



**USAC**  
**TRICENTENARIA**  
Universidad de San Carlos de Guatemala

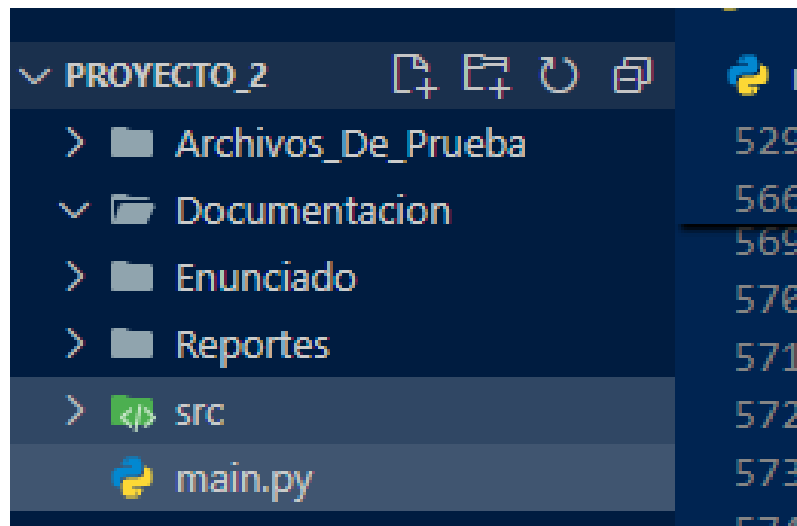
# **PROYECTO 2**

# **MANUAL TÉCNICO**

## **ESTRUCTURA DE DATOS**

Miguel Adrian Tubac Agustin  
202101927

## Descripción de la solución



El programa resuelve el problema de un sistema de control de vehículos, clientes y viajes. Los cuales se procesan en estructuras de datos implementadas en memoria, con la finalidad de mejorar la velocidad del control de los activos de la empresa Llega Rapidito.

## Lenguaje Utilizado:



Los requerimientos necesarios para la edición y ejecución del programa son la utilización del lenguaje de programación denominado Python, en el cual se crea la interfaz agradable con el usuario. Con el uso de la librería Tkinter.



# Clases

**Cliente:** en esta clase se definen las propiedades del cliente, así mismo, con las funcionalidades de modificación de los atributos.


```
class Cliente:
    def __init__(self, dpi:str, no
        self.dpi = dpi
        self.nombres = nombres
        self.apellidos = apellidos
        self.genero = genero
        self.telefono = telefono
        self.direccion = direccion
        self.cantidad_viajes = 0
```

```
class Ruta:
    def __init__(self, or
        self.origen:str =
        self.destino:str
        self.tiempo:int =
```

**Ruta:** en esta clase se definen las propiedades de las rutas que maneja el sistema, así como, los atributos de tiempo.

**Vehiculo:** en esta clase se definen las propiedades de los vehículos, esta clase cuenta con los métodos para modificar los atributos.

```
class Vehiculo:
    def __init__(self, placa
        self.placa = placa
        self.marca = marca
        self.modelo = modelo
        self.precio = precio
        self.cantidad_viajes
```



```
class Vertice:
    def __init__(self, valor: str, vecinos: ListaVecinos, peso: int, peso_acumulado: int, padre: Vertice):
        self.valor: str = valor
        self.vecinos: ListaVecinos = vecinos
        self.peso: int = peso
        self.peso_acumulado: int = peso_acumulado
        self.padre: Vertice = padre
```


**Vertice:** en esta clase se definen las propiedades de los vértices que componen al grafo de las rutas, en la misma se definen los datos necesarios.

**Viaje:** en esta clase se definen las propiedades de los viajes, en el cual se almacenan los datos que un usuario realice en el sistema.

```
class Viaje:
    def __init__(self, id: int, origen: str, destino: str, fecha: str, cliente: Cliente, vehiculo: Vehiculo):
        self.id: int = id
        self.origen: str = origen
        self.destino: str = destino
        self.fecha: str = fecha
        self.cliente: Cliente = cliente
        self.vehiculo: Vehiculo = vehiculo
```

```
class ArbolB:
    def __init__(self, orden: int):
        self.raiz: NodoArbolB = None
        self.orden: int = orden
```

**ArbolB:** en esta clase se definen las propiedades del ArbolB, en donde se encuentran los métodos necesarios para la creación del mismo.



**Cola:** en esta clase se definen las propiedades de la cola, la cual es utilizada para obtener la ruta más corta.

```
class Cola():#Esta es una  
    def __init__(self):  
        self.cabeza = None
```

```
class ListaAdyacencia:  
    def __init__(self):  
        self.vertices:  
    def insertar(self,
```

**ListaAdyacencia:** en esta clase se definen las propiedades que conformaran el grafo, ya que el mismo proviene de una lista de Adyacencia.

**Lista:** en esta clase se definen los métodos necesarios para la implementación de una lista enlazada simple.

```
class Lista:  
    def __init__(self):  
        self.cabeza:  
    def insertar_fina  
        aux: Nodo = s
```

# Métodos Y Funciones

```
def eliminar(self, placa: str)
    """
    Elimina un vehículo del árbol
    :param placa: Placa del vehículo
    :return: True si se eliminó
    """
    if not self.raiz.claves:
```

**Eliminar:** este método se encarga de eliminar un vehículo del Arbol B, para lo cual necesita del parámetro de la placa del vehículo

**Obtener\_ruta:** con este método se obtiene la ruta más corta del grafo a partir de la ciudad origen a la ciudad destino.

```
def obtener_ruta(self, ori
    ruta:Lista[Vertice] =
    nodos_visitados:Cola =
    nodos:Cola = Cola()

    # Aca se busca el vert
    original:Vertice = cop
```

```
def cargar_archivo_vehiculos(
    archivo = filedialog.asko
    if not archivo:
        print("No se seleccio
        messagebox.showinfo("
        return
```

**Cargar\_archivo\_vehiculos:** este método nos permite exportar los datos del archivo de entrada al sistema.