UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERÍA

INGENIERÍA INDUSTRIAL

PROGRAMACIÓN

Trabajo final, segunda entrega

DOCENTE:

John Heider Dávila Dávila

INTEGRANTES:

Carolayn Restrepo

Alexandra Vásquez

13 DE JULIO DE 2025

Contenido

INTRODUCCIÓN

Objetivo del código

El objetivo principal de este sistema es permitir la gestión eficiente del parqueadero ParqUdea, ubicado en el entorno de la Universidad de Antioquia. A través de una aplicación desarrollada en Python para consola, se facilita el registro de usuarios, la entrada y salida de vehículos, el cálculo de cobros, la generación de reportes administrativos y la exportación de datos.

Problema que resuelve

Antes del desarrollo de este software, el parqueadero operaba de forma manual, registrando los datos en papel y entregando recibos físicos con información limitada. Este método presentaba múltiples inconvenientes:

- Pérdida de información.
- Errores humanos en el cálculo de tarifas.
- Falta de trazabilidad y estadísticas.
- No existía un sistema de facturación ni reportes administrativos.

Este código automatiza ese proceso, brindando una solución digital confiable y funcional, sin necesidad de interfaz gráfica, pero completamente operativa desde consola.

Público objetivo

Este sistema está dirigido a:

- **Operarios del parqueadero ParqUdea**, quienes realizan la atención diaria y requieren registrar vehículos, emitir recibos y realizar cobros.
- **Administradores del parqueadero**, encargados de generar reportes, consultar estadísticas y monitorear el uso de los espacios.
- **Estudiantes o desarrolladores** que deseen estudiar un caso práctico de desarrollo de software en consola, orientado a problemas reales de gestión operativa.

REQUISITOS PREVIOS

- **Entorno de ejecución**
  - ✓ Tener acceso a Google Colab o un entorno local con Python
  - ✓ Si se usa localmente, se recomienda tener conocimientos básicos sobre cómo ejecutar un script .py.
- **Conocimientos básicos del usuario**

- ✓ Saber cómo ingresar texto por consola (el programa solicita datos por teclado).
- ✓ Comprender instrucciones básicas que aparecen en pantalla.
- ✓ No se requieren conocimientos avanzados de programación para usarlo, pero sí atención al ingresar los datos correctamente.

- **No se requieren librerías externas**
  - ✓ El sistema solo usa dos librerías estándar de Python:
    - ➢ Datetime (para controlar fechas y horas)
    - ➢ Math (para redondeos y cálculos)
  - ✓ Estas vienen incluidas con Python, por lo tanto, no es necesario instalar nada adicional

- **Datos necesarios para el registro**
  Para registrar un usuario correctamente, deberás tener a la mano:
  - ✓ Nombre y apellido del conductor (sin números ni símbolos)
  - ✓ Documento de identidad (entre 3 y 15 dígitos numéricos)
  - ✓ Placa del vehículo (3 letras seguidas de 3 números)

MANUAL DE USUARIO

PARQUEADERO PARQUDEA

1. **IMPORTAR LIBRERÍAS**
   Estas herramientas vienen incluidas en Python como librerías o módulos. Permiten realizar funciones esenciales que el programa necesita para operar correctamente:
   - ✓ **import datetime**: Carga la librería que permite registrar la hora exacta de ingreso y salida de los vehículos, calcular el tiempo de parqueo y mostrarlo en los recibos. Sin ella, no sería posible llevar un control preciso del tiempo.
   - ✓ **import math**: Carga funciones matemáticas avanzadas, como redondeos, que se usan para calcular cobros por fracciones de hora.

```
import datetime
import math
```

2. **LISTAS**
   Este segmento del código define las listas globales que almacenan la información clave del parqueadero. Son contenedores compartidos por todo el programa que permiten registrar nuevos usuarios, monitorear los vehículos actualmente parqueados y guardar el historial de retiros. En esencia, son la base para gestionar y seguir la actividad del parqueadero.

```
USUARIOS_REGISTRADOS = []
CARROS_PARQUEADOS = []
VEHICULOS_RETIRADOS_HISTORIAL = []
```

3. **VARIABLES**

   Este apartado es la "configuración básica" del parqueadero. Aquí se establecen los valores iniciales y las reglas fijas que el programa usará en su día a día. Este fragmento le da al programa toda la información clave sobre cómo debe operar el parqueadero: su capacidad, cuántos carros hay dentro y cuánto se debe cobrar por el servicio.

```
ESPACIOS_TOTALES = 64
ESPACIOS_OCUPADOS = 0
NOMBRE_PARQUEADERO = "ParqUdea"
VALOR_HORA_COMPLETA = 7000
VALOR_CUARTO_HORA = 1500
```

4. **CREDENCIALES DE ADMINISTRADOR**

   Las credenciales son importantes para la seguridad y para darle a tu programa las herramientas especiales que necesita. Por un lado, define quiénes son los usuarios que pueden acceder a la parte "secreta" o de gestión del sistema. Por otro lado, "carga" funcionalidades extras para manejar fechas, horas y cálculos matemáticos.

   Si alguien intenta iniciar sesión con un usuario o contraseña incorrectos, el sistema lo detectará gracias a esta lista.

```
ADMIN_USUARIOS = {
    "Admin": "clave123",
    "Gerente": "pass456"
}
```

5. **FUNCIONES DE VALIDACIÓN**

   Esta función se puede considerar un "inspector de texto". Su propósito es asegurar que la información que se ingresa en campos como nombres y apellidos contenga únicamente letras del alfabeto (mayúsculas o minúsculas) y espacios en blanco. No permite que se ingresen números, símbolos o cualquier otro carácter que no sea una letra o un espacio.

```python
def tiene_solo_letras_y_espacios(texto):
    if not texto: return False
    for caracter in texto:
        if not ('a' <= caracter <= 'z' or 'A' <= caracter <= 'Z' or caracter == ' '):
            return False
    return True
```

Utilizamos también un verificador de longitud mínima. Su objetivo es confirmar que un texto (como un nombre o un documento) tenga una cantidad de caracteres igual o mayor que el número que se le exige.

```python
def tiene_longitud_minima(texto, minimo):
    return len(texto) >= minimo
```

Además, se añadió una función que actuara como un "filtro de solo números". Su objetivo es verificar que un texto (como un número de documento o una parte de una placa) contenga solamente dígitos del 0 al 9, sin ninguna letra, espacio o símbolo. Es crucial para asegurar que los campos que deben ser numéricos no contengan caracteres erróneos. Si se encuentra algo que no sea un número, la función lo detecta y lo marca como inválido.

```python
def tiene_solo_numeros(texto):
    if not texto: return False
    for caracter in texto:
        if not ('0' <= caracter <= '9'): return False
    return True
```

El propósito de este segmento es verificar que la placa de un vehículo tenga el formato exacto requerido: 3 letras mayúsculas seguidas de 3 números. Esta validación es crucial para que todas las placas se registren de forma uniforme y puedan ser buscadas y comparadas correctamente en el sistema.

```python
def validar_formato_placa(placa):
    if not tiene_longitud_exacta(placa, 6): return False
    for i in range(3):
        if not ('A' <= placa[i] <= 'Z'): return False
    for i in range(3, 6):
        if not ('0' <= placa[i] <= '9'): return False
    return True
```

6. FUNCIONES DE USUARIO

Para dar cumplimiento a lo requerido el propósito de esta función es mostrar en pantalla, de manera clara, la información de todos los usuarios que han sido registrados en el sistema del parqueadero hasta el momento.

- ✓ **Si hay usuarios registrados:** La función revisa la lista de USUARIOS_REGISTRADOS y muestra cada uno de ellos, indicando su número, nombre completo, documento y la placa del vehículo asociado. Esto es muy útil para ver rápidamente quiénes están inscritos.
- ✓ **Si no hay usuarios:** Si la lista de USUARIOS_REGISTRADOS está vacía, simplemente informa que "No hay usuarios registrados aún".

```
def ver_usuarios_registrados():
    print("\n--- Usuarios Registrados en el Sistema (Actualizado) ---")
    if USUARIOS_REGISTRADOS:
        for i, user in enumerate(USUARIOS_REGISTRADOS):
            print(f"{i+1}. Nombre: {user['nombre']} {user['apellido']}, Documento: {user['documento']}, Placa: {user['placa']}")
    else:
        print("No hay usuarios registrados aún.")
```

Aquí tenemos el "punto de registro de clientes" en el sistema. Su objetivo principal es añadir la información de una nueva persona que desea usar el parqueadero. Cuando se activa esta función, el sistema le pedirá al operador los datos de la persona (como su nombre, apellido, número de identificación y la placa de su vehículo). Una vez que toda la información es ingresada correctamente, esta función se encarga de guardarla en el "registro de clientes" del parqueadero (USUARIOS_REGISTRADOS).

```
def registrar_usuario():

    # ... (código completo de registrar_usuario) ...


    print("\n--- Iniciar Registro de Nuevo Usuario ---")
```

```
while True:
    nombre = input("Ingrese el Nombre del usuario: ")
    errores_nombre = []
    if not tiene_longitud_minima(nombre, 3): errores_nombre.append("El nombre debe tener al menos 3 letras.")
    if not tiene_solo_letras_y_espacios(nombre): errores_nombre.append("El nombre no debe contener números ni caracteres especiales.")
    if errores_nombre:
        for error in errores_nombre: print(f"Error en Nombre: {error}")
    else: break

while True:
    apellido = input("Ingrese el Apellido del usuario: ")
    errores_apellido = []
    if not tiene_longitud_minima(apellido, 3): errores_apellido.append("El apellido debe tener al menos 3 letras.")
    if not tiene_solo_letras_y_espacios(apellido): errores_apellido.append("El apellido no debe contener números ni caracteres especiales.")
    if errores_apellido:
        for error in errores_apellido: print(f"Error en Apellido: {error}")
    else: break
```

```python
while True:
    documento = input("Ingrese el Documento del usuario: ").strip()
    errores_documento = []
    if not tiene_solo_numeros(documento): errores_documento.append("El documento solo debe contener números.")
    long_doc = len(documento)
    if not (3 <= long_doc <= 15): errores_documento.append("El documento debe tener entre 3 y 15 dígitos.")
    documento_ya_existe = False
    for usuario_existente in USUARIOS_REGISTRADOS:
        if usuario_existente['documento'] == documento:
            documento_ya_existe = True
            break
    if documento_ya_existe: errores_documento.append(f"El documento {documento} ya está registrado.")
    if errores_documento:
        for error in errores_documento: print(f"Error en Documento: {error}")
    else: break
```

```python
while True:
    placa = input("Ingrese la Placa del vehículo : ").upper()
    errores_placa = []
    if not tiene_longitud_exacta(placa, 6): errores_placa.append("La placa debe tener exactamente 6 caracteres.")
    if not validar_formato_placa(placa): errores_placa.append("La placa debe tener 3 letras seguidas de 3 números")
    placa_ya_existe = False
    for usuario_existente in USUARIOS_REGISTRADOS:
        if usuario_existente['placa'] == placa:
            placa_ya_existe = True
            break
    if placa_ya_existe: errores_placa.append(f"La placa {placa} ya está asociada a otro usuario.")
    if errores_placa:
        for error in errores_placa: print(f"Error en Placa: {error}")
    else: break
```

```python
nuevo_usuario = {
    "nombre": nombre,
    "apellido": apellido,
    "documento": documento,
    "placa": placa
}
USUARIOS_REGISTRADOS.append(nuevo_usuario)
print("\n--- Usuario Registrado Exitosamente ---")
print(f"Nombre: {nombre} {apellido}")
print(f"Documento: {documento}")
print(f"Placa Asociada: {placa}")

ver_usuarios_registrados()
```

Con esto se gestionará el acceso de los vehículos que desean parquearse. Antes de permitir que un carro entre, esta función realiza varias verificaciones importantes:

- ✓ **Disponibilidad:** Revisa si todavía hay puestos libres en el parqueadero.
- ✓ **Registro del cliente:** Confirma que la placa del vehículo pertenece a un usuario que ya está registrado en el sistema.
- ✓ **Vehículo no duplicado:** Se asegura de que el mismo carro no esté ya parqueado dentro.

Si todas las condiciones se cumplen, la función registra la hora en que el vehículo entró y actualiza el número de espacios que están ocupados.

```python
def ingresar_vehiculo():

    # ... (código completo de ingresar_vehiculo) ...


    global ESPACIOS_OCUPADOS
    print("\n--- Ingresar Vehículo al Parqueadero ---")
    if ESPACIOS_OCUPADOS >= ESPACIOS_TOTALES:
        print("\n Lo sentimos, El parqueadero está lleno. No se puede ingresar más vehículos.")
        return
    if not USUARIOS_REGISTRADOS:
        print("\n No hay usuarios registrados en el sistema.")
        print(" Registre al menos un usuario antes de intentar ingresar un vehículo.")
        return

    placa_ingreso = input("Ingrese la PLACA del vehículo (ej. ABC123) para ingresar: ").upper()
```

```python
usuario_encontrado = None
for user in USUARIOS_REGISTRADOS:
    if user['placa'] == placa_ingreso:
        usuario_encontrado = user
        break

if not usuario_encontrado:
    print(f"Error: La placa {placa_ingreso} no está registrada en el sistema de usuarios.")
    return

for carro_parqueado in CARROS_PARQUEADOS:
    if carro_parqueado['placa'] == placa_ingreso:
        print(f"Error: El vehículo con placa {placa_ingreso} ya se encuentra parqueado.")
        return

hora_entrada = datetime.datetime.now()
nuevo_carro_parqueado = {
    "placa": placa_ingreso,
    "documento_usuario": usuario_encontrado['documento'],
    "hora_entrada": hora_entrada
}
CARROS_PARQUEADOS.append(nuevo_carro_parqueado)
ESPACIOS_OCUPADOS += 1
```

```python
print(f"\nVehículo con placa {placa_ingreso} ingresado a las {hora_entrada.strftime('%H:%M:%S')}.")
print(f"Espacios disponibles: {ESPACIOS_TOTALES - ESPACIOS_OCUPADOS}")

print("\n--- RECIBO DE INGRESO ---")
print(f"Parqueadero: {NOMBRE_PARQUEADERO}")
print(f"Placa: {placa_ingreso}")
print(f"Usuario: {usuario_encontrado['nombre']} {usuario_encontrado['apellido']}")
print(f"Hora de Entrada: {hora_entrada.strftime('%Y-%m-%d %H:%M:%S')}")
print(f"Espacio Asignado: {ESPACIOS_OCUPADOS}")
print("¡Gracias por su visita!")
```

Aquí tendremos el punto de salida y cobro del parqueadero. Su propósito principal es gestionar el momento en que un vehículo abandona las instalaciones, incluyendo el cálculo de lo que debe pagar el cliente. Cuando se activa esta función, el sistema:

- ✓ **Identifica el vehículo:** Busca el carro que se desea retirar entre los que están actualmente parqueados.
- ✓ **Calcula el tiempo y el costo:** Determina cuánto tiempo estuvo parqueado el vehículo (desde su ingreso hasta su salida) y, usando las tarifas establecidas (valor por hora y por cuarto de hora), calcula el monto total a pagar.
- ✓ **Actualiza el historial:** Mueve el registro de ese vehículo de la lista de carros "parqueados" a la lista de "historial de retirados", conservando un registro de todas las salidas y los cobros realizados.
- ✓ **Libera el espacio:** Disminuye la cuenta de espacios ocupados, señalando que un puesto del parqueadero ha quedado libre para el siguiente vehículo.

```
def retirar_vehiculo():
    # ... (código completo de retirar_vehiculo) ...
    global ESPACIOS_OCUPADOS

    print("\n--- Retirar Vehículo del Parqueadero ---")

    if not CARROS_PARQUEADOS:
        print("No hay vehículos parqueados actualmente para retirar.")
        return

    placa_retiro = input("Ingrese la PLACA del vehículo a retirar (ej. ABC123): ").upper()
```

```
vehiculo_encontrado_idx = -1
vehiculo_a_retirar = None

for i, carro_parqueado in enumerate(CARROS_PARQUEADOS):
    if carro_parqueado['placa'] == placa_retiro:
        vehiculo_a_retirar = carro_parqueado
        vehiculo_encontrado_idx = i
        break

if not vehiculo_a_retirar:
    print(f"Error: El vehículo con placa {placa_retiro} no se encuentra parqueado.")
    return
```

```
hora_salida = datetime.datetime.now()
hora_entrada = vehiculo_a_retirar['hora_entrada']

tiempo_transcurrido = hora_salida - hora_entrada
segundos_totales = tiempo_transcurrido.total_segundos()

horas_enteras = int(segundos_totales // 3600)
segundos_restantes = segundos_totales % 3600

cobro_por_horas = horas_enteras * VALOR_HORA_COMPLETA

cuartos_de_hora = 0
if segundos_restantes > 0:
    cuartos_de_hora = math.ceil(segundos_restantes / 900)

cobro_por_cuartos = cuartos_de_hora * VALOR_CUARTO_HORA

total_a_pagar = cobro_por_horas + cobro_por_cuartos
```

```
if total_a_pagar < VALOR_HORA_COMPLETA:
    total_a_pagar = VALOR_HORA_COMPLETA

VEHICULOS_RETIRADOS_HISTORIAL.append({
    "placa": vehiculo_a_retirar['placa'],
    "documento_usuario": vehiculo_a_retirar['documento_usuario'],
    "hora_entrada": vehiculo_a_retirar['hora_entrada'],
    "hora_salida": hora_salida,
    "tiempo_total_segundos": segundos_totales,
    "pago_total": total_a_pagar
})

CARROS_PARQUEADOS.pop(vehiculo_encontrado_idx)
ESPACIOS_OCUPADOS -= 1

print("\n--- RECIBO DE RETIRO ---")
print(f"Parqueadero: {NOMBRE_PARQUEADERO}")
print(f"Placa: {placa_retiro}")
print(f"Hora de Entrada: {hora_entrada.strftime('%Y-%m-%d %H:%M:%S')}")
print(f"Hora de Salida: {hora_salida.strftime('%Y-%m-%d %H:%M:%S')}")
print(f"Tiempo Total Parqueado: {tiempo_transcurrido}")
print(f"Horas Completas: {horas_enteras}")
print(f"Cuartos de Hora Adicionales: {cuartos_de_hora}")
print(f"Cobro por Horas: ${cobro_por_horas:,.0f}")
print(f"Cobro por Cuartos: ${cobro_por_cuartos:,.0f}")
print(f"TOTAL A PAGAR: ${total_a_pagar:,.0f}")

print(f"Espacios disponibles: {ESPACIOS_TOTALES - ESPACIOS_OCUPADOS}")
```

7. MODULO DE ADMINISTRACION

El propósito de este segmento es verificar la identidad de la persona que intenta acceder a las opciones de gerencia.
Cuando un usuario elige entrar como administrador, esta función le pedirá un nombre de usuario y una contraseña. Luego, comparará estos datos con los usuarios y contraseñas registrados como administradores en el sistema. Si los datos son correctos, permitirá el acceso; de lo contrario, lo denegará.

```python
def login_administrador():
    """Solicita usuario y contraseña para acceder al modo administrador."""
    print("\n--- ACCESO ADMINISTRADOR ---")
    max_intentos = 3
    for intento in range(1, max_intentos + 1):
        usuario = input("Usuario: ").lower()
        contrasena = input("Contraseña:")

        if usuario in ADMIN_USUARIOS and ADMIN_USUARIOS[usuario] == contrasena:
            print("\n Acceso de administrador concedido")
            return True
        else:
            print(f"Credenciales incorrectas. Intento {intento} de {max_intentos}.")

    print("Demasiados intentos fallidos. Volviendo al menú principal.")
    return False
```

El "directorio" del módulo de administración tiene como propósito presentar en pantalla todas las opciones y reportes a los que un administrador puede acceder una vez que ha iniciado sesión exitosamente.
Cuando el administrador entra al sistema, esta función se encarga de mostrarle un listado claro de lo que puede hacer: ver el total de vehículos, los ingresos, los tiempos de parqueo, etc. Es esencial para guiar al administrador a través de las diferentes herramientas de gestión.

```python
def mostrar_menu_administrador():
    """Muestra las opciones del submenú de administración."""
    print("\n===== MENÚ DE ADMINISTRACIÓN =====")
    print("1. Total de vehículos registrados (histórico)")
    print("2. Total de vehículos retirados (histórico)")
    print("3. Total de vehículos sin retirar (parqueados actualmente)")
    print("4. Total facturado por vehículos retirados")
    print("5. Tiempo promedio de estancia por vehículo (retirados)")
    print("6. Lista de todos los usuarios registrados")
    print("7. Vehículo con tiempo de parqueo máximo y mínimo (retirados)")
    print("8. Volver al Menú Principal")
```

Este es el "centro de operaciones" para los administradores del parqueadero, permite que el personal autorizado consulte información clave y vea reportes detallados sobre el funcionamiento y las finanzas del negocio.

- ✓ **Acceso Restringido:** Lo primero que hace es asegurar que solo un administrador válido pueda entrar.
- ✓ **Menú Interactivo y Continuo:** Una vez dentro, el programa muestra repetidamente un menú de opciones y espera a que el administrador elija qué reporte quiere ver. Este menú se sigue mostrando hasta que el administrador decida salir.
- ✓ **Generación de Reportes Específicos:** Según el número que elija el administrador, la función presenta diferentes tipos de información:
- ✓ **Contadores:** Muestra el total de usuarios registrados, cuántos vehículos han salido y cuántos están parqueados actualmente, así como los espacios disponibles.
- ✓ **Financieros:** Calcula y muestra el monto total de dinero facturado por los vehículos que ya se han retirado.
- ✓ **Estadísticos:** Calcula el tiempo promedio que los vehículos pasan parqueados y puede identificar el vehículo que estuvo menos tiempo y el que estuvo más tiempo.
- ✓ **Listados:** Puede mostrar una lista completa de todos los usuarios registrados en el sistema.
- ✓ **Salida Segura:** Existe una opción específica para que el administrador pueda salir de este módulo y regresar al menú principal del sistema.
- ✓ **Mensajes de Error:** Si el administrador ingresa una opción que no existe, el sistema le informa que es una "Opción no válida".

```python
def modulo_administracion():
    """Gestiona el submenú y los reportes del administrador."""
    if not login_administrador():
        return
```

```
while True:
    mostrar_menu_administrador()
    opcion_admin = input("Seleccione una opción (1-8):")

    if opcion_admin == '1':
        print("\n REPORTE: Total de Vehículos Registrados")
        print(f"Total de usuarios registrados: {len(USUARIOS_REGISTRADOS)}")

    elif opcion_admin == '2':
        print("\n REPORTE: Total de Vehículos Retirados")
        print(f"Total de vehículos que han sido retirados: {len(VEHICULOS_RETIRADOS_HISTORIAL)}")

    elif opcion_admin == '3':
        print("\n REPORTE: Total de Vehículos Sin Retirar")
        print(f"Vehículos actualmente parqueados: {len(CARROS_PARQUEADOS)}")
        print(f"Espacios ocupados: {ESPACIOS_OCUPADOS}")
        print(f"Espacios disponibles: {ESPACIOS_TOTALES - ESPACIOS_OCUPADOS}")

    elif opcion_admin == '4':
        print("\n REPORTE: Total Facturado por Vehículos Retirados")
        total_facturado = sum(v['pago_total'] for v in VEHICULOS_RETIRADOS_HISTORIAL)
        print(f"Total facturado: ${total_facturado:,.0f}")
```

```
elif opcion_admin == '5':
    print("\n REPORTE: Tiempo Promedio de Estancia por Vehículo")
    if VEHICULOS_RETIRADOS_HISTORIAL:
        sum_segundos = sum(v['tiempo_total_segundos'] for v in VEHICULOS_RETIRADOS_HISTORIAL)
        promedio_segundos = sum_segundos / len(VEHICULOS_RETIRADOS_HISTORIAL)
        promedio_hms = str(datetime.timedelta(seconds=int(promedio_segundos)))
        print(f"Tiempo promedio de estancia: {promedio_hms}")
    else:
        print("No hay vehículos retirados aún para calcular el promedio.")

elif opcion_admin == '6':
    print("\n REPORTE: Lista de Todos los Usuarios Registrados")
    ver_usuarios_registrados()

elif opcion_admin == '7':
    print("\n REPORTE: Vehículo con Tiempo de Parqueo Máximo y Mínimo")
    if VEHICULOS_RETIRADOS_HISTORIAL:
        min_tiempo_vehiculo = VEHICULOS_RETIRADOS_HISTORIAL[0]
        max_tiempo_vehiculo = VEHICULOS_RETIRADOS_HISTORIAL[0]

        for v in VEHICULOS_RETIRADOS_HISTORIAL:
            if v['tiempo_total_segundos'] < min_tiempo_vehiculo['tiempo_total_segundos']:
                min_tiempo_vehiculo = v
            if v['tiempo_total_segundos'] > max_tiempo_vehiculo['tiempo_total_segundos']:
                max_tiempo_vehiculo = v
```

```
        print("\nVehículo con Menor Tiempo de Estancia:")
        print(f"  Placa: {min_tiempo_vehiculo['placa']}")
        print(f"  Tiempo: {str(datetime.timedelta(seconds=int(min_tiempo_vehiculo['tiempo_total_segundos'])))}")
        print(f"  Pago: ${min_tiempo_vehiculo['pago_total']:,.0f}")

        print("\nVehículo con Mayor Tiempo de Estancia:")
        print(f"  Placa: {max_tiempo_vehiculo['placa']}")
        print(f"  Tiempo: {str(datetime.timedelta(seconds=int(max_tiempo_vehiculo['tiempo_total_segundos'])))}")
        print(f"  Pago: ${max_tiempo_vehiculo['pago_total']:,.0f}")
    else:
        print("No hay vehículos retirados aún para este reporte.")

elif opcion_admin == '8':
    print("Volviendo al Menú Principal...")
    break
else:
    print("\nOpción no válida. Por favor, ingrese un número del 1 al 8.")
```

8. LÓGICA PRINCIPAL DEL PROGRAMA (MENÚ)

En este segmento se presenta de forma clara en la pantalla el menú principal con todas las acciones que se pueden realizar en el sistema.

- ✓ Muestra un título del menú principal.
- ✓ Lista las 5 opciones principales: Registrar, Ingresar, Retirar, Administrar y Salir.

Es lo primero que ve un usuario y le sirve de guía para saber qué puede hacer con el programa.

```
def mostrar_menu():
    """Muestra las opciones del menú principal."""
    print("\n===== MENÚ PRINCIPAL DEL PARQUEADERO =====")
    print("1. Registrar Nuevo Usuario")
    print("2. Ingresar Vehículo")
    print("3. Retirar Vehículo")
    print("4. Acceder como Administrador")
    print("5. Salir")
```

La función llamada main (que significa "principal"), es el motor que hace que todo el programa de parqueadero funcione continuamente. Es la que se encarga de:

- ✓ **Mantener el programa en marcha:** Utiliza un ciclo (while True) para que el programa se mantenga activo y repita el menú principal una y otra vez hasta que el usuario decida salir. Esto significa que no tienes que reiniciar el programa cada vez que haces una operación.
- ✓ **Mostrar el Menú:** En cada repetición del ciclo, llama a mostrar_menu_principal() para que el operador siempre vea las opciones disponibles.

- ✓ **Recibir la Elección del Usuario:** Espera a que el operador escriba un número (del 1 al 5) para indicar qué acción quiere realizar.
- ✓ **Dirigir la Acción:** Dependiendo del número que el operador ingrese, la función **main llama a la función específica que realiza** esa tarea:
  - ➢ Si elige '1', llama a registrar_usuario.
  - ➢ Si elige '2', llama a ingresar_vehiculo.
  - ➢ Si elige '3', llama a retirar_vehiculo.Si elige '4', llama a modulo_administracion (para que un administrador pueda ver los reportes).
- ✓ **Finalizar el Programa:** Si el operador elige '5', muestra un mensaje de despedida y break (rompe el ciclo), lo que hace que el programa termine su ejecución.
- ✓ **Manejar Errores de Elección:** Si el operador escribe un número diferente a los del 1 al 5, el programa le avisa que la opción no es válida y le pide que intente de nuevo.

```
def mostrar_menu():
    """Muestra las opciones del menú principal."""
    print("\n===== MENÚ PRINCIPAL DEL PARQUEADERO =====")
    print("1. Registrar Nuevo Usuario")
    print("2. Ingresar Vehículo")
    print("3. Retirar Vehículo")
    print("4. Acceder como Administrador")
    print("5. Salir")
```

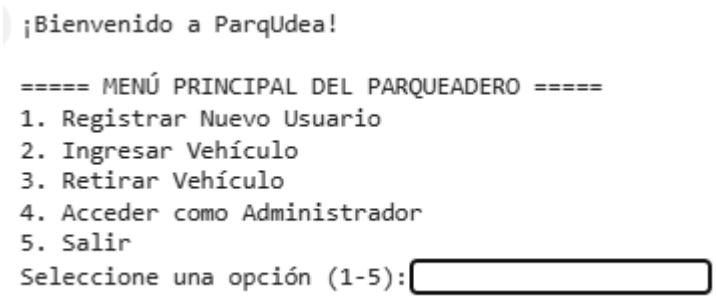
9. PUNTO DE ENTRADA DEL PROGRAMA

Este pequeño bloque de código actúa como el "botón de encendido" del sistema de parqueadero. Corresponde al punto de entrada principal del programa y tiene una función crucial: indica al computador cuándo debe empezar la ejecución real del sistema. Específicamente, al detectar que el archivo se está ejecutando directamente (y no como un módulo importado), llama a la función main(), la cual contiene el menú principal y la lógica que permite al usuario interactuar con todas las funcionalidades disponibles: registrar usuarios, ingresar vehículos, retirarlos o acceder al módulo de administración. En resumen, este fragmento le da la orden al programa de comenzar a funcionar y ponerse a disposición del usuario.

```
if __name__ == "__main__":
    main()
```

¿QUÉ RESULTADOS ESPERAR?

Una vez en ejecución, verás un menú como este:



Al interactuar con el sistema, podrás:

- **Registrar usuarios**, validando que los datos estén completos y correctos.
- **Ingresar vehículos**, generando un recibo con fecha y hora de entrada.
- **Retirar vehículos**, calculando automáticamente el cobro según el tiempo, y mostrando un recibo detallado.
- **Entrar al módulo de administración**, si tienes usuario y contraseña, para consultar:
    - ✓ Total de usuarios registrados.
    - ✓ Vehículos retirados y sin retirar.
    - ✓ Total facturado.
    - ✓ Estadísticas como tiempo promedio de parqueo y vehículo con mayor/menor tiempo.

Los recibos se imprimen directamente en la consola con todos los datos clave: placa, nombre, hora de entrada/salida, cobro por horas y por fracciones, y total a pagar.

Si cometes errores (por ejemplo, una placa ya registrada o un documento inválido), el sistema te lo indicará claramente y te pedirá corregir.