

Estructura del Informe

CAPITULO 1: Análisis del Problema

1. **Descripción del problema:** Las agencias de turismo manejan información de clientes, tours, reservas y pagos. Cuando estos datos se administran de forma manual o dispersa, se producen problemas como duplicidad de registros, pérdida de información, demoras en la atención y dificultades para generar reportes.

El sistema propuesto busca resolver este problema permitiendo registrar, actualizar y consultar información de manera organizada y eficiente mediante estructuras de datos estudiadas en el curso. El sistema debe permitir registrar clientes, administrar tours, crear reservas vinculando un cliente y un tour, registrar pagos y generar reportes generales

2. Requerimientos del sistema

A. Funcionales

- **Gestión de clientes**
 - Agregar y actualizar clientes.
 - Buscar cliente por ID.
 - Eliminar cliente.
 - Listar todos los clientes registrados.
- **Gestión de tours**
 - Agregar y actualizar tours.
 - Buscar tour.
 - Eliminar tour.
 - Listar tours disponibles.
- **Gestión de reservas**
 - Crear reserva.
 - Cambiar estado (Pendiente, Pagado, Cancelado).
 - Listar reservas.
 - Validar que cliente y tour existan antes de reservar.
- **Gestión de pagos**
 - Registrar pago asociado a una reserva.
 - Listar historial de pagos.
- **Reportes**
 - Mostrar reservas por cliente.
 - Mostrar ingresos por tour.
 - Mostrar estados de reserva.

B. No funcionales

- **Usabilidad:** Debe ser fácil de usar mediante menús.
- **Mantenibilidad:** Estructurado mediante módulos independientes.
- **Eficiencia:** Respuesta rápida usando estructuras en memoria.
- **Portabilidad:** Debe ejecutarse en cualquier ambiente Python.
- **Simplicidad:** No requiere bases de datos externas.

3. Estructuras de datos propuestas

Para almacenar la información se utilizarán **diccionarios de Python**, representando entidades:

- Clientes
- Tours
- Reservas
- Pagos

4. Justificación de la elección

Los diccionarios permiten:

- Búsqueda rápida por clave.
- Estructuras simples y flexibles.
- Fácil actualización y eliminación.
- Perfecta representación de entidades tipo registro.
- Permiten simular una base de datos pequeña sin requerir archivos.

Por estas razones, son ideales para un prototipo CRUD modular como el requerido en este proyecto

Capítulo 2: Diseño de la Solución

1. Descripción de estructuras de datos y operaciones: El sistema usará varias estructuras de datos estudiadas en clase, cada una asignada a un módulo específico.

A. Clientes

- Operaciones:
 - Agregar/actualizar cliente
 - buscar cliente
 - eliminar cliente
 - listar clientes

B. Tours

- Operaciones:
 - Agregar/actualizar tour
 - Buscar tours
 - Eliminar tours
 - listar tours

C. Reservas

- Operaciones:
 - Crear reserva
 - Cambiar estado
 - Listar reservas

D. Pagos

- Operaciones:
 - Registrar pago
 - Listar Pago

2. Algoritmos principales:(PSEINT)

a. Pseudocódigo para AGREGAR O ACTUALIZAR cliente

INICIO

```
LEER id_cliente
LEER nombre
LEER dni
LEER correo
```

SI id_cliente existe en clientes

actualizar datos
SINO
 crear nuevo cliente
FIN SI

MOSTRAR "Cliente guardado correctamente"
FIN

b. Pseudocódigo para crear una reserva

INICIO
 LEER id_reserva
 LEER id_cliente
 LEER id_tour

SI cliente NO existe
 MOSTRAR "Cliente no encontrado"
 FIN
FIN SI

SI tour NO existe
 MOSTRAR "Tour no encontrado"
 FIN
FIN SI

reservas[id_reserva] = {
 id_cliente,
 id_tour,
 estado = "Pendiente"
}

MOSTRAR "Reserva registrada"
FIN

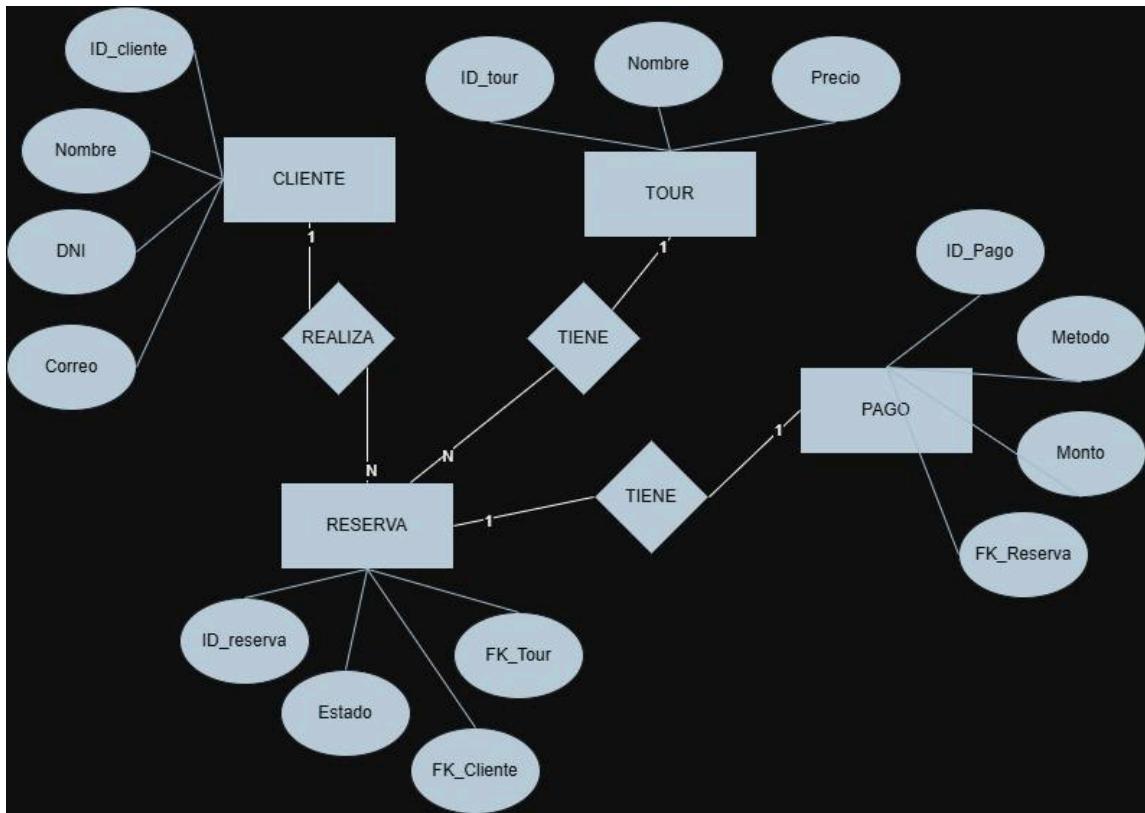
c. Pseudocódigo para CAMBIAR ESTADO de una reserva

INICIO
 LEER id_reserva

SI reserva no existe
 MOSTRAR "No existe"
 FIN
FIN SI

LEER nuevo_estado
SI nuevo_estado ES valido (Pendiente / Pagado / Cancelado)
 actualizar estado
 MOSTRAR "Estado actualizado"
SINO
 MOSTRAR "Estado inválido"
FIN SI
FIN

3. Diagramas de Flujo



4. Justificación del diseño:

- El sistema utiliza módulos independientes para mejorar mantenibilidad.
- El uso de diccionarios permite alta velocidad en operaciones CRUD.
- El menú general facilita la navegación.
- La estructura modular evita errores y hace el código escalable.
- Los algoritmos siguen un flujo simple, con validación en cada operación

Capítulo 3: Solución Final

1. Código limpio, bien comentado y estructurado.

```

# =====
# SISTEMA DE GESTIÓN PARA AGENCIA DE VIAJES
# =====

```

```

clientes = {}
tours = {}
reservas = {}
pagos = {}

# =====
# MÓDULO: CLIENTES
# =====

```

```

def agregar_cliente():
    print("\n--- Registrar Cliente ---")
    cid = input("ID del cliente: ").strip()

    # Validación: evitar duplicados
    if cid in clientes:
        print("ERROR: Ese ID ya existe.")

    # ...

```

```

    return

    nombre = input("Nombre completo: ")
    dni = input("DNI: ")
    correo = input("Correo: ")

    # Guardar datos del cliente
    clientes[cid] = {
        "nombre": nombre,
        "dni": dni,
        "correo": correo
    }
    print("Cliente registrado con éxito.")

def buscar_cliente():
    print("\n--- Buscar Cliente ---")
    cid = input("ID del cliente: ")

    # Verificación de existencia
    if cid in clientes:
        print(clientes[cid])
    else:
        print("No existe el cliente.")

def actualizar_cliente():
    print("\n--- Actualizar Cliente ---")
    cid = input("ID del cliente: ")

    if cid not in clientes:
        print("No existe ese cliente.")
        return

    print("Deje el campo vacío si no desea modificarlo.")
    nombre = input("Nuevo nombre: ")
    correo = input("Nuevo correo: ")

    # Solo actualizar campos llenos
    if nombre.strip():
        clientes[cid]["nombre"] = nombre
    if correo.strip():
        clientes[cid]["correo"] = correo

    print("Cliente actualizado.")

def eliminar_cliente():
    print("\n--- Eliminar Cliente ---")
    cid = input("ID del cliente: ")

```

```

# Eliminar si existe
if cid in clientes:
    del clientes[cid]
    print("Cliente eliminado.")
else:
    print("No existe el cliente.")

def listar_clientes():
    print("\n--- Lista de Clientes ---")

    if not clientes:
        print("No hay clientes registrados.")
        return

    # Mostrar todos los clientes
    for cid, data in clientes.items():
        print(f"{cid}: {data['nombre']} | DNI: {data['dni']} | Correo: {data['correo']}")

# =====
# MÓDULO: TOURS
# =====

def agregar_tour():
    print("\n--- Registrar Tour ---")
    tid = input("ID del tour: ")

    if tid in tours:
        print("Ese tour ya existe.")
        return

    nombre = input("Nombre del tour: ")
    precio = float(input("Precio: "))

    # Guardar tour
    tours[tid] = {"nombre": nombre, "precio": precio}
    print("Tour agregado.")

def buscar_tour():
    print("\n--- Buscar Tour ---")
    tid = input("ID del tour: ")

    if tid in tours:
        print(tours[tid])
    else:
        print("No existe el tour.")

```

```

def actualizar_tour():
    print("\n--- Actualizar Tour ---")
    tid = input("ID del tour: ")

    if tid not in tours:
        print("Ese tour no existe.")
        return

    print("Deje el campo vacío si no desea modificarlo.")
    nombre = input("Nuevo nombre: ")
    precio = input("Nuevo precio: ")

    if nombre.strip():
        tours[tid]["nombre"] = nombre
    if precio.strip():
        tours[tid]["precio"] = float(precio)

    print("Tour actualizado.")


def eliminar_tour():
    print("\n--- Eliminar Tour ---")
    tid = input("ID del tour: ")

    if tid in tours:
        del tours[tid]
        print("Tour eliminado.")
    else:
        print("No existe el tour.")


def listar_tours():
    print("\n--- Lista de Tours ---")

    if not tours:
        print("No hay tours registrados.")
        return

    for tid, data in tours.items():
        print(f"{tid}: {data['nombre']} | Precio: {data['precio']}")

# =====
# MÓDULO: RESERVAS
# =====

def crear_reserva():
    print("\n--- Crear Reserva ---")
    rid = input("ID de la reserva: ")

```

```

if rid in reservas:
    print("Ese ID ya existe.")
    return

cid = input("ID del cliente: ")
tid = input("ID del tour: ")

# Validaciones
if cid not in clientes:
    print("No existe el cliente.")
    return

if tid not in tours:
    print("No existe el tour.")
    return

reservas[rid] = {"cliente": cid, "tour": tid, "estado": "Pendiente"}
print("Reserva creada correctamente.")

def cambiar_estado_reserva():
    print("\n--- Cambiar Estado de Reserva ---")
    rid = input("ID de la reserva: ")

    if rid not in reservas:
        print("No existe esa reserva.")
        return

    print("1) Pagado 2) Cancelado")
    op = input("Nuevo estado: ")

    if op == "1":
        reservas[rid]["estado"] = "Pagado"
    elif op == "2":
        reservas[rid]["estado"] = "Cancelado"
    else:
        print("Opción inválida.")
        return

    print("Estado actualizado.")

def listar_reservas():
    print("\n--- Lista de Reservas ---")

    if not reservas:
        print("No hay reservas.")
        return

```

```

for rid, data in reservas.items():
    print(f"{{rid}}: Cliente {{data['cliente']}} | Tour {{data['tour']}} | Estado: {{data['estado']}}")

# =====
# MÓDULO: PAGOS
# =====

def registrar_pago():
    print("\n--- Registrar Pago ---")
    pid = input("ID del pago: ")

    if pid in pagos:
        print("Ese ID ya existe.")
        return

    rid = input("ID de la reserva: ")

    if rid not in reservas:
        print("Esa reserva no existe.")
        return

    monto = float(input("Monto: "))
    metodo = input("Método de pago: ")

    pagos[pid] = {"reserva": rid, "monto": monto, "metodo": metodo}

    # Actualizar estado de reserva
    reservas[rid]["estado"] = "Pagado"

    print("Pago registrado.")

def listar_pagos():
    print("\n--- Lista de Pagos ---")

    if not pagos:
        print("No hay pagos registrados.")
        return

    for pid, data in pagos.items():
        print(f"{{pid}}: Reserva {{data['reserva']}} | Monto: {{data['monto']}} | Método: {{data['metodo']}}")

# =====
# MENÚS
# =====

def menu_clientes():

```

```

while True:
    print("\n--- MENÚ CLIENTES ---")
    print("1) Agregar\n2) Buscar\n3) Actualizar\n4) Eliminar\n5) Listar\n0) Volver")
    op = input("Opción: ")

    if op == "1": agregar_cliente()
    elif op == "2": buscar_cliente()
    elif op == "3": actualizar_cliente()
    elif op == "4": eliminar_cliente()
    elif op == "5": listar_clientes()
    elif op == "0": break
    else: print("Opción inválida.")


def menu_tours():
    while True:
        print("\n--- MENÚ TOURS ---")
        print("1) Agregar\n2) Buscar\n3) Actualizar\n4) Eliminar\n5) Listar\n0) Volver")
        op = input("Opción: ")

        if op == "1": agregar_tour()
        elif op == "2": buscar_tour()
        elif op == "3": actualizar_tour()
        elif op == "4": eliminar_tour()
        elif op == "5": listar_tours()
        elif op == "0": break
        else: print("Opción inválida.")


def menu_reservas():
    while True:
        print("\n--- MENÚ RESERVAS ---")
        print("1) Crear reserva\n2) Cambiar estado\n3) Listar\n0) Volver")
        op = input("Opción: ")

        if op == "1": crear_reserva()
        elif op == "2": cambiar_estado_reserva()
        elif op == "3": listar_reservas()
        elif op == "0": break
        else: print("Opción inválida.")


def menu_pagos():
    while True:
        print("\n--- MENÚ PAGOS ---")
        print("1) Registrar pago\n2) Listar pagos\n0) Volver")
        op = input("Opción: ")

        if op == "1": registrar_pago()
        elif op == "2": listar_pagos()

```

```

        elif op == "0": break
    else: print("Opción inválida.")

# =====
# MENÚ PRINCIPAL
# =====

def menu_principal():
    while True:
        print("\n===== SISTEMA DE AGENCIA DE VIAJES =====")
        print("1) Clientes\n2) Tours\n3) Reservas\n4) Pagos\n0) Salir")
        op = input("Opción: ")

        if op == "1": menu_clientes()
        elif op == "2": menu_tours()
        elif op == "3": menu_reservas()
        elif op == "4": menu_pagos()
        elif op == "0": break
        else:
            print("Opción inválida.")

menu_principal()

```

2. Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos

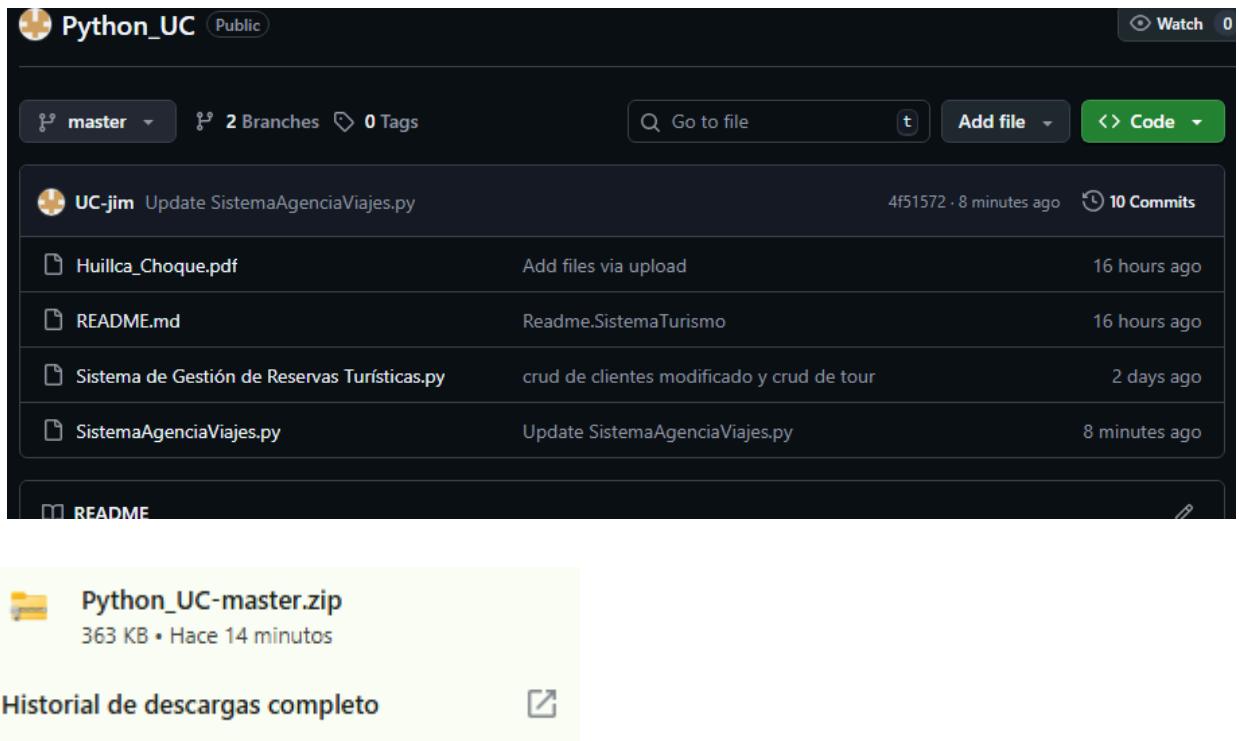
PROBLEMAS	SALIDA	CONSOLA DE DEPURACIÓN	<u>TERMINAL</u>	PUERTOS
			ID de la reserva: 1 ID del cliente: 1 ID del tour: 1 No existe el tour.	
			--- Crear Reserva --- ID de la reserva: 1 ID del cliente: 2 ID del tour: 1 No existe el cliente.	

3. Manual de usuario

Primero ingresamos a github y descargamos el archivo con el repositorio , dentro del readme

tambien indica lo que incluye el codigo y cual es su objetivo :

https://github.com/UC-jim/Python_UC



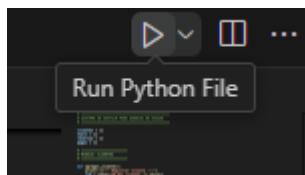
Segundo , extraemos el codigo y jalamos la carpeta a un editor de codigo donde podemos verificar que el codigo esta y poder ejecutarlo debemos tener instalado todas las extensiones de python , el python extension el pylance , ala par tener instalado la ultima version de python.

The screenshot shows a Python IDE interface with the file `SistemaAgenciaViajes.py` open. The code implements a travel agency system with various menu options for clients, tours, reservations, payment methods, and a main menu.

```
File Edit Selection View Go Run Terminal Help ← → ⌘ Python_UC-master ... SistemaAgenciaViajes.py ... Hallaz_Cheque.pdf

EXPLORER
PYTHON_UC_MASTER
  Hallaz_Cheque.pdf
  README.md
  Sistema de Gestión de Reservas Turísticas.py
  SistemaAgenciaViajes.py

SistemaAgenciaViajes.py
...
297     else: print("Opción Inválida.")
298
299     def menu_reservas():
300         while True:
301             print("\n--- MENÚ RESERVAS ---")
302             print("1) Crear reserva() Cambiar estado(nº) Listar pagos Volver")
303             op = input("Opción: ")
304
305             if op == "1": crear_reserva()
306             elif op == "2": cambiar_estado_reserva()
307             elif op == "3": listar_reservas()
308             elif op == "0": break
309             else: print("Opción Inválida.")
310
311
312     def menu_pagos():
313         while True:
314             print("\n--- MENÚ PAGOS ---")
315             print("1) Registrar pago() Listar pagos Volver")
316             op = input("Opción: ")
317
318             if op == "1": registrar_pago()
319             elif op == "2": listar_pagos()
320             elif op == "0": break
321             else: print("Opción Inválida.")
322
323
324     # -----#
325     # MENÚ PRINCIPAL
326     # -----#
327
328     def menu_principal():
329         while True:
330             print("----- SISTEMA DE AGENCIA DE VIAJES -----")
331             print("1) Clientes(nº) Tours(nº) Reservas(nº) Pagos(nº) Salir")
332             op = input("Opción: ")
333
334             if op == "1": menu_clientes()
335             elif op == "2": menu_tours()
336             elif op == "3": menu_reservas()
337             elif op == "4": menu_pagos()
338             elif op == "0": break
339             else:
340                 print("Opción Inválida.")
341
342     menu_principal()
```



Tercera haremos clic en el boton de run python file y nos abrira la terminal donde esta el sistema.

```
--- MENÚ CLIENTES ---
1) Agregar
2) Buscar
3) Actualizar
4) Eliminar
5) Listar
0) Volver
Opción: []
```

nos dara un menu donde nos mostrara varias opciones , primero agregaremos un cliente , para poder realizar las demas acciones luego como buscar actualizar , eliminar o lista. y de acuerdo a los datos que nos pedire rellenaremos

```
--- Registrar Cliente ---
ID del cliente: 2
Nombre completo: alex huaman
DNI: 72516423
Correo: freleida@gmail.com
```

```
Cliente registrado con éxito.
```

nos pedira un identificador , nombre completo el dni y el correo una vez termines nos dira cliente cliente registrado con exito y podremos realizar las demas acciones , tambien recordemos que no se nos permitira llenar con el mismo id dado que dara error y nos volvera al menu principal para elegir otra opcion.

```
--- Registrar Cliente ---
ID del cliente: 2
ERROR: Ese ID ya existe.
```

El menu de buscar cliente nos mostrara el id con que tipo de cliente y su nombre.

```
--- Buscar Cliente ---
ID del cliente: 1
{'nombre': 'pedro', 'dni': '1234', 'correo': 'freleida'}

--- MENÚ CLIENTES ---
1) Agregar
2) Buscar
3) Actualizar
4) Eliminar
5) Listar
0) Volver
Opción: []
```

```
--- MENÚ RESERVAS ---
```

- 1) Crear reserva
- 2) Cambiar estado
- 3) Listar
- 0) Volver

Opción: |

en el menu reservas veremos las siguientes opciones

```
--- MENÚ TOURS ---
```

- 1) Agregar
- 2) Buscar
- 3) Actualizar
- 4) Eliminar
- 5) Listar
- 0) Volver

Opción: |

en el menu Tours veremos las siguientes opciones son similares a las de agregar clientes pero en este momento crearemos tours para poder agregarlos

```
--- MENU PAGOS ---
```

- 1) Registrar pago
- 2) Listar pagos
- 0) Volver

Opción: |

por ultimo tenemos la vista donde registraremos los pagos y veremos que pagos se realizaron con el listar pagos

Capítulo 4: Evidencias de Trabajo en Equipo

1. Repositorio con Control de Versiones (Capturas de Pantalla)

- Registro de commits claros y significativos que evidencien aportes individuales (proactividad).
- Historial de ramas y fusiones si es aplicable.

Enlace a la herramienta colaborativa Imágenes de commit :

https://github.com/UC-jim/Python_UC/branches/

https://github.com/UC-jim/Python_UC/commits/master/

This screenshot shows the GitHub repository 'Commits - UC-jim/Python_UC' with the 'master' branch selected. The commits are listed in chronological order:

- Commits on Nov 23, 2025:
 - Update SistemaAgenciaViajes.py (Verified) 4f31572
 - Update SistemaAgenciaViajes.py (Verified) cd12dbe
 - Update SistemaAgenciaViajes.py (Verified) 578242a
 - Update SistemaAgenciaViajes.py (Verified) dff1108
- Commits on Nov 22, 2025:
 - Add files via upload (Verified) 461c55a
 - Add files via upload (Verified) 91ea652
 - Readme.SistemaTurismo (Verified) b705d63
 - crud de clientes modificado y crud de tour (Verified) 064ff22
- Commits on Nov 21, 2025:
 - crud de clientes (Verified) 031421d
 - primer archivo del proyecto (Verified) ab7de08

This screenshot shows a pull request on GitHub. The pull request is titled "Add files via upload" and is from user "Shomish" into the "master" branch. The commit message is: "Este sistema de gestión para una agencia de viajes permite administrar de forma sencilla clientes, tours, reservas y pagos. Incluye funciones completas de registro, búsqueda, actualización, eliminación y listado para cada módulo. Además, permite crear reservas vinculadas a clientes y tours, cambiar su estado y registrar pagos que actualizan automáticamente las reservas. Todo está organizado con menús interactivos que facilitan la navegación y el uso del sistema." The pull request has been merged automatically, indicated by the green checkmark and the message "No conflicts with base branch".

This screenshot shows the main page of the GitHub repository. The README file contains the following content:

```
Proyecto: Sistema de Gestión para Agencia de Viajes

Este proyecto implementa un sistema interactivo en Python diseñado para organizar las operaciones principales de una agencia de viajes. El sistema funciona mediante menús y permite realizar registros, búsquedas, actualizaciones y eliminaciones dentro de cuatro módulos fundamentales:

Módulos incluidos
Clientes: Registro y administración de datos personales como nombre, DNI y correo.
Tours: Gestión de tours disponibles, incluyendo nombre y precio.
Reservas: Creación de reservas vinculadas a clientes y tours, con control de estados (Pendiente, Pagado, Cancelado).
Pagos: Registro de pagos asociados a reservas, actualizando automáticamente el estado correspondiente.

Funcionalidades principales
CRUD completo para clientes y tours
```

This screenshot shows the 'Branches' page on GitHub. It displays three branches:

- Default**: The 'master' branch, last updated 4 minutes ago.
- Your branches**: The 'Shomish-patch-1' branch, last updated 16 hours ago.
- Active branches**: The 'Shomish-patch-1' branch, last updated 16 hours ago.

This screenshot shows the 'Branches' page on GitHub, identical to the previous one, displaying the 'master' and 'Shomish-patch-1' branches.