

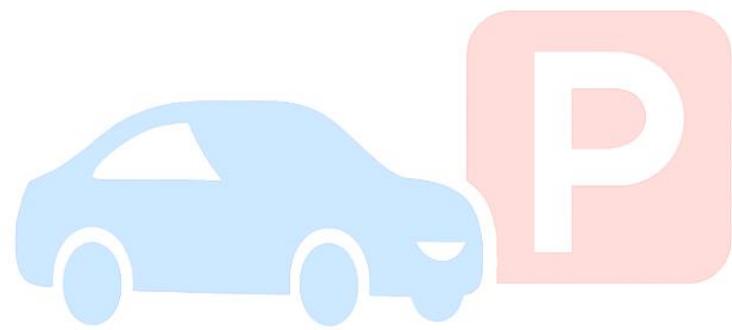
# MANUAL DE USUARIO

## SMARTPARKING

Sistema de gestión de parqueadero desarrollado para optimizar la entrada, salida y administración de vehículos de un espacio controlado.



## SMARTPARKING



# SMARTPARKING

# Tabla de contenido

|   |           |
|---|-----------|
| <b>1. INTRODUCCIÓN .....</b>                          | <b>5</b>  |
| OBJETIVO DEL SISTEMA.....                             | 5         |
| REQUISITOS DEL SISTEMA .....                          | 5         |
| <b>2. INSTALACIÓN DEL PROGRAMA .....</b>              | <b>6</b>  |
| REQUISITOS DE SOFTWARE .....                          | 6         |
| DESCARGA Y CONFIGURACIÓN PASO1 .....                  | 6         |
| ¿CÓMO ABRIR EL PROGRAMA CORRECTAMENTE? PASO 2.....    | 7         |
| <b>3. USO DEL PROGRAMA.....</b>                       | <b>9</b>  |
| MÓDULO DEL ADMINISTRADOR.....                         | 9         |
| MÓDULO DEL MENÚ PRINCIPAL.....                        | 14        |
| MÓDULO DESCRIPCIÓN REGISTRO DE USUARIOS .....         | 20        |
| MÓDULO DE OPERACIONES CENTRALES DEL PARQUEADERO.....  | 26        |
| <i>Flujo de operaciones.....</i>                      | 29        |
| MÓDULO GESTIÓN DE INGRESO Y RETIROS DE VEHÍCULOS..... | 32        |
| <b>4. INTERACCIÓN CON EL MENÚ PRINCIPAL .....</b>     | <b>43</b> |
| 1. REGISTRAR USUARIO .....                            | 44        |
| 2. INGRESAR VEHÍCULO .....                            | 44        |
| 3. RETIRAR VEHÍCULO .....                             | 44        |
| 5. MODO ADMINISTRADOR .....                           | 45        |
| 6. SALIR Y EXPORTAR DATOS .....                       | 45        |
| <b>TRABAJOS CITADOS .....</b>                         | <b>46</b> |

# SMARTPARKING

|                     |    |
|---------------------|----|
| Ilustración 1 ..... | 6  |
| Ilustración 2 ..... | 7  |
| Ilustración 3 ..... | 43 |



# 1. Introducción

Bienvenido al manual de usuario del sistema **Smart Parking**, el software de gestión de parqueaderos desarrollado para la Universidad de Antioquia. Este programa ha sido diseñado para simplificar la administración de vehículos y usuarios, automatizar los cobros y generar reportes administrativos de manera eficiente.

## Objetivo del sistema

El objetivo principal de Smart Parking es automatizar el registro de ingreso y salida de vehículos, calcular los cobros de manera precisa según el tiempo de estancia, y proporcionar herramientas administrativas para el análisis de ocupación y facturación.

**Eficiencia Operacional:** Reduce el tiempo y los errores asociados con los procesos manuales de registro y cobro.

**Trazabilidad:** Mantiene un historial digital completo de todos los ingresos, retiros y transacciones de facturación.

**Disponibilidad de Datos:** Permite generar reportes en tiempo real y exportar resultados para análisis financiero.

**Seguridad y Consistencia:** Garantiza que la información de usuarios y transacciones sea segura y consistente.

## Requisitos del Sistema

El software Smart Parking está diseñado para funcionar en cualquier terminal que tenga Python instalado. Es una aplicación de consola, lo que significa que se ejecuta a través de la línea de comandos de su sistema operativo.

## 2. Instalación del programa

Para utilizar Smart Parking, necesitará tener Python instalado en su ordenador.



ILUSTRACIÓN 1.

### Requisitos de Software

**Python:** Asegúrese de tener Python 3.x instalado. Puede descargarlo desde el sitio web oficial de Python:

[HTTPS://WWW.PYTHON.ORG/](https://www.python.org/)

### Descarga y Configuración Paso 1

- **Descargar el Proyecto:** Obtenga la carpeta del proyecto Smart Parking (Trabajofinal/Smartparking) del repositorio de GitHub.  
[HTTPS://GITHUB.COM/DANIELMR03/TRABAJOFINAL](https://github.com/DANIELMR03/TRABAJOFINAL)
- **Descomprimir:** Si descargó un archivo .zip, descomprimalo en una ubicación de su preferencia en su ordenador.
- **Estructura de Carpetas:** Asegúrese de que la estructura de carpetas sea la siguiente dentro de la carpeta principal del proyecto:

| Nombre        | Tipo                  | Tamaño |
|---------------|-----------------------|--------|
| __pycache__   | Carpeta de archivos   |        |
| datos         | Carpeta de archivos   |        |
| administrador | Archivo de origen ... | 2 KB   |
| main          | Archivo de origen ... | 3 KB   |
| usuario       | Archivo de origen ... | 2 KB   |
| utilidades    | Archivo de origen ... | 2 KB   |
| vehiculo      | Archivo de origen ... | 3 KB   |

## ILUSTRACIÓN 2

- **Abrir una Terminal o Símbolo del Sistema:**
  - Windows: Presione Win + R, escriba cmd y presione Enter.
  - macOS: Abra "Terminal" desde Aplicaciones > Utilidades.
  - Linux: Abra su aplicación de terminal.
- **Navegar al Directorio del Proyecto:** Use el comando cd (change directory) para ir a la carpeta donde guardó el proyecto. Por ejemplo, si lo guardó en C:\SmartParking:  
**cd C:\SmartParking**
- **Ejecutar el Programa:** Una vez que esté en el directorio correcto, ejecute el programa principal con el siguiente comando:  
**python main.py**  
 El menú principal de Smart Parking debería aparecer en su terminal.

## ¿Cómo abrir el programa correctamente? Paso 2

- **COPIAR LA CARPETA COMPLETA:**

¿Qué copiar?

Copia toda la carpeta principal del proyecto

Incluye todo lo que está dentro:

  - ✓ src/
  - ✓ doc/
  - ✓ requirements.txt
  - ✓ env/ (opcional, pero mejor evitarla ya que puede hacer que el programa no encuentre la ruta)

**Lo recomendable es NO copiar la carpeta env**

En lugar de copiar env, es mejor crear un nuevo entorno virtual en el otro computador (más abajo está el paso a paso de cómo crear el entorno virtual).

- **PEGAR LA CARPETA EN EL OTRO COMPUTADOR:**

Pega la carpeta en cualquier ubicación, por ejemplo, en el Escritorio del otro equipo.

- **INSTALAR PYTHON EN EL OTRO EQUIPO**

Asegúrate de que el otro computador tenga:

- Python 3.10 o superior
- Visual Studio Code
- Las extensiones de Python ya instaladas

- **CREAR Y ACTIVAR UN NUEVO ENTORNO VIRTUAL:**

Paso a paso:

1. Abre Visual Studio Code.
2. Abre la carpeta del proyecto (proyecto-parqueadero) desde “Archivo > Abrir carpeta”.
3. Abre la terminal (Ctrl + ñ).
4. Ejecuta este comando para crear el entorno: **python -m venv env**
5. Activa el entorno virtual:
  - En Windows: \env\Scripts\activate

Verás que aparece: **(env) C:\Users\Nombre\Desktop\proyecto-parqueadero>**

- **INSTALAR LAS LIBRERÍAS NECESARIAS:**

Con el entorno activado, instala las librerías necesarias usando requirements.txt: **pip install -r requirements.txt**

Esto instalará pandas, numpy, plotly

- **EJECUTAR EL PROYECTO:**

Ya puedes correr el proyecto. Ingresa este código en la terminal **python src/main.py**

Y te debe salir el menú principal del sistema.

### 3. Uso del programa

Para poder evidenciar el uso del programa se plantea mostrar de manera estructurada y detallada cada parte de los módulos con la finalidad de que el lector tenga un mejor entendimiento sobre los módulos del sistema, así como permitirle poder evidenciar la explicación de línea por línea.

#### Módulo del administrador

```
import json
import csv
import os
from datetime import datetime
from utilidades import RUTA_USUARIOS, RUTA_INGRESOS, RUTA_RETIROS

RUTA_ADMIN = os.path.join("src", "datos", "admin.json")

def menu_administrador():
    print("\n--- ACCESO ADMINISTRADOR ---")
    usuario = input("Usuario: ").strip()
    clave = input("Contraseña: ").strip()
    if not autenticar(usuario, clave):
        print("X Acceso denegado.")
        return
    print("✓ Acceso concedido.")
    mostrar_reportes()

def autenticar(usuario, clave):
    try:
        with open(RUTA_ADMIN, "r") as file:
            data = json.load(file)
            return data.get(usuario) == clave
    except FileNotFoundError:
        return False

def mostrar_reportes():
    print("\n--- REPORTES ADMIN ---")
    usuarios = sum(1 for _ in open(RUTA_USUARIOS)) if os.path.exists(RUTA_USUARIOS) else 0
    ingresados = sum(1 for _ in open(RUTA_INGRESOS)) if os.path.exists(RUTA_INGRESOS) else 0
    retirados = sum(1 for _ in open(RUTA_RETIROS)) if os.path.exists(RUTA_RETIROS) else 0
```

```

pagos = []
tiempos = []
try:
    with open(RUTA_RETIROS, mode="r") as archivo:
        lector = csv.reader(archivo)
        for fila in lector:
            pagos.append(int(fila[6]))
            inicio = datetime.strptime(fila[2], "%Y-%m-%d %H:%M:%S")
            fin = datetime.strptime(fila[3], "%Y-%m-%d %H:%M:%S")
            tiempos.append((fin - inicio).total_seconds() / 60)
except:
    pass
print(f"Usuarios registrados: {usuarios}")
print(f"Vehículos en parqueadero: {ingresados}")
print(f"Vehículos retirados: {retirados}")
print(f"Total recaudado: ${sum(pagos):,.0f}")
if tiempos:
    print(f"Tiempo promedio de estadía: {sum(tiempos)/len(tiempos):.1f} minutos")
    print(f"Máximo: {max(tiempos):.1f} min, Mínimo: {min(tiempos):.1f} min")

```

Primeramente, comenzaremos por desglosar por partes el código para tener conocimiento y su razón de ser acerca del funcionamiento de cada parte del código.

```

import json
import csv
import os
from datetime import datetime
from utilidades import RUTA_USUARIOS, RUTA_INGRESOS, RUTA_RETIROS

```

Inicialmente se empieza describiendo el funcionamiento de las librerías las cuales tienen aporte fundamental en el buen desarrollo y funcionamiento del sistema de parqueo. json: Funciona para tener un acceso a los datos de autenticación del administrador, permitiendo leer el archivo admin.json, el cual da acceso al sistema de la plataforma. csv: su función se centra en leer archivos planos como lo son (ingresos.csv, retiros.csv) mediante a los registros que se obtiene del parqueadero. datetime: Esta librería se centra en calcular el tiempo de estancia de los automóviles en el parqueadero. utilidades: Cumple con importar rutas para tener un acceso de manera funcional hacia los datos, permitiendo así una buena práctica para conservar un orden.

```
#Ruta del Archivo del Administrador  
RUTA_ADMIN = os.path.join("src", "datos", "admin.json")
```

Su función es definir una ruta concreta para ubicar y enrutar el archivo el cual contiene los datos sobre la autenticidad de los administradores en el sistema. Esta parte facilita un acceso más eficiente.

```
#Función menú administrador  
  
def menú_administrador():  
  
    print("\n--- ACCESO ADMINISTRADOR ---")  
  
    usuario = input("Usuario: ").strip()  
  
    clave = input("Contraseña: ").strip()  
  
  
    if not autenticar(usuario, clave):  
  
        print("Acceso denegado.")  
  
        return
```

```
    print(" Acceso concedido.")  
  
    mostrar_reportes()
```

La finalidad de esta función radica en gestionar el ingreso del administrador hacia el sistema del parqueadero. se llama la función autenticar(usuario, clave) para poder verificar las credenciales que se presentan almacenadas en el archivo admin.json. En tal caso de que las credenciales ingresadas no sean validas se muestra un mensaje mediante acceso denegado y se sale de la función por medio de un return. La función radica en cumplir los requisitos que se presentan en el documento del trabajo, sobre que solo un administrador valido puede acceder de manera correcta al menú administrador.

```
# Función autenticar usuario  
  
def autenticar(usuario, clave):  
  
    try:  
  
        with open(RUTA_ADMIN, "r") as file:  
  
            data = json.load(file)  
  
            return data.get(usuario) == clave  
  
    except FileNotFoundError:  
  
        return False
```

El objetivo de esta función se centra en validar la contraseña y el usuario que una persona presenta en el sistema con la finalidad de acceder al sistema del administrador registrada en el sistema de parqueo. Mediante a esto se mencionará

levemente los múltiples elementos que forman la función con la finalidad de tener un mejor manejo en el concepto del código:

- try: Se utiliza para poder tener un mejor manejo de los leves errores que pueden ocurrir al integrar abrir o leer el archivo admin.json. Esto principalmente es para tener un mejor manejo sobre la estabilidad y la utilidad del programa.
- with open (RUTA\_ADMIN, "r") as file: Permite abrir el archivo admin.json que se encuentra ubicado en RUTA\_ADMIN, progresivamente se utiliza el modo lectura "r", y se utiliza el ciclo with para tener certeza de que el archivo se cierra automáticamente después de ya utilizado.
- data = json.load(file): Esta función es transformar el contenido presente en el archivo json en un diccionario.
- return data.get(usuario) == clave: Verifica si la clave (contraseña) asociada al usuario en el archivo coincide con la ingresada. Se usa get() en lugar de acceder directamente con data[usuario] para evitar errores si el usuario no existe.
- except FileNotFoundError: Si el archivo admin.json no existe, se captura esta excepción específica. Así se evita que el programa se detenga de forma inesperada.

En resumen, esta función permite de manera clara autenticar el usuario de administrador. A continuación, se presentan los reportes, de igual manera se mostrará la parte del código a explicar y se desglosará la explicación sobre cada línea.

```
#Reportes

def mostrar_reportes():

    print("\n--- REPORTES ADMIN ---")

    usuarios = sum(1 for _ in open(RUTA_USUARIOS)) if os.path.exists(RUTA_USUARIOS) else 0
    ingresados = sum(1 for _ in open(RUTA_INGRESOS)) if os.path.exists(RUTA_INGRESOS) else 0
    retirados = sum(1 for _ in open(RUTA_RETIROS)) if os.path.exists(RUTA_RETIROS) else 0
    pagos = []
    tiempos = []

    try:

        with open(RUTA_RETIROS, mode="r") as archivo:
            lector = csv.reader(archivo)
            for fila in lector:
                pagos.append(int(fila[6]))
                inicio = datetime.strptime(fila[2], "%Y-%m-%d %H:%M:%S")
                fin = datetime.strptime(fila[3], "%Y-%m-%d %H:%M:%S")
                tiempos.append((fin - inicio).total_seconds() / 60)
```

```

except:
    pass

print(f" ◊ Usuarios registrados: {usuarios}")
print(f" ◊ Vehículos en parqueadero: {ingresados}")
print(f" ◊ Vehículos retirados: {retirados}")
print(f" ◊ Total recaudado: ${sum(pagos):,.0f}")

if tiempos:
    print(f" ◊ Tiempo promedio de estadía: {sum(tiempos)/len(tiempos):.1f} minutos")
    print(f" ◊ Máximo: {max(tiempos):.1f} min, Mínimo: {min(tiempos):.1f} min")

```

Esta función cumple con la tarea de mostrar los reportes en la consola del parqueadero, una vez el administrador ha diligenciado sus datos de manera correcta en el sistema, utiliza múltiples funciones y expresiones con la finalidad de mostrar reportes de usuarios registrados utilizando elementos como (usuarios.csv, ingresos.csv, retiros.csv). Estos elementos permiten, además de mostrar el número de usuarios registrados, mostrar los vehículos que se encuentran dentro del parqueadero, así como el número de automóviles que han salido del establecimiento. Para eludir diversos errores se incluye (os.path.exists). Por consiguiente, se resaltan dos listas las cuales son pagos y tiempos, las dos listas se encargan de registrar el tiempo de un vehículo, así como el registro del dinero almacenado por los vehículos que han ingresado al parqueadero. Se empezará describiendo parte por parte sobre el

```

try:
    with open(RUTA_RETIROS, mode="r") as archivo:
        lector = csv.reader(archivo)

```

Elementos como with open (RUTA\_RETIROS, mode="r") as archivo: lector = csv.reader(archivo): cumple con la función de abrir el archivo en un tipo de modo lectura, y se itera sobre cada fila del csv.reader. se extrae el valor pagado, se prioriza de igual manera el tiempo de estadía a partir de que el automóvil ingresa al parqueadero hasta que lo abandona, además de llevar de igual manera el registro de las fechas y horas exactas.

```

except:
    pass

```

Cumple la función de ignorar a toda costa los errores que se pueden presentar ocultándolos para que no se reflejen en la solución final.

```

print(f" ◊ Usuarios registrados: {usuarios}")
print(f" ◊ Vehículos en parqueadero: {ingresados}")
print(f" ◊ Vehículos retirados: {retirados}")

```

```
print(f" ◊ Total recaudado: ${sum(pagos):,.0f}")
```

Los elementos de print reflejan los resultados de las cifras básicas presentadas en el parqueadero.

```
if tiempos:  
    print(f" ◊ Tiempo promedio de estadía: {sum(tiempos)/len(tiempos):.1f} minutos")  
    print(f" ◊ Máximo: {max(tiempos):.1f} min, Mínimo: {min(tiempos):.1f} min")
```

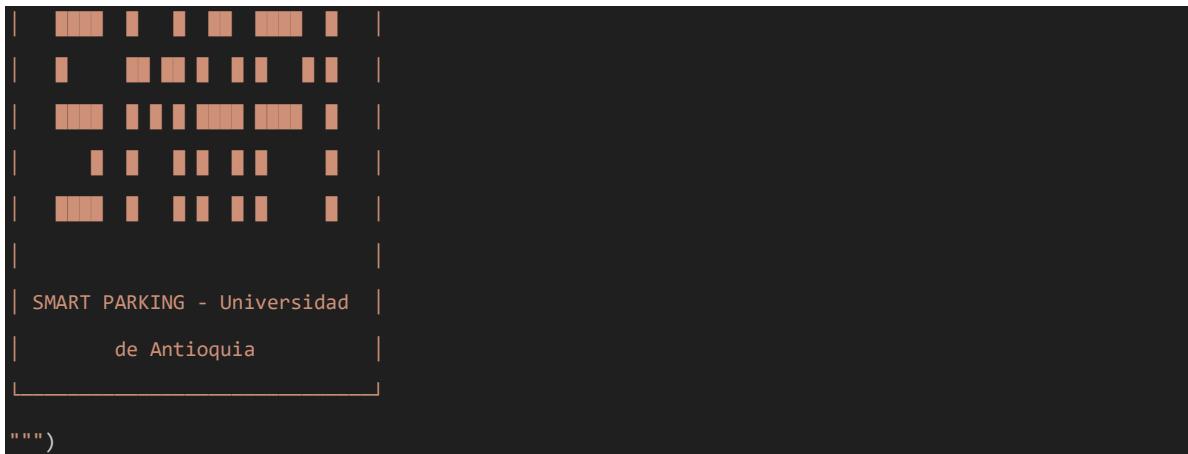
Cumple con la función de tener un cálculo aproximado del máximo y mínimo de estadía que se puede presentar en el parqueadero por medio de sus usuarios.

En conclusión, el código sobre el módulo del administrador representa una parte estructural necesaria para un buen sistema de gestión en cuanto al parqueadero, ya que permite tener acceso y control a diferentes aspectos que requiere el administrador, permitiéndole tener a la mano la generación de reportes operativos esenciales para un funcionamiento óptimo.

Con base a estructuras planas y modulares, este módulo permite al administrador tener acceso a la información por medio del archivo Json. En caso de que la información registrada sea exitosa se plantean estadísticas importantes, como el número de automóviles ingresados al establecimiento, usuarios registrados, un total de automóviles retirados, un total de tiempo de estancia en el parqueadero, así como también se presenta el recaudo de concepto de dinero. El código hace un uso efectivo de los archivos planos (csv, Json), además de diferentes librerías, como os, csv, datetime, así como un manejo de estructuras de listas, para poder tener un mejor funcionamiento respecto al código, demostrando un buen manejo de herramientas que proporciona python. Para concluir se resaltan las buenas prácticas ejercidas en el código, las cuales cooperan con el manejo de errores, espacios entre funciones y demás. Este código cumple con la constitución de un avance óptimo en el planteamiento y resolución del problema.

## Módulo del menú principal

```
from usuario import menú_usuarios  
from vehiculo import menu_vehiculos  
from administrador import menu_administrador  
from utilidades import exportar_datos  
  
def menu_principal():  
    def print_logo():  
        print("""
```



```

        if confirmar == "s":
            exportar_datos()
            print("✓ Archivo 'reporte_smartparking_DD-MM-YYYY.csv' creado | \n¡Hasta pronto!")
        else:
            print("¡Hasta pronto!")
        break
    else:
        print("Opción no válida.")

```

```

if __name__ == "__main__":
    menu_principal()

```

Como se realizó en el módulo anterior se procederá a realizar de manera precisa y específica cada parte del código que hace parte del módulo del menú principal con la finalidad de comprender de manera sencilla el funcionamiento del código y como cada parte de este es fundamental para su buen desarrollo y funcionamiento.

```

from usuario import menu_usuarios
from vehiculo import menu_vehiculos
from administrador import menu_administrador
from utilidades import exportar_datos

```

Se introduce con el inicio del código la cual es una secuencia de importar funciones importantes desde otros módulos que se presentan el sistema, restructurando el código en componentes separados de manera clara. Esta tarea funciona para permitir que el archivo principal (menu\_principal.py), actúe de manera central sobre el control del sistema. A continuación, resaltemos cada línea y cuál es su función:

- from usuario import menu\_usuarios: Se encarga de importar la función la cumple con la función de mostrar el submenú para el registro correcto de los usuarios en el sistema del parqueadero. Su función radica en permitir el ingreso de usuarios validando de esta manera su nombre, apellido, documento y la placa de su vehículo.
- from vehículo import menu\_vehiculos: Su finalidad se centra en importar la función la cual cumple la tarea del manejo del ingreso como del desalojo de los vehículos. Esta función está programada para tener un registro sobre el intervalo de tiempo de entrada y salida de los automóviles, así como del concepto de cobro.

- from administrador import menu\_administrador: Se centra en importar el menú del administrador el cual tiene un acceso restringido a personas autorizadas. A través de este módulo se controla el sistema, y se tiene a la mano estadísticas y reportes sobre el sistema.
- from utilidades import exportar\_datos: Su funcionalidad se centra en exportar los datos del sistema mediante un archivo csv. Lo que se logra con esto es poder almacenar de manera ordenada la información que se recolecta a lo largo de un periodo de jornada.

```
def menu_principal():
    def print_logo():
        print("""
        | █ █ █ █ █ █ █ |
        | █ █ █ █ █ █ █ |
        | █ █ █ █ █ █ █ |
        | █ █ █ █ █ █ █ |
        | █ █ █ █ █ █ █ |
        | █ █ █ █ █ █ █ |
        | █ █ █ █ █ █ █ |
        | █ █ █ █ █ █ █ |
        | SMART PARKING - Universidad |
        | de Antioquia |
        | """)
```

La función def menú\_principal() y la función def print\_logo() son dos funciones que se basan en definir tanto el primer acercamiento que tiene el usuario con el software del parqueadero, así como imprimir el logo característico del proyecto del parqueadero. La presentación cumple con la finalidad de darle un acercamiento ameno al usuario además de mostrar una identidad visual.

```
while True:
    print("\n")
    print_logo()
    print(" [-----] ")
    print(" |       MENÚ PRINCIPAL      | ")
    print(" |-----| ")
    print(" | 1. Registrar usuario   | ")
    print(" | 2. Ingresar vehículo    | ")
    print(" | 3. Retirar vehículo     | ")
```

```

        print(" | 4. Modo administrador      | ")
        print(" | 5. Salir                  | ")
        print(" \n-----| ")

```

El ciclo while true, esta línea implementa un bucle infinito que mantiene una sintaxis bastante sencilla pues su función es mantener el sistema operativo activo hasta que el usuario decida salir del sistema del parqueadero. Su finalidad constituye en mantener una interacción constante y fluida con el usuario.

Seguido del ciclo while se resalta un salto de línea para separar visualmente cada línea e iteración del menú. Seguido de ese elemento se imprime el menú principal con las 5 opciones características del sistema y las cuales son las que defienden el menú principal mostrando diferentes opciones, las cuales se presentan a continuación:

- registrar el usuario; Permite tanto registrar como validar nuevos usuarios. - ingresar el vehículo; Permite el ingreso de usuarios nuevos que con antelación fueron registrados.
- retirar el vehículo; Permite tener un manejo en el registro de la salida de los vehículos del recinto del parqueadero.
- modo administrador; Este es un acceso restringido únicamente para las personas autorizadas del parqueadero, esta función permite la visualización completa del sistema del parqueadero.
- salir; Para concluir se finaliza el programa dando una opción de exportar los datos del sistema.

```

opcion = input("Ingrese una opción: ")

if opcion == "1":
    menu_usuarios()
elif opcion == "2":
    menu_vehiculos("ingreso")
elif opcion == "3":
    menu_vehiculos("retiro")
elif opcion == "4":
    menu_administrador()
elif opcion == "5":
    print("\n-----| ")
    print(" |      >> EXPORTAR DATOS <<      | ")
    print(" \n-----| ")
    confirmar = input("¿Generar reporte CSV? (S/N): ").lower()
    if confirmar == "s":

```

```

        exportar_datos()

        print("✓ Archivo 'reporte_smartparking_DD-MM-YYYY.csv' creado |\n¡Hasta pronto!")

    else:
        print("¡Hasta pronto!")
        break

else:
    print("Opción no válida.")

```

```

if __name__ == "__main__":
    menu_principal()

```

"Para concluir se plantea en desglosar la última parte del código para tener una idea de su funcionamiento en cuanto a la ayuda del parqueadero. Se empieza con "opción = input ("Ingrese una opción: ") " el cual solicita al usuario que ingrese el número de la opción deseada, esta entrada funciona como apertura para la siguiente fase del código, el cual es una estructura condicional if-elif. Esta secuencia condicional funciona para dar la orientación de guía habiendo múltiples opciones relacionada con el módulo anterior, ya que plantea las opciones ya mencionadas a continuación se describe su funcionalidad.

- if opción == "1": menu\_usuarios (); su finalidad es llamar al módulo menu\_usuario(), el cual concede el registro de nuevos usuarios realizando el respectivo manejo de datos para su validación (nombre, apellido, documento, placa).
- elif opción == "2": menu\_vehiculos("ingreso"); Esta opción dos llamas al módulo menu\_vehiculo () con la finalidad de permitir el ingreso de nuevos vehículos por medio del parámetro "ingreso", especificando de esta manera que se va a registrar la entrada de un nuevo vehículo al parqueadero.
- elif opcion == "3": menu\_vehiculos("retiro"); Esta opción 3 llama de igual manera al módulo anterior pero esta vez con una finalidad diferente, ya que por medio del parámetro "retiro", indica que se desea retirar el vehículo del establecimiento del parqueadero, permitiendo además calcular el intervalo de tiempo del vehículo en el recinto, así como el concepto de cobranza correspondiente.
- elif opcion == "4": menu\_administrador (); La opción 4 permite el acceso al menú del administrador, desde el cual se plantean las estadísticas y diferentes reportes del sistema del parqueadero.
- elif opcion == "5": la opción cinco indica la salida del sistema finalizando de esta manera el programa. Además, da la opción de formar un archivo CSV recopilando de esta manera todos los datos del sistema permitiendo de esta manera para los encargados guardar los registros de una jornada laboral.

Por consiguiente, se imprime un aviso de que se van a exportar los datos, sucesivamente se pide al cliente la confirmación de dicha acción (S o N). se utiliza el .lower(), para asegurarse que la respuesta del cliente sea indiferente a una mayúscula o minúscula. Si se escoge "s" se llama a la función que crea el archivo CSV con la información previamente recopilada, si no se escoge simplemente se cierra el programa sin exportar la información. Para concluir el ciclo while se utiliza la función break.

Para dar por terminado el código se resalta una opción no válida el cual imprime una entrada invalida, resaltando en este caso letra o números que estén en fuera de lugar o rango, mostrando el mensaje no valido. la parte del if \_\_name\_\_ == "\_\_main\_\_": funciona como un punto de entrada, quiere decir que es como un tipo de bloqueo el cual asegura que solo se ejecuta con el archivo directamente.

El código del menú principal integra un centro de núcleo el cual compone un sistema de navegación y de gestión entorno al usuario del parqueadero, por medio de una consola sencilla con una buena estructuración con su logo característico, guiando de manera amigable al usuario para las funciones fundamentales del sistema.

## Módulo descripción registro de usuarios

Como ya se explicó con los dos módulos anteriores, primeramente, se exhibe todo el módulo con la finalidad de permitir tener idea de las partes que lo componen, y por consiguiente se realiza una explicación detallada sobre cada línea que compone el módulo.

```
import csv
import re
import os

RUTA_USUARIOS = os.path.join("src", "datos", "usuarios.csv")
```

```
def validar_nombre(nombre):
    return nombre.isalpha() and len(nombre) >= 3
```

```
def validar_documento(documento):
    return documento.isdigit() and 3 <= len(documento) <= 15
```

```
def validar_placa(placa):
    return bool(re.match(r"^[A-Z]{3}[0-9]{3}$", placa.upper()))
```

```

def registrar_usuario():
    print("\n--- REGISTRO DE USUARIO ---")

    nombre = input("Ingrese el nombre: ").strip()
    apellido = input("Ingrese el apellido: ").strip()
    documento = input("Ingrese el número de documento: ").strip()
    placa = input("Ingrese la placa del vehículo (ej. ABC123): ").strip().upper()

    errores = []

    if not validar_nombre(nombre):
        errores.append(" El nombre debe tener al menos 3 letras y no contener números.")

    if not validar_nombre(apellido):
        errores.append(" El apellido debe tener al menos 3 letras y no contener números.")

    if not validar_documento(documento):
        errores.append(" El documento debe contener entre 3 y 15 dígitos numéricos.")

    if not validar_placa(placa):
        errores.append(" La placa debe tener el formato correcto: 3 letras seguidas de 3 números.")

    if errores:
        print("\n".join(errores))
        return

with open(RUTA_USUARIOS, mode="a", newline="", encoding="utf-8") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow([documento, nombre, apellido, placa])
    print(f" Usuario registrado exitosamente.")

def menu_usuarios():
    registrar_usuario()

```

Para dar una introducción a la estructura de códigos que componen el siguiente módulo, empezaremos explicando primeramente las librerías y cuál es su papel enfocado en el desarrollo y la resolución del sistema del software del parqueadero.

```
import csv  
import re  
import os
```

Las tres librerías componen un propósito fundamental y específico que abarca distintos tipos de funciones que permiten que el módulo funcione de una manera óptima. La librería csv permite tanto leer como escribir archivos de forma csv el cual es una forma bastante legible y sencilla de guardar datos de manera estructurada, su funcionalidad en el software del parqueadero es fundamental ya que ayuda a almacenar los datos de los usuarios registrados en el parqueadero. La librería "re" permite el trabajo con expresiones regulares, su función en el software se especifica en por medio de la función validar\_placa () verificar que se cumpla el formato establecido para la placa de un vehículo la cual requiere una estructura de 3 letras seguidas como de 3 números. Por último, la librería "os", permite la utilización de herramientas las cuales ayudan al sistema a tener rutas de archivos, en el software se utiliza para encaminar una ruta del archivo usuario .csv.

```
RUTA_USUARIOS = os.path.join("src", "datos", "usuarios.csv")
```

Esta función cumple con la tarea de definir una llamada a RUTA\_USUARIOS, la cual contiene otra ruta donde se abarcan los datos de los usuarios por medio de usuarios.csv. os.path.join () construye diferentes rutas seguras y que además son compatibles con un sistema operativo, en el caso del módulo se unen tres carpetas las cuales son src; la cual es la carpeta del código fuente, seguido de la carpeta "datos" la cual se encarga de almacenar los datos y hace un hincapié a la siguiente carpeta la cual es "usuarios.csv" la cual se encarga de recopilar y almacenar los datos y la información de los usuarios que se registran en el parqueadero.

```
def validar_nombre(nombre):  
    return nombre.isalpha() and len(nombre) >= 3  
  
def validar_documento(documento):  
    return documento.isdigit() and 3 <= len(documento) <= 15  
def validar_placa(placa):  
    return bool(re.match(r"^[A-Z]{3}[0-9]{3}$", placa.upper()))
```

La finalidad de estas tres funciones de validación es esenciales en para tener un buen uso del programa del parqueadero por medio del usuario, verificando que los datos que se ingresen se han correctos. Se realizará una explicación de cada función así de cual papel desarrollan en la resolución de este módulo:

- def validar\_nombre(nombre): return nombre.isalpha() and len(nombre) >= 3; la función validar\_nombre cumple con las especificaciones de ingreso del nombre y apellido del usuario, además de que se encarga por medio del comando isalpha() que la información introducida contenga solamente letras con el comando, por medio del comando len(nombre) >= 3, el archivo se encarga de que la información introducida tenga mínimo 3 caracteres, mejorando de gran manera la coherencia y la calidad del sistema del parqueadero.
- def validar\_documento(documento): return documento.isdigit() and 3 <= len(documento) <= 15; la función def validar\_documento(documento), se encarga de realizar una validación sobre el número de documento, viene incluida con el comando .isdigit() el cual cumple con la tarea de que se contenga solo números al momento de acceder el documento, el comando 3 <= len(documento) <= 15, se encarga de que la información requerida en este ítem tenga entre 3 y 15 caracteres de dígitos.
- def validar\_placa(placa): return bool(re.match(r"^[A-Z]{3}[0-9]{3}\$", placa.upper())); Para finalizar la función validar\_placa(placa): se encarga de validar que la placa ingresada por el usuario tenga el formato correcto, el comando return bool(re.match(r"^[A-Z]{3}[0-9]{3}\$", placa.upper())) se desglosara por partes ya que es fundamental en este inciso, el comando bool se encarga de convertir los resultados en formato booleano, true si coincide la información suministrada y false si esta no lo hace, el comando re.match(r"^[A-Z]{3}[0-9]{3}\$", placa.upper()), se encarga de por medio de expresiones regulares validar si la matrícula de la placa sigue un escritura específica de 3 letras mayúsculas seguidas desde la A hasta la Z y exactamente 3 dígitos seguidos con los números 0 y 9 el símbolo \$ representa el final de la cadena.

```
def registrar_usuario():
    print("\n--- REGISTRO DE USUARIO ---")

    nombre = input("Ingrese el nombre: ").strip()
    apellido = input("Ingrese el apellido: ").strip()
    documento = input("Ingrese el número de documento: ").strip()
    placa = input("Ingrese la placa del vehículo (ej. ABC123): ").strip().upper()
```

La función registrar usuarios empieza imprimiendo el encabezado inicial de registro de usuarios, indicando de manera gráfica al usuario que está en el proceso de registro de su vehículo para el ingreso al parqueadero, el comando de línea

nombre = input("Ingrese el nombre: ").strip() exige al cliente que ingrese su nombre, la parte de la línea .strip() se encarga de eliminar espacios es blanco al principio y al final para prevenir errores en la escritura de la información, la línea apellido = input("Ingrese el apellido: ").strip(), se caracteriza por cumplir la misma función que el campo anterior esta vez con la diferencia de que la información introducida es el apellido, utilizando de igual manera .strip() para eliminar espacios. La línea documento = input("Ingrese el número de documento: ").strip() se encarga de capturar el número de identificación del usuario. por último, la línea de código placa = input("Ingrese la placa del vehículo (ej. ABC123): ").strip().upper() se encarga de exigir la placa del vehículo, de igual manera como las anteriores .strip() se encarga de limpiar el espacio, el comando. upper() cumple la tarea de transformar las letras en mayúscula para asegurarse de la condición sobre la forma de escritura de las placas mantenido un nivel de consistencia.

```
errores = []

if not validar_nombre(nombre):
    errores.append(" El nombre debe tener al menos 3 letras y no contener números.")
```

```
if not validar_nombre(apellido):
    errores.append(" El apellido debe tener al menos 3 letras y no contener números.")
```

```
if not validar_documento(documento):
    errores.append(" El documento debe contener entre 3 y 15 dígitos numéricos.")
```

```
if not validar_placa(placa):
    errores.append(" La placa debe tener el formato correcto: 3 letras seguidas de 3 números.")
```

```
if errores:
    print("\n".join(errores))
    return
```

Ahora procederemos a explicar esta parte del módulo la cual empieza con el comando errores [], el cual es una lista vacía llamada errores, su principal función se centra en recopilar y abarcar todos los posibles errores que se hallan dentro de la validación. Cada línea de if not cumple con la tarea de realizar una verificación usando funciones importantes, que se han definido con antelación como lo son validar\_nombre, validar\_documento. se resalta cada una de estas líneas de código;

- if not validar\_nombre(nombre): errores.append(" El nombre debe tener al menos 3 letras y no contener números."); cumple con la tarea de comprobar si nombre ingresado por el usuario es válido o no, si el usuario pone una información incorrecta, este error se agrega a la lista de errores.
- if not validar\_nombre(apellido): errores.append(" El apellido debe tener al menos 3 letras y no contener números."); Cumple la misma tarea que el campo anterior pero esta vez con la información del apellido.
- if not validar\_documento(documento): errores.append(" El documento debe contener entre 3 y 15 dígitos numéricos."); Se encarga de validar que el documento del usuario este dentro del rango ya mencionado con anterioridad de entre 3 y 15 caracteres numéricos.
- if not validar\_placa(placa): errores.append(" La placa debe tener el formato correcto: 3 letras seguidas de 3 números."); Su función es comprobar si la información del usuario sobre la placa de su vehículo tenga el formato ya establecido de 3 letras mayúsculas, seguidas de 3 números. El comando append() cumple la tarea de registrar los errores que se detectan durante la validación de la información agregando elementos a lista errores permitiendo un resumen de los errores que se cometan en la verificación.
- if errores: print("\n".join(errores)) return; Se centra en verificar el contenido de la lista de errores, si hay por lo menos un error se ejecuta inmediatamente el comando print("\n".join(errores)) mostrando todos los errores en pantalla, el comando "\n".join(lista) es una forma ordenada de unir los textos de una lista.

```
with open(RUTA_USUARIOS, mode="a", newline="", encoding="utf-8") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow([documento, nombre, apellido, placa])
    print(f" Usuario registrado exitosamente.")

def menu_usuarios():
    registrar_usuario()
```

""Este fragmento final de código guarda los elementos de los datos del usuario recopilados con anterioridad en un archivo CSV, además introduce un menú para registro de usuarios, se desglosará la fracción del módulo para abarcar la explicación de cada parte del código.

- with open(RUTA\_USUARIOS, mode="a", newline="", encoding="utf-8") as archivo: Su función es abrir el archivo CSV en modo "a" para agregar nuevas listas sin alterar ni borrar el contenido ya existente.
- RUTA\_USUARIOS debe ser una variable con la ruta al archivo, por ejemplo "usuarios.csv".
- newline="" Previene el ingreso de listas vacías al sistema del parqueadero.

- encoding="utf-8") as archivo: Su finalidad es bastante útil en cuanto a la lectura de la información ya que asegura que tildes y caracteres especiales se manejen de forma correcta.
- escritor = csv.writer(archivo); Su función se centra por medio de la librería csv facilitar escribir filas en formato csv, creando por así decirlo un objeto escritor, permitiendo traducir lista como [documento, nombre, apellido, placa], en líneas de texto separadas por una coma.
- escritor.writerow([documento, nombre, apellido, placa]); Se centra en escribir una fila nueva en el archivo CSV con los datos ingresados por un usuario, encadenando así nuevos elementos de la lista que serán una columna en el archivo.
- print (f" Usuario registrado exitosamente.") Transmite un mensaje exitoso sobre el registro de la información que ha introducido el usuario, una vez los datos del mismo han sido guardados de manera correcta.
- def menu\_usuarios (): registrar\_usuario (); Por último, esta función se centra en declarar una función nueva de menú sin opciones visibles para este momento del código, su finalidad simplemente es llamar a registrar\_usuario () directamente.

En conclusión este módulo permite el procesamiento completo sobre el registro de usuarios en el parqueadero de una forma estructural y sencilla, realizando verificaciones de información como el nombre, documento, apellido y placa, almacenando esta información en un archivo CSV, respaldando así la información, además otorga información sencilla y clara sobre los errores así como la aplicación de buenas prácticas de programación como el buen manejo de rutas convirtiéndose de esta manera en una parte fundamental sobre el desarrollo del sistema del parqueadero.

## Módulo de operaciones centrales del parqueadero

```
import os
import csv
from datetime import datetime

RUTA_USUARIOS = os.path.join("src", "datos", "usuarios.csv")
RUTA_INGRESOS = os.path.join("src", "datos", "ingresos.csv")
RUTA_RETIROS = os.path.join("src", "datos", "retiros.csv")
MAX_CELDAS = 50
```

```
def buscar_usuario(documento):
```

```
try:

    with open(RUTA_USUARIOS, mode="r") as archivo:
        lector = csv.reader(archivo)
        for fila in lector:
            if fila[0] == documento:
                return True
except FileNotFoundError:
    pass
return False
```

```
def obtener_celda_disponible():

    ocupadas = set()

    try:
        with open(RUTA_INGRESOS, mode="r") as archivo:
            lector = csv.reader(archivo)
            for fila in lector:
                ocupadas.add(int(fila[1]))
    except FileNotFoundError:
        pass
```

```
for celda in range(1, MAX_CELDAS + 1):
    if celda not in ocupadas:
        return celda
return None
```

```
def liberar_celda(celda):
    pass
```

```
def calcular_tiempo_y_costo(inicio, fin):
    total_min = int((fin - inicio).total_seconds() // 60)
    horas = total_min // 60
    cuartos = (total_min % 60) // 15
    total = (horas * 7000) + (cuartos * 1500)
    if total < 7000:
        total = 7000
```

```
    return horas, cuartos, total
```

```
def exportar_datos():
    print("📄 Datos exportados automáticamente.")
```

Para iniciar la descripción paso por paso de este módulo empezaremos describiendo nuevamente las librerías, unas librerías ya se han definido con antelación, pero en cada módulo se va especificando que importancia e impacto tienen sobre el módulo de estudio.

```
import os
import csv
from datetime import datetime
```

- La librería os permite relacionar e interactuar con el sistema operativo del parqueadero, teniendo como función principal en este módulo manejar rutas de archivos y directorios. Su impacto en el módulo radica en la construcción de rutas dinámicas un ejemplo de esto sería (`os.path.join("src", "datos")`), garantizando de esta manera un buen manejo del sistema además de que permite que el código tenga un óptimo funcionamiento en diferentes sistemas operativos.
- csv: su función se centra en ayudar en cuanto a la lectura de archivos CSV, su función en este módulo radica en almacenar y recuperar datos y archivos sobre usuarios, retiros e ingresos por medio de estudio de archivos como `usuarios.csv` e `ingresos.csv`.
- datetime; Su función es registrar el cálculo sobre el manejo de fechas y horas, además de la duración de estancia de los vehículos de los usuarios utilizando el servicio del parqueadero.

```
RUTA_USUARIOS = os.path.join("src", "datos", "usuarios.csv")
RUTA_INGRESOS = os.path.join("src", "datos", "ingresos.csv")
RUTA_RETIROS = os.path.join("src", "datos", "retiros.csv")
MAX_CELDAS = 50
```

### Definición de rutas de archivo:

RUTA\_USUARIOS, RUTA\_INGRESOS y RUTA\_RETIROS utilizan `os.path.join()` Con la finalidad de crear nuevas rutas multiplataformas que se relacionan con los archivos CSV el cual es la carpeta que los almacena.

- Usuarios.csv se centra en el manejo de la información de los usuarios (documento, placa, apellido, nombre).
- ingresos.csv; Su función es el registro de vehículos estacionados (hora de ingreso, placa, documento).

- retiros.csv; Se centra en llevar la cuenta del historial de transacciones (documento, celda, placa, pago).
- Max celdas = Se centra en definir la capacidad del parqueadero la cual es de 50 celdas.

Las tres rutas funcionan siguiendo la función src/datos/ lo cual permite tener un estructura miento sobre el manejo del modelo. Esta parte del código ofrece consistencia, y mantebilidad sobre el sistema del parqueadero.

```
def buscar_usuario(documento):
    try:
        with open(RUTA_USUARIOS, mode="r") as archivo:
            lector = csv.reader(archivo)
            for fila in lector:
                if fila[0] == documento:
                    return True
    except FileNotFoundError:
        pass
    return False
```

La función buscar documento se centra en la autenticación y verificación sobre los usuarios que están registrados en el sistema, antes de la utilización de los servicios del parqueadero.

## Flujo de operaciones

# SMARTPARKING

Apertura del archivo usuario.csv (definido por RUTA\_USUARIOS) centrado en modo lectura "r" utilizando un bloque try-except con la finalidad de mejorar el archivo en situaciones o escenarios donde no existe aún. Se procede en utilizar un ciclo with para lograr que el archivo RUTA\_USUARIOS se cierre inmediatamente después de su uso.

- lector = csv.reader(archivo); Busca crear un lector CSV que interactúe con el archivo mediante una lista de filas.
- for fila in lector: El ciclo for funciona para recorrer cada fila del archivo CSV. Cada fila es una lista de cadenas, que se centra en cumplir la función de brindar el documento, nombre, placa.
- if fila[0] == documento: Se centra en comparar el primer espacio de cada fila por eso el elemento [0] que en este caso sería siendo el espacio documento. return True; La función retorno true busca dar indicaciones de que si es verdadero el usuario existe en el sistema del parqueadero.

- except FileNotFoundError: pass La función de esta línea de código se centra en tal caso de que el archivo no exista, se omite el error con pass.
- return False; Si no se encuentra el documento en ninguna fila indica que el usuario simplemente no está registrado en el sistema del parqueadero.

```
def obtener_celda_disponible():
    ocupadas = set()
    try:
        with open(RUTA_INGRESOS, mode="r") as archivo:
            lector = csv.reader(archivo)
            for fila in lector:
                ocupadas.add(int(fila[1]))
    except FileNotFoundError:
        pass
    for celda in range(1, MAX_CELDAS + 1):
        if celda not in ocupadas:
            return celda
    return None
```

La función obtener\_celda\_disponible() se enfoca en ubicar una celda del parqueadero la cual este libre para un nuevo vehículo. def obtener\_celda\_disponible(): Su tarea es rastrear en el intervalo de las 50 celdas una libre en el parqueadero. La línea 2 es "ocupadas = set()" La cual es un conjunto vacío llamado "ocupadas" con el propósito de guardar los números de las celdas que ya se encuentran llenas, se usa set (), el cual cumple la tarea de buscar celdas sin repetir celdas duplicadas. En la línea 3 se crea un bloque try para apresar errores.

- with open(RUTA\_INGRESOS, mode="r") as archivo: Primero abre el archivo RUTA: INGRESOS en modo lectura, para registrar la información sobre los vehículos que ha entrado y en que celda están ubicados.
- lector = csv.reader(archivo); esta fila 5 esta para crear un lector CSV el cual permita recorrer las filas del archivo.
- Las filas 6 7 for fila in lector: ocupadas.add(int(fila[1])); La funcionalidad de las dos filas de es extraer por medio de fila [1] las celdas ocupadas del parqueadero. Se prioriza convertirse en entero y se procede a guardarse en el conjunto de ocupadas.
- except FileNotFoundError:

Pass; Las líneas 8-9 respectivamente permiten demostrar que, si el archivo no existe, quiere decir que todavía no ha ingresado ningún vehículo al establecimiento, ignorando el error, permitiendo que el conjunto "ocupadas" permanezca vacío.

- for celda in range(1, MAX\_CELDAS + 1): Esta línea de código permite recorrer todas las celdas del parqueadero desde la 1 hasta la 50 por medio del comando MAX\_CELDAS.
- if celda not in ocupadas:  
return celda: Permite ubicar celdas que no están en el conjunto de ocupadas, inmediatamente la devuelve como celda disponible.
- return None; Permite establecer de que, si todas las celdas del parqueadero se encuentran ocupadas, se retorna a None, lo que permite identificar de que ya no hay espacios libres en el parqueadero de la universidad.

```
def liberar_celda(celda):
    pass

def calcular_tiempo_y_costo(inicio, fin):
    total_min = int((fin - inicio).total_seconds() // 60)
    horas = total_min // 60
    cuartos = (total_min % 60) // 15
    total = (horas * 7000) + (cuartos * 1500)
    if total < 7000:
        total = 7000
    return horas, cuartos, total
```

La función calcular\_tiempo\_y\_costo (inicio, fin): permite calcular el lapso de tiempo que un vehículo estuvo estacionado utilizando los servicios del parqueadero, además permite establecer el precio que debe pagar por las prestaciones de servicio del parqueadero. Vamos a describir línea por línea.

- def calcular\_tiempo\_y\_costo (inicio, fin):

Línea 1: Se centraliza en definir una función la cual expresa dos parámetros, los cuales son inicio y fin siendo ambos objetos de datetime, su funcionalidad se especifica en decir la hora en el vehículo tiene un ingreso al parqueadero, y la hora en que el vehículo sale del parqueadero.

- Total\_min = int((fin - inicio).total\_seconds() // 60):

Línea 2: Su función se centraliza en el cálculo del tiempo total que ha pasado en minutos, el comando "fin-inicio" da un timedelta, seguido del comando. total\_second() el cual permite convertir el tiempo transcurrido a segundos, "// 60 " convierte el tiempo a minutos, y el comando que da inicio "int" permite establecer los números como enteros.

- horas = total\_min // 60:

Línea 3; Permite establecer la función de cuantas horas completas ha transcurrido de tiempo en el parqueadero.

- cuartos = (total\_min % 60) // 15

Línea 4; establece cuantos cuartos de hora se establecen en el tiempo de uso del parqueadero.

- total = (horas \* 7000) + (cuartos \* 1500):

Línea 5; Establece los valores a pagar por tiempo prestablecido, siendo cada cuarto de hora 1500 pesos colombianos, la hora cuesta 7000 pesos colombianos.

- if total < 7000:

total = 7000

Línea 6-7 lo que permite este código es el cálculo sobre si el total calculado es menor a 7000, se centraliza en cobrar como mínimo 7000 pesos colombianos. Este comando se especializa en tal caso en clientes que utilizan las prestaciones del parqueadero por muy corto tiempo.

- return horas, cuartos, total

Línea 8: Permite retorar los valores de horas, cuartos, total, permitiendo establecer parámetros como cantidad de horas completadas, cuartos de hora completados, y por último el total a pagar por los servicios del parqueadero.

```
def exportar_datos():
    print("📁 Datos exportados automáticamente.")
```

- def exportar\_datos ():

Línea 1: Permite definir una función llamada exportar:datos(): la cual tiene la tarea de exportar información del sistema del parqueadero.

- print ("📁 Datos exportados automáticamente.")

Línea 2: Permite establecer un mensaje informativo sobre imprimir un mensaje el cual exporta los datos.

En conclusión, este módulo permite establecer una solución funcional y escalonado el cual radica en el buen gestionamiento sobre tareas básicas como el ingreso, la asignación y salida de los automóviles de los usuarios. Por medio de funciones como buscar usuario, obtener\_celda\_disponible, calcular\_tiempo\_y\_costo, liberar celda, permite la obtención sobre un registro detallado sobre los usuarios que quieren acceder, la identificación de las celdas libres, además del cálculo a pagar sobre la estancia y la prestación de servicio del parqueadero.

## Módulo gestión de ingreso y retiros de vehículos

```
import csv
import os
from datetime import datetime
```

```
from utilidades import calcular_tiempo_y_costo, buscar_usuario, obtener_celda_disponible,  
liberar_celda, RUTA_INGRESOS, RUTA_RETIROS, MAX_CELDAS
```

```
def menu_vehiculos(accion):  
    if accion == "ingreso":  
        ingreso_vehiculo()  
    elif accion == "retiro":  
        retiro_vehiculo()
```

```
def ingreso_vehiculo():  
    print("\n--- INGRESO DE VEHÍCULO ---")  
    documento = input("Ingrese el documento del usuario: ").strip()
```

```
if not buscar_usuario(documento):  
    print(" El usuario no está registrado.")  
    return
```

```
celda = obtener_celda_disponible()  
if celda is None:  
    print(" No hay espacios disponibles.")  
    return
```

```
hora_ingreso = datetime.now()  
datos = [documento, celda, hora_ingreso.strftime("%Y-%m-%d %H:%M:%S")]
```

```
with open(RUTA_INGRESOS, mode="a", newline="") as archivo:  
    escritor = csv.writer(archivo)  
    escritor.writerow(datos)
```

```
    print(f" Vehículo ingresado en la celda {celda} a las {hora_ingreso.strftime('%H:%M')}.)")
```

```
def retiro_vehiculo():  
    print("\n--- RETIRO DE VEHÍCULO ---")  
    documento = input("Ingrese el documento del usuario: ").strip()
```

```
registros = []
encontrado = None
```

```
with open(RUTA_INGRESOS, mode="r") as archivo:
    lector = csv.reader(archivo)
    for fila in lector:
        if fila[0] == documento:
            encontrado = fila
        else:
            registros.append(fila)
```

```
if not encontrado:
    print(" No se encontró un vehículo registrado con ese documento.")
    return
```

```
celda = encontrado[1]
hora_ingreso = datetime.strptime(encontrado[2], "%Y-%m-%d %H:%M:%S")
hora_salida = datetime.now()

horas, cuartos, total = calcular_tiempo_y_costo(hora_ingreso, hora_salida)
```



```
with open(RUTA_RETIROS, mode="a", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow([documento, celda, hora_ingreso.strftime("%Y-%m-%d %H:%M:%S"),
                      hora_salida.strftime("%Y-%m-%d %H:%M:%S"), horas, cuartos, total])
```

```
with open(RUTA_INGRESOS, mode="w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerows(registros)
```

```
liberar_celda(celda)
print(f" Vehículo retirado. Total a pagar: ${total:.0f} COP")
```

Para dar por finalizado la explicación sobre los elementos que componen la estructura de los módulos, se procederá a explicar el último módulo el cual se centraliza en el manejo administrativo sobre el manejo del movimiento vehicular del parqueadero.

```
import csv
import os
from datetime import datetime
from utilidades import calcular_tiempo_y_costo, buscar_usuario, obtener_celda_disponible,
liberar_celda, RUTA_INGRESOS, RUTA_RETIROS, MAX_CELDAS
```

import csv; su función habilita leer el archivo en un formato csv, principalmente se utiliza en este caso del parqueadero para tener ingresos y retiros del vehículo. Import os permite la verificación sobre archivos, el renombramiento de archivos, o obtener rutas.

From datetime principalmente es utilizada para el registro de fechas y horas sobre el ingreso del automóvil hacia las instalaciones del parqueadero, a así como también permite el cálculo sobre el cobro por los servicios prestados por parte del parqueadero.

from utilidades import calcular\_tiempo\_y\_costo, buscar\_usuario, obtener\_celda\_disponible, liberar\_celda, RUTA\_INGRESOS, RUTA\_RETIROS, MAX\_CELDAS; La función de esta línea principalmente radica en importar funciones desde el módulo "utilidades", cada elemento importado cumple ciertos requisitos en específico;

- calcular\_tiempo\_y\_costo: Permite el cálculo de tiempo sobre las horas y cuartos de hora, así como establece cuanto debe pagar cada automóvil.
- buscar\_usuario: Su función dictamina si el usuario ingresado pertenece a un usuario que está registrado en el sistema del parqueadero.
- obtener\_celda\_disponible: Se centra en rastrear celdas libres devolviendo el número de la primera celda libre que se ubique primero.
- liberar\_celda: Se especializa en eliminar las celdas que ya se encuentran ocupadas por los usuarios.
- RUTA\_INGRESOS: Permite establecer la ruta(archivos) donde se guardó con antelación los registros del ingreso vehicular.
- RUTA\_RETIROS: ruta(archivos) la cual establece los vehículos que se han retirado de las instalaciones del parqueadero.
- MAX\_CELDA: Establece el número de celdas máximo que hay en el parqueadero.

```
def menu_vehiculos(accion):
    if accion == "ingreso":
```

```
    ingreso_vehiculo()
elif acción == "retiro":
    retiro_vehiculo()
```

Esta función ejecuta el control central el cual se centra en plantear funciones que el usuario eligió con anterioridad, se procederá a describir línea por línea de esta función.

- def menu\_vehiculos(acción):

Línea 1: Esta función define un comando llamado menu\_vehiculos que tiene como parámetro principal (acciones). Esta cadena de texto se centra en indicar las operaciones que el usuario quiere ejecutar en cuanto al parqueadero "ingreso", "retiro".

- if acción == "ingreso":  
 ingreso\_vehiculo()

Línea 2-3: Se centra en la activación en cuanto al registro de vehículos al recinto del parqueadero, centrándose en validar el usuario, y asignando una celda disponible si este se encuentra ya registrado con anterioridad.

- elif acción == "retiro":  
 retiro\_vehiculo()

Línea 4-5: Se centra en identificar el proceso de validación de retiro del automóvil, elif permite evaluar esta condición estableciendo si fue falsa. Estas dos líneas de código establecen también el tiempo de estadía y el valor a pagar. Si acción == "ingreso": es verdadera se ejecuta retiros\_vehicular(), si no establece otro elif.

```
def ingreso_vehiculo():
    print("\n--- INGRESO DE VEHÍCULO ---")
    documento = input("Ingrese el documento del usuario: ").strip()

    if not buscar_usuario(documento):
        print(" El usuario no está registrado.")
        return
```

```
    celda = obtener_celda_disponible()
    if celda is None:
        print(" No hay espacios disponibles.")
        return
```

```
    hora_ingreso = datetime.now()
```

```
datos = [documento, celda, hora_ingreso.strftime("%Y-%m-%d %H:%M:%S")]

with open(RUTA_INGRESOS, mode="a", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(datos)
```

```
print(f" Vehículo ingresado en la celda {celda} a las {hora_ingreso.strftime('%H:%M')}).")
```

La función ingreso vehicular se especializa en el registro sobre el ingreso vehicular sobre el parqueadero, a continuación, se procederá a desglosar línea por la línea de este fragmento de código.

- def ingreso\_vehiculo():

Línea 1: Su tarea se destaca principalmente en el registro y la entrada de nuevos vehículos, además también se recalca que se centra en asociar a un usuario registrado, y por consiguiente se especifica en buscar una celda aleatoriamente que se encuentre libre para ser ocupada.

- print("\n--- INGRESO DE VEHÍCULO ---")

Línea 2: Su tarea radica en imprimir un título con la finalidad indicar que el proceso de ingreso del automóvil al recinto a comenzado.

- documento = input("Ingrese el documento del usuario: ").strip()

Línea 3: Se centra en pedirle al usuario que digite su documento con el propósito de verificación de este, el comando. strip() se coloca para que cumpla la tarea de eliminar los espacios sobre el texto ingresado.

- if not buscar\_usuario(documento):

```
print (" El usuario no está registrado.")
```

```
Return
```

Líneas 4-5-6; Se llama a la función buscar\_usuario(documento) con el propósito de verificar si la persona que desea ingresar esta registrado o no dentro del sistema. El parámetro documento recalca el uso de la verificación por medio de este mismo. el comando "if not" el not niega el resultado de buscar\_usuario, esto nos quiere decir que si hay un error en la validación se devuelve. Por último, print muestra si el usuario no está registrado, el return funciona para evitar el sobre exceso de código.

- celda = obtener\_celda\_disponible()

```
if celda is None:
```

```
print(" No hay espacios disponibles.")
```

```
Return
```

Líneas 7-8-9-10: se llama a la función obtener\_disponible() con el propósito de centrarse en obtener una celda libre en las instalaciones del parqueadero, La función retorna un valor para verificar si hay espacios vacíos disponibles o None si no los hay dentro del parqueadero. if celda is None busca verificar si se obtuvo la

celda disponible, el comando None se encarga de mostrar las celdas que no están disponibles. El print se encarga de mostrar el mensaje de no hay espacios disponibles dentro de las instalaciones del parqueadero, el return es útil para definir para dar por finalizado el proceso del código.

- hora\_ingreso = datetime.now()

Línea 11: Su función se especifica en tomar la hora actual del sistema, con esto se busca registrar esa hora como la hora preestablecida de entrada del vehículo.

- datos = [documento, celda, hora\_ingreso.strftime("%Y-%m-%d %H:%M:%S")]

Línea 12; Con esta línea de código se busca preparar los datos los cuales serán guardados en un archivo CSV. El documento funciona como identificador del usuario, celda, se centra en la asignación de espacio del vehículo, obtenida con antelación, por la función obtener\_celda\_disponible(), horas\_ingreso.strftime("%Y-%m-%d %H:%M:%S"), este comando se especializa en tomar un objeto datetime y transformarlo, en una cadena la cual integra y conforma el formato "año-mes-día-hora:minutos;segundo".

- with open(RUTA\_INGRESOS, mode="a", newline="") as archive:

```
escritor = csv.writer(archivo)
```

```
escritor.writerow(datos)
```

Línea 13-14-15: with open(RUTA\_INGRESOS, mode="a", newline="") as archivo: Se encarga de abrir el archivo RUTA\_INGRESOS con el comando append "a" agregar al final. El mode="a" se encarga de la añadidura de nuevas filas, el comando newline="" Se encarga principalmente de prevenir que se sobre escriban o inserten líneas en blanco en los registros preestablecidos del sistema operativo. La línea escrita = csv.writer(archivo) se encarga de la creación de un objeto con la finalidad manipular el archivo en formato CSV. escritor.writerow(datos), esta línea de comando busca escribir una fila con los elementos de la lista datos. Para finalizar se recalca el ciclo with para permitir que el archivo una vez terminado el proceso se cierre de manera automática.

- print(f" Vehículo ingresado en la celda {celda} a las {hora\_ingreso.strftime('%H:%M')}")."

Línea 16: El print se encarga de mostrar un mensaje al usuario, f ("") se un f-string, y se encarga de introducir variables con formato de una cadena de texto, {celda} es un comando el cual se encarga de ser remplazada con una celda asignada a un vehículo, {hora\_ingreso.strftime('%H:%M')},"), busca transformar la hora en un formato integral en horas y minutos.

```
def retiro_vehiculo():

    print("\n--- RETIRO DE VEHÍCULO ---")

    documento = input("Ingrese el documento del usuario: ").strip()

    registros = []
```

```

encontrado = None

with open(RUTA_INGRESOS, mode="r") as archivo:
    lector = csv.reader(archivo)
    for fila in lector:
        if fila[0] == documento:
            encontrado = fila
        else:
            registros.append(fila)

if not encontrado:
    print(" No se encontró un vehículo registrado con ese documento.")
    return

```

Esta función se encarga principalmente sobre el proceso de salida y entrada de vehículos del establecimiento del parqueadero.

- def retiro\_vehiculo():

Línea 1: Se plantea la función retiros\_vehiculos su función en el módulo se centra en retirar un vehículo del sistema.

- print("\n--- RETIRO DE VEHÍCULO ---")

Línea 2: \n se encarga de dejar un espacio en blanco para imprimir el mensaje RETIRO DE VEHICULO.

- documento = input("Ingrese el documento del usuario: ").strip()

Línea 3: Se encarga principalmente de solicitar al usuario su documento, el comando strip busca se ocupa de eliminar espacios en blanco sobre la información proporcionada por el usuario.

- registros = []

encontrado = None

Línea 4-5: registro =[] es una lista vacía la cual prioriza la función de almacenar los registros que no serán borrados, esto quiere decir los números de documentos los cuales son erróneos o no coinciden con el documento proporcionado por el usuario. El comando encontrado = None, es una variable la cual busca almacenar un registro, se implementa en esta línea el None para los vehículos los cuales no se han registrado en el sistema del parqueadero por lo tanto su información no es verificable.

- with open(RUTA\_INGRESOS, mode="r") as archivo:

lector = csv.reader(archivo)

```

for fila in lector:
    if fila[0] == documento:
        encontrado = fila
    else:
        registros.append(fila)

```

Línea 6-12: Con esta parte del código se empieza por abrir el archivo RUTA\_INGRESOS en modo "r" el cual es en modo lectura, el comando csv.reader cumple la tarea de recorrer cada fila como una lista. El comando fila[0], permite establecer que la primera columna en este caso 0 contiene un documento de usuario, si el documento tiene coincidencia con la verificación se busca guardar la fila en encontrado, en tal caso de que esta no coincida se busca un caso alternativo y se guarda en registros.

- if not encontrado:

```
print(" No se encontró un vehículo registrado con ese documento.")
```

```
Return
```

Línea 13-16; busca establecer la verificación si se encontró o no el documento en el registro. Si no se encuentra el documento dado por el usuario se informa al usuario por medio de imprimir el mensaje No se encontró el vehículo registrado con ese documento, se procede a salir de la función y se retorna el comando retorno permite evitar errores de código.

```

celda = encontrado[1]
hora_ingreso = datetime.strptime(encontrado[2], "%Y-%m-%d %H:%M:%S")
hora_salida = datetime.now()

horas, cuartos, total = calcular_tiempo_y_costo(hora_ingreso, hora_salida)

```

Este bloque de código pertenece al flujo encargado de calcular el intervalo de tiempo enfocado sobre el ingreso de vehículos al sistema del parqueadero además de cuanto debe pagar el usuario por las prestaciones del servicio, se priorizo en que sea integral y legible.

- celda = encontrado[1]

Línea 1: Se centra en extraer la celda disponible del parqueadero para el registro encontrado y verificado. Siguiendo la lógica del código se establece que está en la segunda columna fila [1].

- hora\_ingreso = datetime.strptime(encontrado[2], "%Y-%m-%d %H:%M:%S")

Línea 2: Se centra en convertir el texto de la hora en un tiempo a un objeto datetime, esto nos facilita para poder convertir el tiempo como ya se había dicho, un ejemplo de esto sería "2025-06-12 16:15:00"

- hora\_salida = datetime.now()

Línea 3: Permite tener un registro sobre la hora del vehículo.

- horas, cuartos, total = calcular\_tiempo\_y\_costo(hora\_ingreso, hora\_salida)

Línea 4: Se centra en llamar a la función calcular\_tiempo\_y\_costo con el propósito de calcular la hora de entrada y salida del usuario con su automóvil, calculando, las horas, cuarto de horas, y el total del valor a pagar por el uso del parqueadero en el lapso de tiempo preestablecido.

```
with open(RUTA_RETIROS, mode="a", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow([documento, celda, hora_ingreso.strftime("%Y-%m-%d %H:%M:%S"),
                      hora_salida.strftime("%Y-%m-%d %H:%M:%S"), horas, cuartos, total])
```

"Vamos a desglosar línea por línea este bloque de código el cual tiene la tarea de registrar el retiro de los vehículos del establecimiento del parqueadero.

- with open(RUTA\_RETIROS, mode="a", newline="") as archivo:

Línea 1: Primero se abre el archivo en formato CSV RUTA\_RETIROS, en modo append "a" con el propósito de agregar y no borrar lo que ya está definido, esto significa agregar nuevos registros, pero también se centra en no borrar los ya establecidos. el comando newline="" Permite que con la información proporcionada no haya espacios en blanco, el ciclo with permite que el archivo se tenga un cierre de forma correcta tras finalizar.

- escritor = csv.writer(archivo)

Línea 2: Se crea un archivo en formato CSV para poder manejar de una mejor manera la escritura de las filas.

- escritor.writerow([documento, celda, hora\_ingreso.strftime("%Y-%m-%d %H:%M:%S"),
 hora\_salida.strftime("%Y-%m-%d %H:%M:%S"), horas, cuartos, total])

Línea 3-4: El comando escritor.writerow se centra en escribir una fila totalmente nueva con los datos del usuario sobre el retiro del establecimiento del parqueadero.

- documento: identificación del usuario. -celda: El número de celda que ocupó su vehículo.
- hora\_ingreso: Establece el formato de fecha y la hora en que el vehículo ingresó al establecimiento.
- hora\_salida: Establece la fecha y la hora en la cual se está realizando el registro de salida.
- cuartos: Mide el cuarto de hora permitiendo cobrar en un lapso de tiempo corto.
- total: Establece el total a pagar por la utilización del parqueadero en un intervalo de tiempo.

```
with open(RUTA_INGRESOS, mode="w", newline="") as archivo:
    escritor = csv.writer(archivo)
```

```

    escritor.writerows(registros)

liberar_celda(celda)
print(f" Vehículo retirado. Total a pagar: ${total:.0f} COP")

```

Se procederá a explicar esta última parte del módulo la cual se centraliza en reestructurar y actualizar los datos del sistema borrando los datos del vehículo que ha acabado de salir, y así permitiendo liberar una celda nueva.

- with open(RUTA\_INGRESOS, mode="w", newline="") as archivo:

Línea 1: Esta primera línea de este bloque procede a abrir el archivo RUTA\_INGRESOS, el comando mode = "w" quiere decir que se abre en modo escritura, esto quiere decir que con este comando en modo escritura se procede a sobrescribir su contenido.

- escritor = csv.writer(archivo).

escritor = csv.writer(archivo) Línea 2: Se centra en crear por medio de parámetro archivo un escritor CSV.

- escritor.writerows(registros)

Línea 3: Su tarea es escribir todas las filas teniendo excepción sobre el vehículo retirado. La lista llamada registros fue construida con anterioridad para cumplir con este propósito.

- liberar\_celda(celda)

Línea 4: La tarea de esta línea es llamar a la función liberar\_celda, centrándose en borrar la celda de una estructura de ocupación, o por consiguiente marcar una celda disponible para un siguiente usuario.

- print(f" Vehículo retirado. Total, a pagar: \${total:.0f} COP")

Línea 5: Se encarga de mostrar un mensaje final hacia el usuario sobre la confirmación del retiro, además de incluir en este mensaje el monto a pagar en miles de pesos colombianos.

Para finalizar con la explicación y análisis del módulo se procede a dar una breve conclusión. Este módulo cumple una tarea fundamental ya que se encarga del control sobre el proceso de salida de las instalaciones del parqueadero, pasando por parámetros como la validación de documento, hasta el cálculo de dinero a pagar por el intervalo de estadía de tiempo. Se centra en mostrar primero el registro del vehículo en ingresos, tras esto calcula en tiempo transcurrido desde el ingreso, tras esto se ocupa de guardar toda la información sobre el usuario que acabo de salir en retiro mediante el archivo RUTA\_RETIROS, se procede a renovar el archivo ingresos ya con el usuario eliminado del sistema, eliminando de esta forma la celda la cual este usuario con su vehículo estaba ocupando. Este módulo establece una función fundamental para mantener un orden sobre el control de las rotaciones de los vehículos.

## 4. Interacción con el menú principal

El sistema Smart Parking es operado a través de un menú interactivo en la consola. Siga las instrucciones en pantalla para navegar por las opciones.

### Menú Principal

Al iniciar el programa, verá el siguiente menú:



```
+-----+  
| SMART PARKING - Universidad de Antioquia |  
+-----+  
| 1. Registrar usuario |  
| 2. Ingresar vehículo |  
| 3. Retirar vehículo |  
| 4. Modo administrador |  
| 5. Salir |  
+-----+  
Ingrese una opción: |  
  
+-----+  
| >> INGRESAR VEHÍCULO << |  
+-----+  
| Documento del usuario: 987654321 |  
| Placa del vehículo: XYZ456 |  
+-----+  
| ✓ Vehículo asignado a la CELDA #22 |  
| Hora de entrada: 09:15 a.m. |  
+-----+  
| [ RECIBO DE INGRESO ] |  
| Placa: XYZ456 | Celda: #22 |  
| Hora: 09:15 a.m. |  
+-----+  
| >> MODO ADMINISTRADOR << |  
+-----+  
| Usuario: admin |  
| Contraseña: |  
+-----+  
| 1. Reporte de vehículos |  
| 2. Lista de usuarios |  
| 3. Ocupación de celdas |  
| 4. Volver |  
+-----+  
  
+-----+  
| >> REGISTRAR USUARIO << |  
+-----+  
| Nombre: Ana María |  
| Apellido: Gómez |  
| Documento: 987654321 |  
| Placa (ABC123): XYZ456 |  
+-----+  
| ✓ ¡Usuario registrado exitosamente! |  
+-----+  
  
+-----+  
| >> RETIRAR VEHÍCULO << |  
+-----+  
| Documento del usuario: 987654321 |  
| Placa del vehículo: XYZ456 |  
+-----+  
| Tiempo estacionado: 4 horas 30 minutos |  
| Cobro:  
| - Horas completas (4 x $7000): $28000  
| - Cuartos de hora (2 x $1500): $3000  
| TOTAL A PAGAR: $31000 |  
+-----+  
| [ FACTURA ELECTRÓNICA ] |  
| #FAC-002 | Fecha: 01/05/2025 |  
| Placa: XYZ456 | Total: $31000 |  
+-----+  
  
+-----+  
| >> OCUPACIÓN DE CELDAS << |  
+-----+  
| Celdas ocupadas: 42/50 |  
| Celdas libres: 8 |  
| [Mapa de celdas] |  
| 01: [X] 02: [X] 03: [ ] 04: [X] ... |  
| ... |  
| 50: [X] |  
+-----+  
  
+-----+  
| >> EXPORTAR DATOS << |  
+-----+  
| ¿Generar reporte CSV? (S/N): S |  
| ✓ Archivo "reporte_smartparking_01-05-2025.csv" creado |  
| ¡Hasta pronto! |  
+-----+
```

### ILUSTRACIÓN 3

Ingrese el número correspondiente a la opción que desea realizar y presione Enter.

## 1. Registrar Usuario

Esta opción le permite añadir nuevos usuarios al sistema.

- Se le pedirá el nombre, apellido, número de documento y la placa del vehículo.
- Validaciones:
  - Nombre y Apellido: Deben tener al menos 3 letras y no contener números.
  - Documento: Debe contener entre 3 y 15 dígitos numéricos.
  - Placa: Debe tener un formato de 3 letras seguidas de 3 números (ej. (ABC123).
  - Si hay errores en la entrada, el sistema le informará cuáles son.
- Una vez registrado exitosamente, recibirá un mensaje de confirmación.

## 2. Ingresar Vehículo

Use esta opción para registrar la entrada de un vehículo al parqueadero.

- Se le pedirá el documento del usuario y la placa del vehículo.
- El sistema verificará que el usuario esté registrado.
- Validará la disponibilidad de celdas (el parqueadero tiene 50 espacios en total).
- Si hay espacio, se asignará automáticamente la celda libre más baja.
- Se mostrará un recibo con la hora y minuto de ingreso.

## 3. Retirar Vehículo

Esta opción permite registrar la salida de un vehículo y calcular el cobro.

- Se le solicitará el documento del usuario.
- El sistema calculará el tiempo que el vehículo estuvo estacionado (horas completas y cuartos de hora).
- Tarifas:
  - \$7,000 COP por hora completa.
  - \$1,500 COP por cada cuarto de hora.
  - El pago mínimo es de \$7,000 COP (equivalente a una hora completa), incluso si el tiempo estacionado es menor.
- Se mostrará un recibo de salida con el desglose de los cobros y el total a pagar.

## 5. Modo Administrador

Acceda a esta sección para ver reportes y estadísticas del parqueadero.

- Se requerirá un usuario y contraseña para acceder.
- Si las credenciales son correctas, tendrá acceso a los siguientes reportes:

- Total, de vehículos registrados.
- Reporte de vehículos retirados y pendientes (en el parqueadero).
- Informe del total de pagos recaudados.
- Cálculo del tiempo promedio de estancia.
- Identificación del vehículo con tiempo máximo y mínimo de estancia.
- Estado de ocupación por celda.

**NOTA:** Tener presente que el usuario y la contraseña para el administrador son las siguientes:

**Usuario:** admin

**Contraseña:** 1234

## 6. Salir y Exportar Datos

Al seleccionar esta opción, se le preguntará si desea generar un reporte en formato CSV.

- Si confirma (s ), el sistema exportará todos los datos de la jornada a un archivo CSV con un nombre como: **reporte\_smartparking\_DD-MM-YYYY.csv**
- El programa se cerrará después de esta acción.

## Trabajos citados

Daniel, T. J. (13 de 07 de 2025). *GitHub/TrabajoFinal*. Obtenido de Github:

<https://github.com/Danielmr03/TrabajoFinal>

Python. (s.f.). *Python*. Obtenido de Python: <https://www.python.org/>

