solo se registren géneros válidos y evita inconsistencias en la información.

```python
# Validar genero

def leer_genero(mensaje):

    while True:

        valor = input(mensaje).upper().strip()

        if valor.lower() == "salir":

            return None

        if valor in ["M", "F"]:

            return valor

        print("El genero debe ser M o F.")
```

5. **Validación del cálculo de la edad:**

Se desarrolló una función para calcular la edad a partir de la fecha de nacimiento.

El cálculo consiste en restar el año de nacimiento al año actual y ajustar según si el afiliado ya cumplió años en el presente año o no.

UNIVERSIDAD
DEL QUINDÍO

```
# Validar Edad

def calcular_edad(fecha_nacimiento):

    hoy = date.today()

    return hoy.year - fecha_nacimiento.year - (

        (hoy.month, hoy.day) < (fecha_nacimiento.month,
fecha_nacimiento.day)

    )
```

6. **Registro de afiliados:**

Se solicitó toda la información al usuario, validando los datos, y se almacenó en un diccionario dentro de la lista de afiliados**.**

```
# Registro de Afiliados
def registrar_afiliado():
    print("\nRegistro de Afiliado")
    id_afiliado = leer_identificacion("Numero de identificacion: ")
    if id_afiliado is None:
        print("Registro cancelado.")
        return
    if id_afiliado in ids:
        print("Este numero de identificación ya esta registrado.")
        return

    nombres = input("Nombres: ").strip()
    apellidos = input("Apellidos: ").strip()

    fecha_nac = leer_fecha("Fecha de nacimiento (dd/mm/aaaa): ")
    if fecha_nac is None:
        print("Registro cancelado.")
        return
```

UNIVERSIDAD DEL QUINDÍO

```
    plan = leer_plan("Plan de afiliación (A | B | C): ")
    if plan is None:
        print("Registro cancelado.")
        return

    genero = leer_genero("Género ( M / F ): ")
    if genero is None:
        print("Registro cancelado.")
        return

    correo = input("Correo electronico: ").strip()

    afiliado = {
        "id": id_afiliado,
        "nombres": nombres,
        "apellidos": apellidos,
        "fecha_nacimiento": fecha_nac,
        "plan": plan,
        "genero": genero,
        "correo": correo,
    }
    afiliados.append(afiliado)
    ids.add(id_afiliado)
    print("Afiliado registrado correctamente.")
```

7. **Listado de afiliados:**

   El listado de afiliados se hizo mostrando en pantalla todos los registros guardados. Para que aparezcan ordenados, el programa usa el apellido como criterio y después imprime el ID, el nombre completo y el plan de cada afiliado

```
# Listar Afiliados
def listar_afiliados():
    print("\n Lista de Afiliados")
```

UNIVERSIDAD DEL QUINDÍO

```
    for a in sorted(afiliados, key=lambda x: x["apellidos"]):
        print(f"{a['id']} - {a['nombres']} {a['apellidos']} - Plan
{a['plan']}")
```

8. **Búsqueda de afiliados:**

- El programa muestra la opción de **buscar afiliado** en el menú principal.
- Se solicita al usuario ingresar el **número de identificación**.
- El sistema recorre la lista de afiliados y compara el número ingresado con los registrados.
    - Si encuentra coincidencia, muestra todos los datos del afiliado (nombre, apellidos, plan, género, correo, etc.).
    - Si no lo encuentra, muestra el mensaje **"Afiliado no encontrado"**.

```
# Buscar Afiliados
def buscar_afiliado():
    print("\n Buscar Afiliado")
    id_buscar = leer_identificacion("Ingrese numero de identificacion: ")
    if id_buscar is None:
        print("Busqueda cancelada.")
        return
    for a in afiliados:
        if a["id"] == id_buscar:
            print(a)
            return
    print("Afiliado no encontrado.")
```

9. **Estadísticas:**

UNIVERSIDAD DEL QUINDÍO

- El sistema recorre la lista de afiliados registrados.
- Cuenta cuántos afiliados hay en cada plan **(A, B o C)**.
- Calcula la edad de cada afiliado a partir de su fecha de nacimiento.
- Con esas edades, obtiene el promedio por género **(M y F)**.
- Identifica y muestra al afiliado más joven y al afiliado de mayor edad.
- Finalmente, presenta los resultados en consola de forma resumida.

```
# Estadisticas Rapidas
def estadisticas():
    print("\n Estadisticas Rapidas ")

    conteo_planes = {"A": 0, "B": 0, "C": 0}
    for a in afiliados:
        conteo_planes[a["plan"]] += 1
    print("Afiliados por plan:", conteo_planes)

    edades_genero = {"M": [], "F": []}
    for a in afiliados:
        edad = calcular_edad(a["fecha_nacimiento"])
        if a["genero"].upper() in edades_genero:
            edades_genero[a["genero"].upper()].append(edad)

    for g, lista in edades_genero.items():
        if lista:
            promedio = sum(lista) / len(lista)
            print(f"Promedio de edad genero {g}: {promedio:.1f} años")

    if afiliados:
        edades = [calcular_edad(a["fecha_nacimiento"]) for a in afiliados]
        print(f"El afiliado mas joven: {min(edades)} años")
        print(f"El afiliado mas Mayor/Adulto: {max(edades)} años")
```

UNIVERSIDAD
DEL QUINDÍO

10. Exportacion a csv:

- El sistema abre o crea un archivo llamado **`afiliados.csv`**.
- Escribe en la primera fila los encabezados: ID, Nombres, Apellidos, Fecha de Nacimiento, Plan, Género y Correo.
- Recorre la lista de afiliados guardados en el programa.
- Por cada afiliado, escribe una fila en el archivo con sus datos.
- La fecha de nacimiento se guarda en el formato (dd/mm/aaaa) para mantener uniformidad.
- Al finalizar, el programa informa al usuario que los datos fueron exportados correctamente.

```
# Exportacion a csv
def exportar_csv():
    with open("afiliados.csv", "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["ID", "Nombres", "Apellidos", "FechaNacimiento",
"Plan", "Genero", "Correo"])
        for a in afiliados:
            writer.writerow([
                a["id"], a["nombres"], a["apellidos"],
                a["fecha_nacimiento"].strftime("%d/%m/%Y"),
                a["plan"], a["genero"], a["correo"]
            ])
    print("Datos exportados a afiliados.csv")
```

11. Menu Principal:

- Al iniciar el programa se muestra un menú en consola con las opciones disponibles.

UNIVERSIDAD
DEL QUINDÍO

- El usuario selecciona una opción escribiendo el número correspondiente.

- Según la opción elegida, el sistema ejecuta la función asociada:

  1: Registrar afiliado
  2: Listar afiliados
  3: Buscar afiliado por ID
  4: Estadísticas rápidas
  5: Exportar CSV
  6: Salir del programa (guardando los datos)

- Si el usuario escribe un número inválido, el programa muestra un mensaje de error y vuelve a pedir la opción.

- El menú se repite hasta que el usuario seleccione Salir.

```python
# Menu Principal
def menu():
    while True:
        print("\n=== Menú Principal ===")
        print("1. Registrar afiliado")
        print("2. Listar afiliados")
        print("3. Buscar afiliado por ID")
        print("4. Estadísticas rápidas")
        print("5. Exportar CSV")
        print("6. Salir")

        opcion = input("Seleccione una opción: ").strip()
        if opcion == "1":
            registrar_afiliado()
        elif opcion == "2":
            listar_afiliados()
        elif opcion == "3":
            buscar_afiliado()
        elif opcion == "4":
            estadisticas()
        elif opcion == "5":
            exportar_csv()
```

UNIVERSIDAD DEL QUINDÍO

```
        elif opcion == "6":
            exportar_csv()
            print(f"Programa finalizado. Total afiliados guardados:
{len(afiliados)}")
            break
        else:
            print("Opción no válida.")

menu()
```

- **Pruebas unitarias:**

Para este proyecto se implementaron **pruebas unitarias** con el fin de verificar que cada función del programa cumpliera correctamente su tarea. Las pruebas se aplicaron a funciones como la validación de identificación, lectura de fechas, validación del plan y género, cálculo de edad, registro de afiliados, listado, búsqueda, estadísticas rápidas y exportación a CSV.[2]

El propósito de estas pruebas fue:

- **Detectar errores temprano:** asegurar que las funciones individuales trabajen bien antes de unirlas en el sistema completo.
- **Asegurar la confiabilidad:** confirmar que el programa maneje correctamente tanto entradas válidas como inválidas.
- **Mantener el código robusto:** si en el futuro se hacen cambios, las pruebas ayudan a comprobar que las funciones sigan funcionando.
- **Ahorrar tiempo en depuración:** identificar rápidamente dónde falla el sistema en caso de error.

En conclusión, las pruebas unitarias garantizan que cada módulo del programa sea estable y confiable antes de integrarse al sistema final.

4. RESULTADOS

UNIVERSIDAD DEL QUINDÍO

Objetivo:
¿Cuál era el resultado esperado?

El resultado esperado consistía en que el programa registrará los afiliados, validará correctamente, listarlos, buscarlos por identificación, mostrar estadísticas rápidas y exportar la información a un archivo csv.

Resultado:
¿Cuál fue el resultado real?

El programa cumplió con lo planteado. Se lograron realizar las validaciones, organizar los afiliados por apellido, calcular promedios de edad y exportar los datos sin inconvenientes.

Acciones:
¿Qué acciones específicas contribuyeron a cumplir con los resultados esperados?

- El código fue dividido en funciones pequeñas e independientes.
- Se aplicaron validaciones muy detalladas para evitar datos repetidos o erróneos.
- Se utilizaron listas y diccionarios para organizar la información.
- Cada parte del programa fue probada individualmente para confirmar su correcto funcionamiento. (Pruebas unitarias).

¿Qué acciones específicas detraen para alcanzar los resultados esperados?

- Se presentaron errores iniciales en la validación de fechas y duplicados.
- Confusión en el cálculo de edades y promedios.
- La exportación de datos a CSV.

Objetivo:
¿Cuál es el resultado esperado a futuro?

UNIVERSIDAD DEL QUINDÍO

A futuro se espera que las habilidades en programación sean fortalecidas, en especial con validaciones, uso de estructuras de datos y organización del código, de manera que puedan desarrollarse sistemas más completos y eficientes.

Estrategia:
¿Qué acciones incrementarán la probabilidad de cumplir con los resultados esperados futuros?

Para aumentar la probabilidad de cumplir con los resultados futuros se plantea lo siguiente:

- Realizar más prácticas en programación para mejorar el uso de validaciones y estructuras de datos.
- Se revisen y corrijan los errores de manera más detallada.
- Trabajar en prácticas más parecidas para ganar más experiencia.
- Mantener el código organizado, detallado (Comentado) y hacer pruebas para comprobar que funciona correctamente.

Referencias Bibliográficas

- Python Software Foundation. (2025). *Python documentation.*[1]
- Barrero, J. D. (2025). *PROGRAMACIÓN ORIENTADA A OBJETOS (POO)*[Universidad del Quindío.[2]