

# **Pontificia Universidad Católica del Ecuador - Sede Ambato**



**Carrera:** Ingeniería en Sistemas de la Información

**Asignatura:** Programación VI

**Docente:** Ing. Edison Meneses Mg.

**Estudiante:** Joaquín Bermeo

**Tema:** Programación Orientada a Objetos – Sistema de Consultas  
Médicas

**Fecha de Entrega:** 01/05/2025

## **Introducción:**

### **1. Objetivo:**

Este proyecto tiene como objetivo diseñar un sistema que permita facilitar el registro de pacientes y consultas médicas usando la "Programación Orientada a Objetos (POO)", dicha información y datos podrán ser consultados dentro del sistema ya que se irán almacenando. Este sistema fue diseñado para permitir que el guardado y manejo de información sea de manera ordenada y se pueda acceder fácilmente a la misma gracias a búsqueda individual o completa de pacientes.

---

### **2. Contexto:**

El sistema fue diseñado de manera modular, es decir, se usaron varios archivos que trabajan en conjunto para que el programa funcione, esto permite que la organización y optimización del código se lleve de manera adecuada y sea más entendible a diferencia de manejar un solo archivo "main". El proyecto lo que busca es utilizar los conceptos básicos y fundamentales de la Programación Orientada a Objetos como el uso de clases, objetos, métodos/funciones, etc., para lograr cumplir el objetivo principal.

## **Desarrollo:**

### **1. Estructura del proyecto:**

```
consultas_medicas/  
├── main.py  
├── paciente.py  
└── funciones.py
```

#### **Archivo “main.py”:**

Este archivo contiene el menú principal y la lógica que gestiona la interacción con el usuario. El programa se mantiene activo mediante el uso de un ciclo `*while*`, lo que permite que el usuario interactue entre las distintas opciones que se manejan dentro del sistema sin la necesidad de ejecutarlo siempre.

### Archivo “paciente.py”:

Aquí se define la clase “Paciente” la cual contiene los atributos y métodos relacionados a la información de un paciente. Los atributos implementados fueron: nombre, cédula, edad, tipo de sangre y consultas (lista de diccionarios), esta última permite que a un paciente se le pueda registrar varias consultas y se almacenen. Por otro lado, se implementaron métodos que trabajan en conjunto con el archivo de funciones “agregar\_consulta”, mostrar\_información y mostrar\_consultas.

### Archivo “funciones.py”:

Contiene funciones auxiliares de los distintos procesos del sistema como registrar pacientes consultas, buscar pacientes por su cédula y mostrar información de los pacientes registrados. Al inicio se importa la clase Paciente del archivo “paciente.py”. Cada una de estas funciones cuentan con sus respectivas validaciones para que el programa se ejecute y utilice sin que haya errores. Además de las funciones, aquí se inicializa la lista que se encarga de ir almacenando todos los pacientes que se vayan registrando “lista\_pacientes”.

---

## 2. Diseño de la clase “Paciente”:

La clase “Paciente” es fundamental dentro del funcionamiento del sistema, ya que representa el objeto con el que interactúa el usuario. Los atributos que contiene la clase son los siguientes:

- **nombre:** Atributo que almacena el nombre del paciente.
- **cédula:** Atributo que almacena la cédula del paciente.
- **edad:** Atributo que almacena la edad del paciente.
- **tipo\_sangre:** Atributo que almacena el tipo de sangre del paciente.
- **lista\_consultas:** Lista de diccionarios vacía que almacena las consultas del paciente.

```
1 # Definición de la clase Paciente
2 class Paciente:
3     # Constructor de la clase, inicializa los atributos del paciente
4     def __init__(self, nombre, cedula, edad, tipo_sangre):
5         self.nombre = nombre          # Atributo que almacena el nombre del paciente
6         self.cedula = cedula          # Atributo que almacena la cédula del paciente
7         self.edad = edad              # Atributo que almacena la edad del paciente
8         self.tipo_sangre = tipo_sangre # Atributo que almacena el tipo de sangre del paciente
9         self.lista_consultas = []     # Inicializa una lista vacía para almacenar las consultas del paciente
```

Los métodos que se implementaron dentro de la clase “Paciente” son:

- **agregar\_consulta:** Método que recibe como parámetros a fecha, diagnostico y tratamiento con los que se crea un diccionario que después es almacenado en la lista “lista\_consultas”.

```
1 # Método para agregar una nueva consulta al historial del paciente
2 def agregar_consulta(self, fecha, diagnostico, tratamiento):
3     consulta = { # Crea un diccionario con los datos de la consulta
4         "fecha": fecha,
5         "diagnostico": diagnostico,
6         "tratamiento": tratamiento
7     }
8     self.lista_consultas.append(consulta) # Agrega la consulta a la lista de consultas del paciente
9
```

- **mostrar\_información:** Método que se encarga de mostrar la información básica del paciente, es decir, su nombre, cédula, edad, y tipo de sangre.

```
1 # Método para mostrar la información básica del paciente
2 def mostrar_informacion(self):
3     print('\n>>> DATOS <<<')
4     # Muestra los datos básicos del paciente (nombre, cédula, edad, tipo de sangre)
5     print(f'Nombre: {self.nombre}\nCédula: {self.cedula}\nEdad: {self.edad}\nTipo de Sangre: {self.tipo_sangre}')
6
```

- **mostrar\_consultas:** Método que muestra el historial de consultas del paciente, es decir, imprime cada uno de los elementos que han sido registrados en la lista “lista\_consultas”, si esta se encuentra vacía se imprime un aviso.

```
1 # Método para mostrar el historial de consultas del paciente
2 def mostrar_consultas(self):
3     contador_mostrar_inf = 1 # Contador para numerar las consultas
4     print("\n>>> Historial de Consultas <<<")
5
6     if self.lista_consultas: # Si el paciente tiene consultas registradas
7         # Itera sobre todas las consultas y las imprime
8         for consulta in self.lista_consultas:
9             print(f'\nConsulta #{contador_mostrar_inf}:')
10            # Muestra la información de la consulta: fecha, diagnóstico y tratamiento
11            print(f'Fecha: {consulta["fecha"]}\nDiagnostico: {consulta["diagnostico"]}\nTratamiento: {consulta["tratamiento"]}')
12            contador_mostrar_inf += 1 # Aumenta el contador de consultas
13        else:
14            print("AVISO! No existen registros.") # Si no hay consultas, muestra un mensaje
15
```

### 3. Funcionamiento del menú y funciones:

#### Menú:

Se creó una función llamada menu() que está dentro del archivo “main”. Dentro de la función se maneja un ciclo “while True:” que permite que el sistema se mantenga activo hasta que el usuario desee salir. Se lee una única entrada de un número entero que es la opción que ingresa el usuario para ir navegando entre las distintas opciones (1. Registrar nuevo paciente 2. Registrar una consulta médica 3. Mostrar datos completos de un paciente 4. Mostrar todos los pacientes registrados 5. Salir), este proceso de selección se lo realizó mediante el uso de if, elif y else donde se compara el valor ingresado por el usuario y permite acceder a las funciones principales del sistema que se exportan del archivo funciones.py. Además, se aplicó la respectiva validación con un try para que el sistema no colapse cuando se ingrese caracteres que no sean números.

```
1 # Se importa las funciones definidas en el archivo funciones.py
2 from funciones import registrar_paciente, registrar_consulta, mostrar_paciente, mostrar_todos_pacientes
3
4 # Función principal del menú
5 def menu():
6     while True:
7         # Muestra el menú de opciones
8         print("\n\t\t << | MENU DE OPCIONES| >>")
9         print("(1) Registrar nuevo paciente\n(2) Registrar una consulta medica\n(3) Mostrar los datos completos del paciente\n(4) Mostrar todos los pacientes registrados\n(5) Salir")
10
11         # Intenta capturar una opción válida (un número entre 1 y 5)
12         try:
13             opcion = int(input("Seleccione una opción (1-5): "))
14         except ValueError:
15             # Si el usuario ingresa algo que no es un número, muestra un mensaje de error y sigue pidiendo una opción válida
16             print("ERROR, opcion no válida. Ingresar un numero del 1 al 5!")
17             continue
18
19         # Opción para registrar un nuevo paciente
20         if opcion == 1:
21             print("\n\t\t| NUEVO PACIENTE |")
22             # Llama a la función registrar_paciente para registrar un nuevo paciente
23             registrar_paciente()
24
25         # Opción para registrar una nueva consulta médica
26         elif opcion == 2:
27             print("\n\t\t| NUEVA CONSULTA |")
28             # Llama a la función registrar_consulta para registrar una consulta médica
29             registrar_consulta()
30
31         # Opción para mostrar los datos completos de un paciente
32         elif opcion == 3:
33             print("\n\t\t| DATOS PACIENTE |")
34             # Llama a la función mostrar_paciente para mostrar los datos de un paciente específico
35             mostrar_paciente()
36
37         # Opción para mostrar todos los pacientes registrados
38         elif opcion == 4:
39             print("\n\t\t| PACIENTES REGISTRADOS |")
40             # Llama a la función mostrar_todos_pacientes para mostrar todos los pacientes
41             mostrar_todos_pacientes()
42
43         # Opción para salir del programa
44         elif opcion == 5:
45             print("\n Gracias por usar el sistema :3")
46             # Salir del bucle infinito, terminando el programa
47             break
48
49         # Si el usuario ingresa una opción fuera del rango 1-5, muestra un mensaje de error
50         else:
51             print("\nERROR, esta no opcion no esta dentro del rango!")
52
53 # Comprobamos si el script está siendo ejecutado directamente y, si es así, se llama al menú
54 if __name__ == "__main__":
55     menu()
56
```

## Funciones:

### 1. registrar\_paciente():

Esta función se encarga de capturar la información necesaria para posteriormente crear el objeto “paciente” utilizando la clase creada en el archivo “paciente.py”. En la misma función se valida que el usuario no pueda repetir la cédula en varios pacientes con la implementación de la función “buscar\_paciente\_cedula(cedula)”, ya que se trata de un identificador único, de igual forma trabaja en conjunto con la función “validar\_cedula”. Por otro lado, la edad es validada para que el usuario no pueda ingresar números negativos y que solo pueda ingresar números enteros. Finalmente, se creó una lista con todos los tipos de sangre que existen lo que permite comparar la entrada del usuario con la lista y de esta manera validar que se ingrese un tipo de sangre real.

```
1 def registrar_paciente():
2     # Lista de tipos de sangre válidos
3     tipos_sangre = ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"]
4
5     # Captura del nombre del paciente
6     nombre = input("Nombre: ")
7     # Captura de la cédula, asegurándose de que no sea repetida
8     cedula = validar_cedula()
9     while buscar_paciente_cedula(cedula) != None: # Verifica que la cédula no esté registrada
10         print("ERROR, cédula repetida!")
11         cedula = validar_cedula()
12
13     # Captura de la edad, validando que sea un número positivo
14     while True:
15         try:
16             edad = int(input("Edad: "))
17             if edad < 0:
18                 print("ERROR, no puede ingresar una edad negativa!")
19                 continue
20             break
21         except ValueError: # Si se ingresa algo que no sea un número, muestra un error
22             print("ERROR, edad no válida!")
23             continue
24
25     # Captura del tipo de sangre, validando que sea uno de los tipos permitidos
26     tipo_sangre = input("Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): ").strip().upper()
27     while tipo_sangre not in tipos_sangre: # Si el tipo de sangre no es válido
28         print("ERROR, tipo de sangre no válido!")
29         tipo_sangre = input("Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): ").strip().upper()
30
31     # Crea un objeto Paciente y lo agrega a la lista de pacientes
32     paciente = Paciente(nombre, cedula, edad, tipo_sangre)
33     lista_pacientes.append(paciente)
34     print(">>> Paciente registrado con éxito :).")
```

### 2. validar\_cedula():

Como su nombre lo indica, dentro de esta función se pide al usuario ingresar la cédula para validar la parte básica, es decir, que no haya letras y sean 10 dígitos. El dato ingresado se lo

mantiene como tipo String, ya que existen números de cédula que empiezan por el número 0 y al momento de transformarlo a entero no se cuenta el número mencionado y se tomaría como inválida a la cédula ingresada.

```
1 # Función para validar la cédula (debe ser un número de 10 dígitos)
2 def validar_cedula():
3     cedula = input("Ingrese la cédula: ")
4     while cedula.isdigit() == False or len(cedula) < 10: # Si la cédula no es válida
5         print("ERROR, cédula no válida!")
6         cedula = input("Ingrese la cédula: ")
7     return cedula
```

### 3. buscar\_paciente\_cedula(cedula):

Esta función se encarga de buscar a un paciente por su número de cédula dentro de la lista que contiene a todos los pacientes registrados, si existe el paciente retorna al mismo sino retorna None. El único parámetro que recibe es la cédula.

```
1 # Función para buscar un paciente por cédula en la lista
2 def buscar_paciente_cedula(cedula):
3     # Recorre la lista de pacientes para encontrar uno con la misma cédula
4     for paciente in lista_pacientes:
5         if paciente.cedula == cedula:
6             return paciente
7     return None # Si no lo encuentra, devuelve None
```

### 4. registrar\_consulta():

En esta función se realiza el proceso de registrar consultas de cada paciente, al inicio se valida que por lo menos exista un paciente registrado para poder continuar, en el caso de que no existan pacientes lo único que hará es mostrar una alerta de que no existen pacientes registrados. Luego de la primera validación se pide al usuario ingresar una cédula utilizando la función “validar\_cedula” y se guarda en una función, después se busca el paciente con la función “buscar\_paciente\_cedula(cedula)” y en el caso de que exista se pasará a la siguiente parte, caso contrario se muestra que no se encontró al paciente. Finalmente, se captura la información para generar el diagnóstico (fecha, diagnóstico y tratamiento). Se utilizó la librería

datetime para que solo se pueda ingresar el formato de fecha DD/MM/AAAA y que tampoco se ingrese fechas futuras o inválidas.

```
1 # Función para registrar una consulta para un paciente
2 def registrar_consulta():
3     if lista_pacientes: # Si hay pacientes registrados
4         cedula = validar_cedula() # Valida la cédula del paciente
5         paciente = buscar_paciente_cedula(cedula) # Busca el paciente en la lista
6         if paciente: # Si el paciente es encontrado
7             # Bucle para validar que la fecha no sea futura
8             while True:
9                 try:
10                     fecha = input("Fecha (DD/MM/AAAA): ")
11                     fecha_valida = datetime.strptime(fecha, "%d/%m/%Y") # Intenta convertir la fecha
12                     if fecha_valida > datetime.today(): # Si la fecha es futura
13                         print("ERROR, La fecha no puede ser futura!")
14                     else:
15                         break # Si la fecha es válida, rompe el bucle
16                 except ValueError: # Si la fecha no es válida, muestra un error
17                     print("ERROR, fecha no válida!")
18
19             # Captura del diagnóstico y tratamiento
20             diagnostico = input("Diagnóstico: ")
21             tratamiento = input("Tratamiento: ")
22
23             # Llama al método para agregar la consulta al historial del paciente
24             paciente.agregar_consulta(fecha, diagnostico, tratamiento)
25             print(">> Consulta registrada correctamente :).")
26         else:
27             print("Paciente no encontrado :)")
28     else:
29         print("No hay pacientes registrados!")
30
```

## 5. mostrar\_paciente():

Al igual que la función anterior se valida que la lista de pacientes no esté vacía, luego se usa “validar\_cedula” para que el usuario ingrese una cédula válida para buscar el paciente con la función “buscar\_paciente\_cedula(cedula)”. Si se encuentra al paciente se muestra la información completa del mismo, es decir, su información básica y todo el historial de consultas utilizando métodos de la clase “Paciente”.

```
1 def mostrar_paciente():
2     if lista_pacientes: # Si hay pacientes registrados
3         cedula = validar_cedula() # Valida la cédula del paciente
4         paciente = buscar_paciente_cedula(cedula) # Busca el paciente en la lista
5         if paciente: # Si el paciente es encontrado
6             paciente.mostrar_informacion() # Muestra la información básica del paciente
7             paciente.mostrar_consultas() # Muestra el historial de consultas
8         else:
9             print("Paciente no encontrado :)")
10    else:
11        print("No hay pacientes registrados!")
12
```



## 6. mostrar\_todos\_pacientes():

Esta función también valida que haya al menos un usuario registrado en la lista “lista\_pacientes”, después se utiliza un for para ir imprimiendo cada uno de los pacientes, pero en este caso solo mostrando su información básica (nombre, cédula, edad y tipo de sangre). Además, se muestra la cantidad total de usuarios registrados y se usa un contador para enumerar a los pacientes y la visualización de la información sea más organizada.

```
1 # Función para mostrar todos los pacientes registrados
2 def mostrar_todos_pacientes():
3     contador = 1 # Inicia el contador para mostrar el número de paciente
4     if lista_pacientes: # Si hay pacientes registrados
5         print(f'>> Número de pacientes registrados: {len(lista_pacientes)}')
6         for paciente in lista_pacientes: # Recorre todos los pacientes
7             print(f'\n| PACIENTE #{contador}|')
8             paciente.mostrar_informacion() # Muestra los datos del paciente
9             contador += 1 # Aumenta el contador para el siguiente paciente
10    else:
11        print("No hay pacientes registrados!") # Si no hay pacientes, muestra un mensaje
12
```

## Capturas de pantalla de la ejecución:

### 1. Menú:

```
<< | MENU DE OPCIONES | >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): a
ERROR, opcion no válida. Ingresar un numero del 1 al 5!

<< | MENU DE OPCIONES | >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 1

| NUEVO PACIENTE |
Nombre: |
```

### 2. Registrar Pacientes:

```
| NUEVO PACIENTE |
Nombre: Joaquín Bermeo
Ingrese la cedula: 123456789a
ERROR, cedula no valida!
Ingrese la cedula: 1804653820
Edad: -1
ERROR, no puede ingresar una edad negativa!
Edad: 18
Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): x
ERROR, tipo de sangre no valido!
Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): A+
>> Paciente registrado con éxito :).
```

```
| NUEVO PACIENTE |
Nombre: Pepito
Ingrese la cedula: 1804653820
ERROR, cédula repetida!
Ingrese la cedula: 1802530582
Edad: 20
Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): b-
>> Paciente registrado con éxito :).
```

### 3. Registrar Consulta:

```
<< | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 2

| NUEVA CONSULTA |
Ingrese la cedula: a
ERROR, cedula no valida!
Ingrese la cedula: 1234567890
Paciente no encontrado :(

<< | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 2

| NUEVA CONSULTA |
Ingrese la cedula: 1804653820
Fecha (DD/MM/AAAA): 12
ERROR, fecha no valida!
Fecha (DD/MM/AAAA): 12/04/2030
ERROR, La fecha no puede ser futura!
Fecha (DD/MM/AAAA): 12/04/2025
Diagnóstico: Gripe
Tratamiento: Pastillas
>> Consulta registrada correctamente :).
```

```
| NUEVA CONSULTA |
Ingrese la cedula: 1804653820
Fecha (DD/MM/AAAA): 20/04/2025
Diagnóstico: Fiebre
Tratamiento: Medicamentos
>> Consulta registrada correctamente :).
```

### 4. Mostrar los datos completos del paciente:

```
<< | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 3

| DATOS PACIENTE |
Ingrese la cedula: a
ERROR, cedula no valida!
Ingrese la cedula: 123456789a
ERROR, cedula no valida!
Ingrese la cedula: 1234567890
Paciente no encontrado :(
```

```
<< | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 3

| DATOS PACIENTE |
Ingrese la cedula: 1804653820

>>> DATOS <<<
Nombre: Joaquin Bermeo
Cedula: 1804653820
Edad: 18
Tipo de Sangre: A+

>>> Historial de Consultas <<<

Consulta #1:
Fecha: 12/04/2025
Diagnostico: Gripe
Tratamiento: Pastillas

Consulta #2:
Fecha: 20/04/2025
Diagnostico: Fiebre
Tratamiento: Medicamentos
```

```

<< | MENU DE OPCIONES | >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 3

| DATOS PACIENTE |
Ingrese la cedula: 1802530582

>>> DATOS <<<
Nombre: Pepito
Cedula: 1802530582
Edad: 20
Tipo de Sangre: B-

>>> Historial de Consultas <<<
AVISO! No existen registros.

```

## 5. Mostrar todos los pacientes registrados:

```

<< | MENU DE OPCIONES | >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 4

| PACIENTES REGISTRADOS |
>> Número de pacientes registrados: 2

| PACIENTE #1 |

>>> DATOS <<<
Nombre: Joaquín Bermeo
Cedula: 1804653820
Edad: 18
Tipo de Sangre: A+

| PACIENTE #2 |

>>> DATOS <<<
Nombre: Pepito
Cedula: 1802530582
Edad: 20
Tipo de Sangre: B-

```

## 6. Salir

```

<< | MENU DE OPCIONES | >>
(1) Registrar nuevo paciente
(2) Registrar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 5

Gracias por usar el sistema :3
PS D:\PUCESA\CUARTO SEMESTRE\Programación IV\GitHub\PROGRAMACION4_GITHUB_3B\Semana4\ProyectoConsultasMedicas\consultas_medicas>

```

## Conclusiones

### ¿Qué aprendí durante el desarrollo del proyecto?

- Este proyecto me permitió aplicar y mejorar los conceptos de la Programación Orientada a Objetos, como el uso de clases, objetos y métodos. Por otro lado, aprendí a estructurar un proyecto de manera modular (separado en varios archivos) y a trabajar de manera mucho más optimizada y organizada.

### ¿Qué partes fueron más difíciles o interesantes?

- Personalmente una de las partes más difíciles e interesantes fue crear la clase Persona y usar de manera adecuada sus métodos, ya que no es igual que utilizar funciones que no se encuentran dentro de una clase, al inicio es complejo adaptarse, pero después se vuelve interesante porque el código se ve mucho más limpio. Otra de las partes más difíciles fue validar de manera adecuada todos los datos, no por complejidad sino

porque se debe ir analizando a detalle que información no debe estar permitida y la que sí.

### **Posibles mejoras o nuevas funcionalidades**

- Almacenar los datos de los pacientes y consultas en un archivo externo como un “.txt” o “.json” para que la información no sea volátil y se pueda tener una “base de datos”.
- Se podría añadir otra opción la cuál permita editar los datos del paciente o directamente eliminar un paciente.
- Validaciones más estrictas de los datos de entrada como por ejemplo que se valide si la cédula es verdadera.