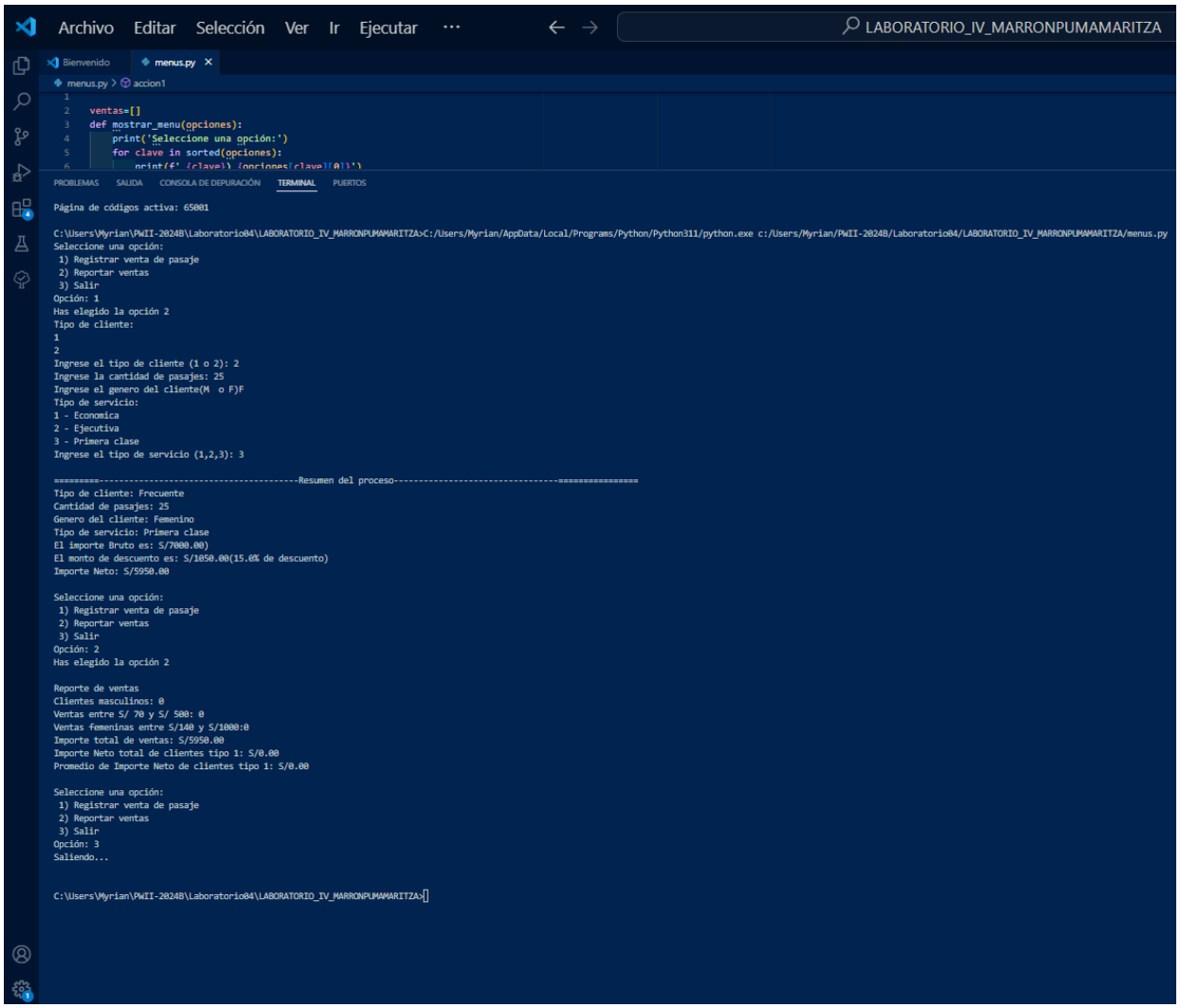
INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	*PROGRAMACIÓN WEB 2*				
TÍTULO DE LA PRÁCTICA:	*CASO: EMPRESA DE TRANSPORTE AÉREO*				
NÚMERO DE PRÁCTICA:	*4*	AÑO LECTIVO:	*2024*	NRO. SEMESTRE:	*II*
FECHA DE PRESENTACIÓN	*12/10/2024*	HORA DE PRESENTACIÓN	*11/30/00*		
INTEGRANTE (s) *Marron Puma, Maritza Claudia*				NOTA (0-20)	
DOCENTE(s): *Lino José Pinto Oppe*					



ejecuciones/pruebas del código fuente explicadas gradualmente.

ESULTADOS Y PRUEBAS

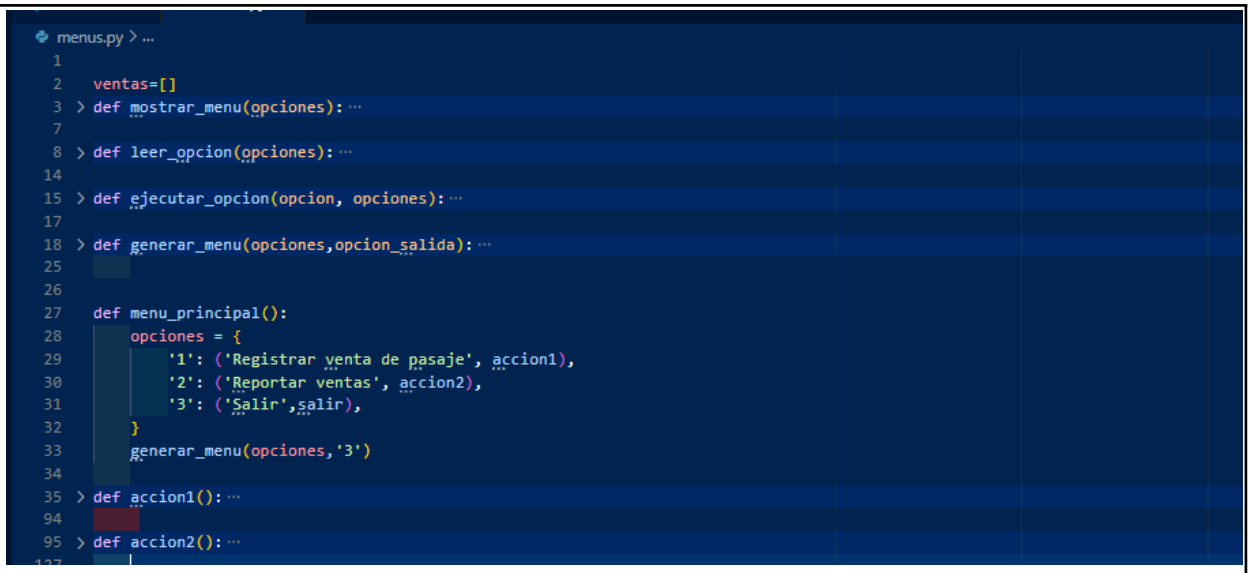
Este informe detalla el proceso de creación y configuración de un sistema de registro y reporte de ventas de pasajes. El objetivo principal de este código es permitir que los usuarios puedan registrar ventas y generar reportes detallados, lo cual facilita la gestión de ventas en una agencia de viajes simulada. A continuación, narro cada parte del código, explicando qué hice, por qué lo hice y la lógica que quise implementar en cada sección.

El usuario podría escoger la primera opción, para el ingreso de datos, o la segunda opción, para ver los resultados de las ventas. Si desea continuar con otro registro de ventas puede volver a escoger la opción 1. Si no desea continuar el registro de ventas escoger la opción 2. Salir.

El sistema se basa en listas para almacenar ventas y diccionarios para describir cada venta de manera detallada. El diseño del código está orientado a ser modular, con cada funcionalidad clave (registro, reporte, menú) implementada como funciones independientes.

Creación del Menú Principal

Primero, decidí crear un menú principal que sirviera como punto de interacción para el usuario. Aquí, quería asegurarme de que el usuario pudiera navegar fácilmente por el sistema y seleccionar entre registrar una venta, ver un reporte o salir del programa. La lógica detrás de esta decisión es darle control total al usuario sobre las acciones a realizar.



Diseñé este menú utilizando un diccionario donde cada opción está mapeada a una función específica. Al usar un diccionario, logré que la selección del usuario estuviera claramente conectada con la acción a realizar, lo que simplifica el flujo del programa. Esto me permite gestionar las opciones de manera más estructurada y flexible.

El diccionario `opciones` asocia las cadenas de texto (opciones del menú) a funciones específicas, lo que permite a los usuarios seleccionar qué acción realizar. Esta implementación facilita la extensión del código, ya que agregar nuevas funcionalidades solo implica añadir entradas al diccionario. Además, ***`generar_menu(opciones, '3')`*** define la opción por defecto como '3' para salir, lo cual gestiona una interacción más fluida.

Registro de Venta de Pasaje

Para el registro de ventas,, creé la función `accion1`. Mi idea aquí era permitir que el usuario ingresara la información necesaria para registrar una venta, como el tipo de cliente, el número de pasajes, el género del cliente, y el tipo de servicio seleccionado. Sabía que necesitaba asegurarme de que todos los datos ingresados fueran válidos antes de proceder, por eso implementé controles y validaciones para cada entrada.

```python
def accion1():
    tipocliente = int(input("Ingrese el tipo de cliente (1 o 2): "))
    cantpasajes = int(input("Ingrese la cantidad de pasajes: "))
    genero = input("Ingrese el genero del cliente(M o F)").upper()
    tiposervice = int(input("Ingrese el tipo de servicio (1, 2, 3): "))

    preciopasaje = [70.00, 140.00, 280.00][tiposervice - 1]
    descuento = [0, 0.05, 0.12, 0.15][min(cantpasajes - 1, 3)]

    importbruto = can  (variable) importbruto: float
    montodescuento = importbruto * descuento
    importeNeto = importbruto - montodescuento

    venta = {'tipocliente': tipocliente, 'genero': genero, 'importeNeto': importeNeto, 'tiposervicio': tiposervice}
    ventas.append(venta)
```

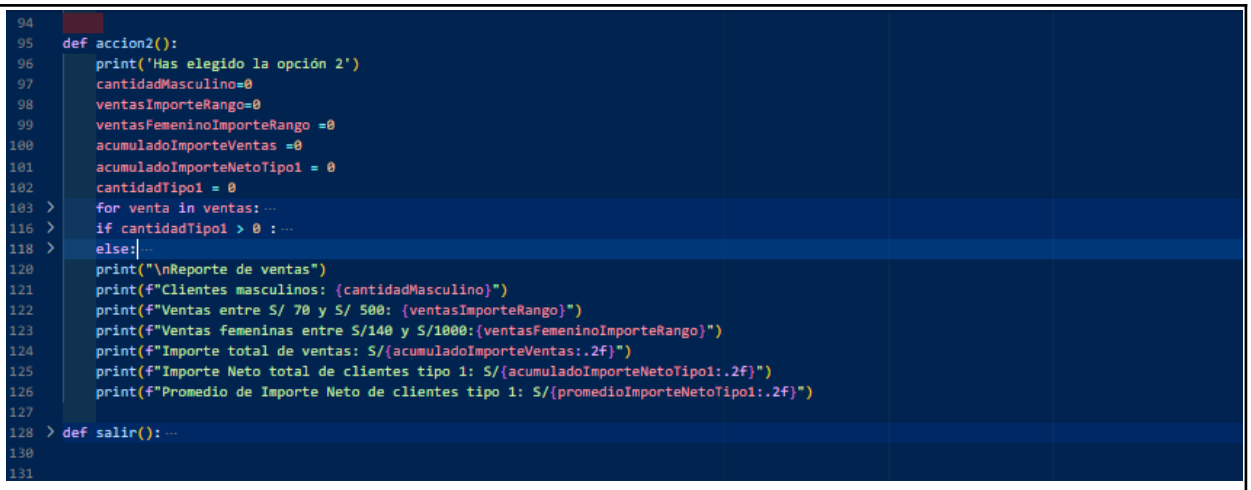
Aquí la lógica es simple pero eficaz: dependiendo de las opciones ingresadas por el usuario, calculo el precio del pasaje y aplico los descuentos correspondientes. Utilicé listas para asociar directamente el tipo de servicio con su precio y el número de pasajes con su descuento. Esta estructura me permitió evitar múltiples estructuras `if-else`, haciendo que el código fuera más compacto y fácil de mantener.

Además, guardo la información de cada venta en una lista para poder procesarla más adelante. Esto era clave para mí, ya que me permite generar reportes acumulados en otra parte del programa.

Para almacenar los datos de una venta, utilizo un diccionario donde cada clave representa una característica relevante de la venta (tipo de cliente, género, servicio). Los precios de los pasajes se seleccionan a través de un índice basado en el valor de `tiposervice`, optimizando así la selección sin necesidad de usar estructuras de control largas. Del mismo modo, los descuentos se calculan utilizando el número de pasajes como índice, con un control de límites mediante **min()** para asegurar que no se excedan los índices de la lista de descuentos. Finalmente, la venta se almacena en una lista global `ventas`, que acumula todas las ventas del sistema.

Reporte de Ventas

Sabía que una de las funcionalidades importantes del sistema era generar reportes sobre las ventas realizadas. Así que en la función accion2 me enfoqué en hacer un análisis simple pero útil, como contar cuántos clientes masculinos se registraron, cuántas ventas cayeron dentro de ciertos rangos de importe, y calcular el importe total de ventas.



```
def accion2():
    print('Has elegido la opción 2')
    cantidadMasculino=0
    ventasImporteRango=0
    ventasFemeninoImporteRango =0
    acumuladoImporteVentas =0
    acumuladoImporteNetoTipo1 = 0
    cantidadTipo1 = 0
    for venta in ventas:
    if cantidadTipo1 > 0 :
    else:
    print("\nReporte de ventas")
    print(f"Clientes masculinos: {cantidadMasculino}")
    print(f"Ventas entre S/ 70 y S/ 500: {ventasImporteRango}")
    print(f"Ventas femeninas entre S/140 y S/1000:{ventasFemeninoImporteRango}")
    print(f"Importe total de ventas: S/{acumuladoImporteVentas:.2f}")
    print(f"Importe Neto total de clientes tipo 1: S/{acumuladoImporteNetoTipo1:.2f}")
    print(f"Promedio de Importe Neto de clientes tipo 1: S/{promedioImporteNetoTipo1:.2f}")

def salir():
```

En esta parte del código, mi objetivo era realizar cálculos rápidos que proporcionaran información útil al usuario. Utilicé la función ***sum()*** con expresiones generadoras para filtrar y contar las ventas que cumplían con ciertos criterios,para hacer el análisis de las ventas sin recorrer varias veces la lista de ventas.

Utilizo expresiones generadoras dentro de la función ***sum()*** para realizar cálculos en tiempo real sobre la lista de ventas, evitando la necesidad de crear listas adicionales en memoria. Por ejemplo, `sum(1 for v in ventas if v['genero'] == 'M')` cuenta cuántos clientes masculinos existen, mientras que otros filtros realizan cálculos específicos de rangos de importes para generar estadísticas más detalladas..

Funciones Auxiliares para el Menú

Finalmente, creé algunas funciones auxiliares para gestionar la interacción con el menú, como mostrar las opciones, leer la selección del usuario y ejecutar la acción seleccionada. Quería asegurarme de que la navegación fuera fácil y que el usuario siempre pudiera ver las opciones disponibles.

Aquí divido las tareas del menú en tres funciones pequeñas y específicas, que manejan la presentación de las opciones, la validación de la selección del usuario y la ejecución de la acción correspondiente. El uso de ***sorted()*** al mostrar el menú garantiza que las opciones siempre aparezcan en orden numérico. ***leer_opcion()*** valida las entradas del usuario y solo devuelve una opción válida. ***ejecutar_opcion()*** utiliza la opción seleccionada para llamar a la función asociada, almacenada en el diccionario **opciones**.

```python
ventas=[]
def mostrar_menu(opciones):
    print('Seleccione una opción:')
    for clave in sorted(opciones):
        print(f' {clave}) {opciones[clave][0]}')

def leer_opcion(opciones):
    opcion = input('Opción: ')
    while opcion not in opciones:
        print('Opción incorrecta, vuelva a intentarlo.')
        opcion = input('Opción: ')
    return opcion

def ejecutar_opcion(opcion, opciones):
    opciones[opcion][1]()
```

La razón por la cual diseñé estas funciones de esta manera fue para separar la lógica del menú del resto del código,para lograr modularidad en el programa. Además, usar funciones separadas para leer y ejecutar opciones hace que sea más fácil agregar o modificar funciones en el futuro sin alterar la estructura básica del programa.

Conclusiones

Al final, conseguí un sistema que funciona bien para registrar y gestionar ventas de forma sencilla y eficiente. Todo quedó bien organizado, con cada función haciendo las tareas específicas, lo que facilita que el usuario lo use sin problemas y que los cálculos sean correctos. En resumen, el sistema quedó fácil de manejar y bastante claro.

EFERENCIAS Y BIBLIOGRAFÍA

*[1] **Zelle, J. M.** (2010). Python Programming: An Introduction to Computer Science (2nd ed.). Franklin, edle & Associates Inc.*

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	x	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	3	x	1	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	x	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	x	1	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	x	2	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	x	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	x	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	x	3	
TOTAL		20		15	

¡OJO! Anexar a su informe

RUBRICA PARA EL CONTENIDO DEL INFORME Y DEMOSTRACIÓN

El alumno debe marcar o dejar en blanco en celdas de la columna Checklist si cumplió con el ítem correspondiente.

Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo ítems.

El alumno debe autocalificarse en la columna Estudiante de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 2: Rúbrica para contenido del Informe y demostración