



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FACPYA

FACULTAD DE CONTADURÍA PÚBLICA Y ADMINISTRACIÓN

EVIDENCIA 1

ESTRUCTURA DE DATOS Y SU PROCESAMIENTO

Licenciado en Tecnologías de la Información

Semestre: 3° **Grupo:** 31

Maestro: Imelda Lizette Ramírez Rodríguez

Integrantes de equipo:

Hernández González Angela Michelle	#2020879
Molina Ramírez Fernando Nicolas	#2015797
Rentería Flores Yaressi Abigail	#2052959
Rodríguez Sánchez Giovanni Missael	#1955224
Valerio Aguirre Giselle Nahomi	#2012254
Rodríguez Delgado Emerson Aldair	#2017900
Ramos Espinosa Luis Daniel	#2059274
Zambrano Alcorta Roxana Dior	#2025417

Ciudad Universitaria, 16 agosto 2024.

INDICE

INTRODUCCION	3
MODELO DE NEGOCIO	4
CUADRO SINOPTICO	6
PRESENTACIÓN SINTETIZADA DEL PROBLEMA DE ESTUDIO	8
DIAGRAMA DE FLUJO	12
VALIDACIÓN PYTHON	13
VALIDACION DE REPOSITORIO EN GIT	17
CONCLUSION GRUPAL.....	22
CONCLUSION INDIVIDUAL	23

INTRODUCCION

En estructura de datos adecuada es crucial para la eficiencia y claridad de los programas. Cuatro de las estructuras de datos más fundamentales en Python son las listas, las tuplas, los diccionarios y los conjuntos. Cada una de estas estructuras ofrece características únicas que las hacen más adecuadas para diferentes tipos de tareas y operaciones.

Una de las funciones son las listas que son colecciones ordenadas y mutables que permiten almacenar elementos de diferentes tipos y ofrecen una amplia gama de métodos para manipulación y acceso a datos. En contraste, las tuplas son similares a las listas, pero inmutables, lo que significa que una vez creada una tupla, sus elementos no pueden ser modificados, haciéndolas útiles para almacenar datos constantes. Los diccionarios son colecciones desordenadas de pares clave-valor, permitiendo un acceso rápido a los valores mediante claves únicas y siendo extremadamente versátiles para asociar datos. Finalmente, los conjuntos son colecciones desordenadas de elementos únicos, ideales para operaciones que implican la verificación de pertenencia y la eliminación de duplicados.

Para entender mejor las diferencias y similitudes entre estas estructuras, hemos preparado un cuadro comparativo que detalla las características principales, las operaciones básicas y las ventajas de cada una, proporcionando una visión clara de cuándo y por qué usar cada estructura en diferentes situaciones.

MODELO DE NEGOCIO

Programa de Frutería

1. Descripción del Producto: El programa es una solución de software diseñada para fruterías y verdulerías, que permite a los clientes seleccionar productos, generar tickets de compra, y realizar un seguimiento de sus compras. Este sistema facilita tanto la operación de la tienda como la experiencia del cliente, optimizando el proceso de compra.

2. Propuesta de Valor: El software simplifica la gestión de las ventas en la frutería al automatizar la generación de tickets y el seguimiento de inventario, lo que ahorra tiempo y reduce errores. Los clientes pueden ver de manera clara sus compras y obtener un resumen detallado de su gasto, lo que mejora su satisfacción.

3. Segmento de Clientes:

- **Fruterías y Verdulerías:** Pequeños negocios que desean automatizar su proceso de ventas.
- **Clientes Finales:** Usuarios de la frutería que buscan una experiencia de compra rápida y eficiente.

4. Canales de Distribución:

- **Distribución Directa:** Venta del software directamente a los propietarios de fruterías.
- **Marketplace en Línea:** Publicación del software en plataformas de venta de aplicaciones para pequeñas empresas.
- **Paquete de Instalación en Sitio:** Ofrecimiento del software como parte de un servicio de instalación y capacitación.

5. Fuentes de Ingreso:

- **Venta Directa del Software:** Ingreso por la venta de licencias del programa a las fruterías.
- **Suscripción Mensual/Anual:** Ingreso recurrente por servicios adicionales como soporte técnico y actualizaciones.
- **Personalización del Software:** Ingreso por servicios de personalización según las necesidades específicas del cliente.

6. Relación con el Cliente:

- **Soporte Técnico:** Línea de soporte para ayudar a los usuarios con la instalación, uso del software y resolución de problemas.

- **Actualizaciones y Mantenimiento:** Provisión de actualizaciones periódicas para mejorar las funcionalidades del software y añadir nuevas características.
- **Capacitación:** Cursos o tutoriales para enseñar a los empleados de la frutería a utilizar el programa de manera efectiva.

7. Recursos Clave:

- **Equipo de Desarrollo:** Programadores que mantengan y mejoren el software.
- **Equipo de Soporte:** Personal dedicado a ofrecer asistencia técnica.
- **Base de Datos de Productos:** Actualización constante de la base de datos con precios y disponibilidad de productos.

8. Actividades Clave:

- **Desarrollo Continuo:** Mejoras y adiciones a las funcionalidades del software.
- **Marketing y Ventas:** Estrategias de promoción del software en fruterías.
- **Soporte y Capacitación:** Ofrecimiento de servicios de asistencia y formación para usuarios finales.

9. Socios Clave:

- **Proveedores de Hardware:** Para la provisión de equipos que soporten el software (p.ej., impresoras de tickets).
- **Distribuidores Locales:** Aliados para la promoción y distribución del software en diferentes localidades.

10. Estructura de Costos:

- **Desarrollo de Software:** Costo asociado a la creación y mantenimiento del programa.
- **Soporte y Mantenimiento:** Costos relacionados con la provisión de asistencia técnica y actualización del software.
- **Marketing y Ventas:** Costos asociados a la promoción y venta del software.
- **Infraestructura:** Costos de servidores, almacenamiento y otros recursos tecnológicos necesarios para el funcionamiento del software.

CUADRO SINOPTICO

TIPOS DE ESTRUCTURAS DE DATOS EN PYTHON

	DEFINICIÓN	SINTAXIS	CÓDIGO
Listas	Permiten guardar cualquier tpo de dato enteros, cadenas, objetos e incluso mas listas También se pueden crear a partir de otros iterables.	Las listas en Python se definen utilizando corchetes []. Pueden contener cualquier tipo de dato (números, cadenas, otros objetos, incluso otras listas), y los elementos se separan por comas.	<pre> 1 # Lista de alumnos 2 alumnos = ['Ana', 'Luis', 'Carlos', 'Maria', 'Elena'] 3 4 # Mostrar la lista de alumnos 5 print('Lista de alumnos:', alumnos) 6 7 # Añadir un alumno 8 alumnos.append('Jorge') 9 print('Lista actualizada:', alumnos) 10 11 # Eliminar un alumno 12 alumnos.remove('Luis') 13 print('Lista final:', alumnos) </pre>
Tuplas	Se utilizan para almacenar colecciones de elementos, y su principal característica es que son inmutables, lo que significa que una vez que se crea una tupla, no se puede modificar .	Las tuplas se definen utilizando paréntesis (). Como las listas, pueden contener diferentes tipos de elementos, pero son inmutables, lo que significa que no se pueden modificar después de su creación.	<pre> 2 alumnos = ('Ana', 'Luis') + ('Carlos', 'Maria', 'Elena') 3 print('Tupla concatenada:', alumnos) 4 5 # Acceder a un elemento 6 print('Tercer alumno:', alumnos[2]) 7 8 # Desempaquetado 9 a1, a2, a3, a4, a5 = alumnos 10 print('Alumnos desempaquetados:', a1, a2, a3, a4, a5) 11 12 # Verificar si un elemento está en la tupla 13 print('Maria está en la tupla' if 'Maria' in alumnos else 'Maria no está en la tupla.') 14 15 # Modificar la tupla convirtiéndola en lista 16 alumnos = tuple(list(alumnos) + ['Jorge']) 17 alumnos = tuple(alumno for alumno in alumnos if alumno != 'Luis') 18 print('Tupla modificada:', alumnos) 19 20 # Tuplas anidadas 21 anidados = ('Ana', 'Luis'), ('Carlos', 'Maria'), ('Elena', 'Jorge') 22 print('Primer alumno de la clase A:', anidados[0][0]) </pre>

TIPOS DE ESTRUCTURAS DE DATOS EN PYTHON

	DEFINICIÓN	SINTAXIS	CÓDIGO
Diccionarios	Permiten almacenar colecciones de elementos en forma de pares de clave-valor. Esta forma de almacenamiento facilita el acceso y la manipulación de los datos, ya que cada valor se asocia a una clave única.	Los diccionarios en Python se definen usando llaves {}. Están compuestos por pares clave-valor, donde cada clave es única dentro del diccionario, y se utiliza para acceder a su valor asociado.	<pre> 1 # Crear un diccionario 2 frutas = {"manzana": 3, "banana": 5, "naranja": 2} 3 4 # Acceder a un valor 5 print(frutas["manzana"]) 6 7 # Agregar un nuevo elemento 8 frutas["pera"] = 4 9 10 # Modificar un valor existente 11 frutas["banana"] = 6 12 13 # Eliminar un elemento 14 del frutas["naranja"] 15 16 # Recorrer el diccionario 17 for fruta, cantidad in frutas.items(): 18 print(fruta, cantidad) </pre>
Conjuntos	Son colecciones desordenadas de elementos únicos. A diferencia de las listas y los diccionarios, los conjuntos no permiten elementos duplicados y no tienen un orden específico.	Los conjuntos en Python se definen usando llaves {} o la función set().	<pre> 1 # Definir dos conjuntos 2 conjunto_A = {1, 2, 3, 4, 5} 3 conjunto_B = {4, 5, 6, 7, 8} 4 5 # Unión de conjuntos 6 union = conjunto_A conjunto_B 7 print("Unión de A y B:", union) 8 9 # Intersección de conjuntos 10 interseccion = conjunto_A & conjunto_B 11 print("Intersección de A y B:", interseccion) 12 13 # Diferencia de conjuntos 14 diferencia = conjunto_A - conjunto_B 15 print("Diferencia de A y B:", diferencia) 16 17 # Diferencia simétrica de conjuntos 18 diferencia_simetrica = conjunto_A ^ conjunto_B 19 print("Diferencia simétrica de A y B:", diferencia_simetrica) 20 21 # Verificar si un elemento está en el conjunto 22 elemento = 3 23 pertenece = elemento in conjunto_A 24 print("El elemento", elemento, "está en el conjunto A?", pertenece) </pre>

PRESENTACIÓN SINTETIZADA DEL PROBLEMA DE ESTUDIO



The screenshot shows a Jupyter Notebook with the following code:

```
# Funcion para mostrar el menú de la fruteria
def mostrar_menu():
    print("\n--- Menú de Frutería ---")
    print("1. Frutas")
    print("2. Verduras")
    print("3. Mostrar todos los tickets")
    print("4. Salir")
    print("Elige una opción: ", end='')

# Funcion para mostrar el menú de frutas
def mostrar_frutas():
    print("\n--- Frutas Disponibles ---")
    for i, (fruta, precio) in enumerate(frutas.items(), start=1):
        print(f"{i}. {fruta} - ${precio:.2f}")

# Funcion para mostrar el menú de verduras
def mostrar_verduras():
    print("\n--- Verduras Disponibles ---")
    for i, (verdura, precio) in enumerate(verduras.items(), start=1):
        print(f"{i}. {verdura} - ${precio:.2f}")

# Funcion para agregar los productos
def agregar_producto(cesta, producto, precio):
    if producto in cesta:
```

The screenshot shows the continuation of the Jupyter Notebook with the following code:

```
    cesta[producto] += 1
    else:
        cesta[producto] = 1
    return precio

# Funcion para mostrar el total a pagar en el ticket
def mostrar_ticket(cesta, nombre):
    print(f"\n--- Ticket de Compra de {nombre} ---")
    total = 0
    for producto, cantidad in cesta.items():
        precio = frutas.get(producto, verduras.get(producto, 0))
        subtotal = precio * cantidad
        total += subtotal
    print(f"{producto} x{cantidad} - ${subtotal:.2f}")
    print(f"Total a pagar: ${total:.2f}")

# Funcion para mostrar los tickets de los usuarios
def mostrar_todos_tickets(usuarios):
    print("\n--- Tickets de Todos los Usuarios ---")
    for nombre, cesta in usuarios.items():
        mostrar_ticket(cesta, nombre)

def seleccionar_producto(opciones, tipo):
    try:
```

The screenshot shows a Jupyter Notebook in VS Code with the following code:

```
def seleccionar_producto(opciones, tipo):
    try:
        seleccion = int(input(f"Selecciona una {tipo} (número): ")) - 1
        if 0 <= seleccion < len(opciones):
            producto = list(opciones.keys())[seleccion]
            precio = opciones[producto]
            return producto, precio
        else:
            print("Selección fuera de rango.")
            return None, None
    except ValueError:
        print("Entrada no válida.")
        return None, None

# Diccionario de frutas con su precio
frutas = {
    'Manzana': 1.00,
    'Banana': 0.50,
    'Naranja': 0.80,
    'Fresa': 1.20,
    'Mango': 1.50,
    'Durazno': 1.00
}

# Diccionario de verduras con su precio
verduras = {
```

The terminal output shows the execution of the code, with the user inputting '1' to select the first item (Manzana) from the fruits dictionary.

The screenshot shows a Jupyter Notebook in VS Code with the following code:

```
# Diccionario de verduras con su precio
verduras = {
    'Zanahoria': 0.60,
    'Lechuga': 0.70,
    'Pepino': 0.50,
    'Tomate': 0.90,
    'Aguacate': 1.00
}

usuarios = {}
usuario_actual = None

# Registro del usuario
while True:
    if usuario_actual is None:
        usuario_actual = input("\nIntroduce tu nombre: ").strip()
        if usuario_actual not in usuarios:
            usuarios[usuario_actual] = {}
            print(f"Bienvenido, {usuario_actual}!")

    mostrar_menu()
    opcion = input().strip()

    if opcion == '1':
        mostrar_frutas()
        fruta, precio = seleccionar_producto(frutas, 'fruta')
```

The terminal output shows the execution of the code, with the user inputting their name and selecting the first item (Manzana) from the fruits dictionary.

The screenshot shows a Jupyter Notebook titled 'Evidencia1.ipynb' in the VS Code editor. The code is written in Python 3.12.2 and implements a menu-driven program for a fruit store. The code includes functions for displaying fruits, vegetables, and tickets, and for adding products to a list. The main loop allows the user to select an option from a menu and perform actions like viewing products or adding items. The code is as follows:

```
if opcion == '1':
    mostrar_frutas()
    fruta, precio = seleccionar_producto(frutas, 'fruta')
    if fruta:
        agregar_producto(usuarios[usuario_actual], fruta, precio)
elif opcion == '2':
    mostrar_verduras()
    verdura, precio = seleccionar_producto(verduras, 'verdura')
    if verdura:
        agregar_producto(usuarios[usuario_actual], verdura, precio)
elif opcion == '3':
    mostrar_todos_tickets(usuarios)
elif opcion == '4':
    mostrar_todos_tickets(usuarios)
    print("¡Gracias por tu visita!")
    break
else:
    print("Opción no válida. Por favor, selecciona de nuevo.")

# Preguntar si el usuario quiere continuar o salir

if opcion != '4':
    continuar = input("\n¿Deseas hacer más compras? (s/n): ").strip().lower()
    if continuar != 's':
        mostrar_todos_tickets(usuarios)
        print("¡Gracias por tu visita!")
```

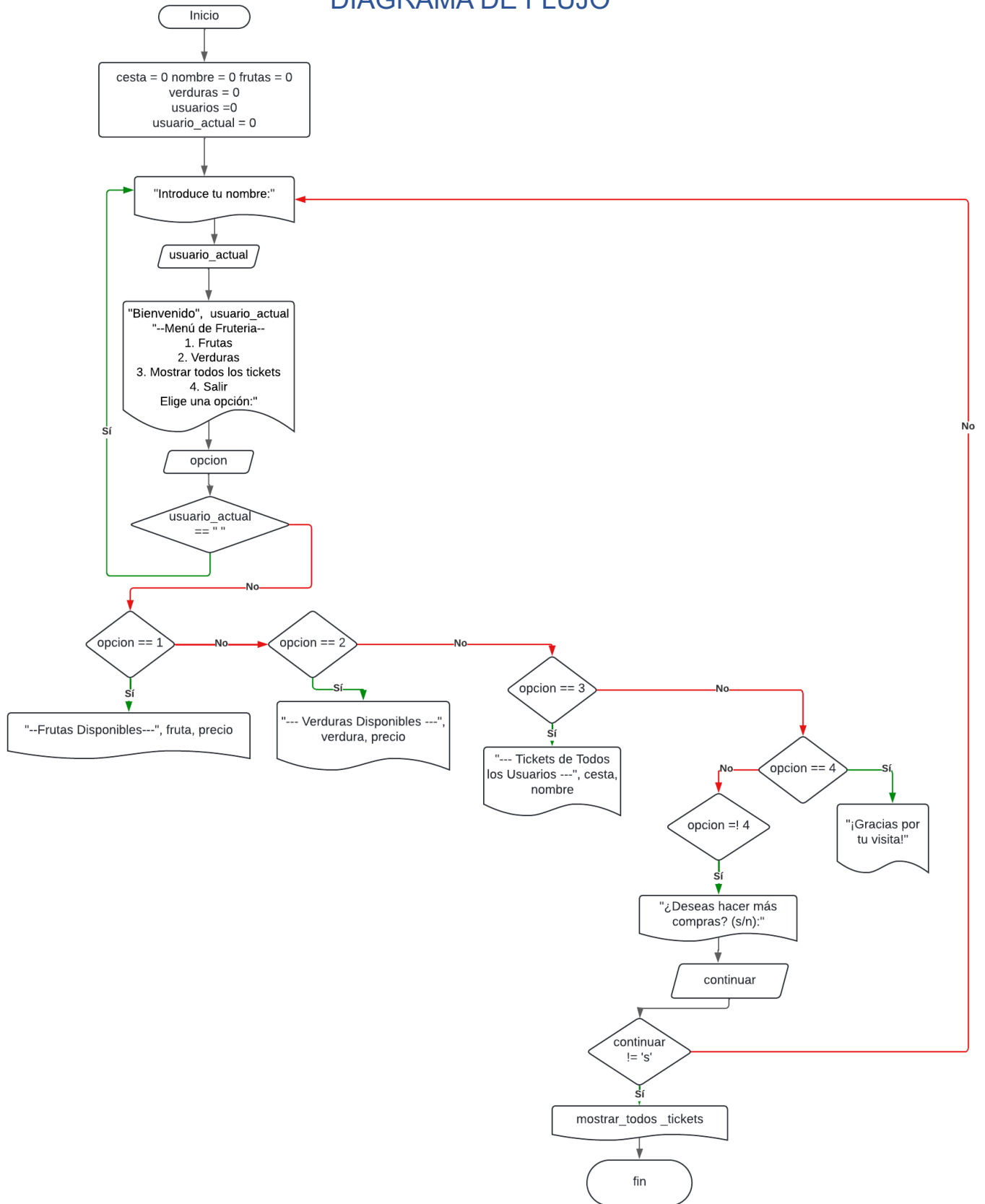
The screenshot shows the terminal output of the Jupyter Notebook. The program has executed the code from the previous screenshot, and the output is as follows:

```
[4]
Python

... Bienvenido, Deyanira Eliza!

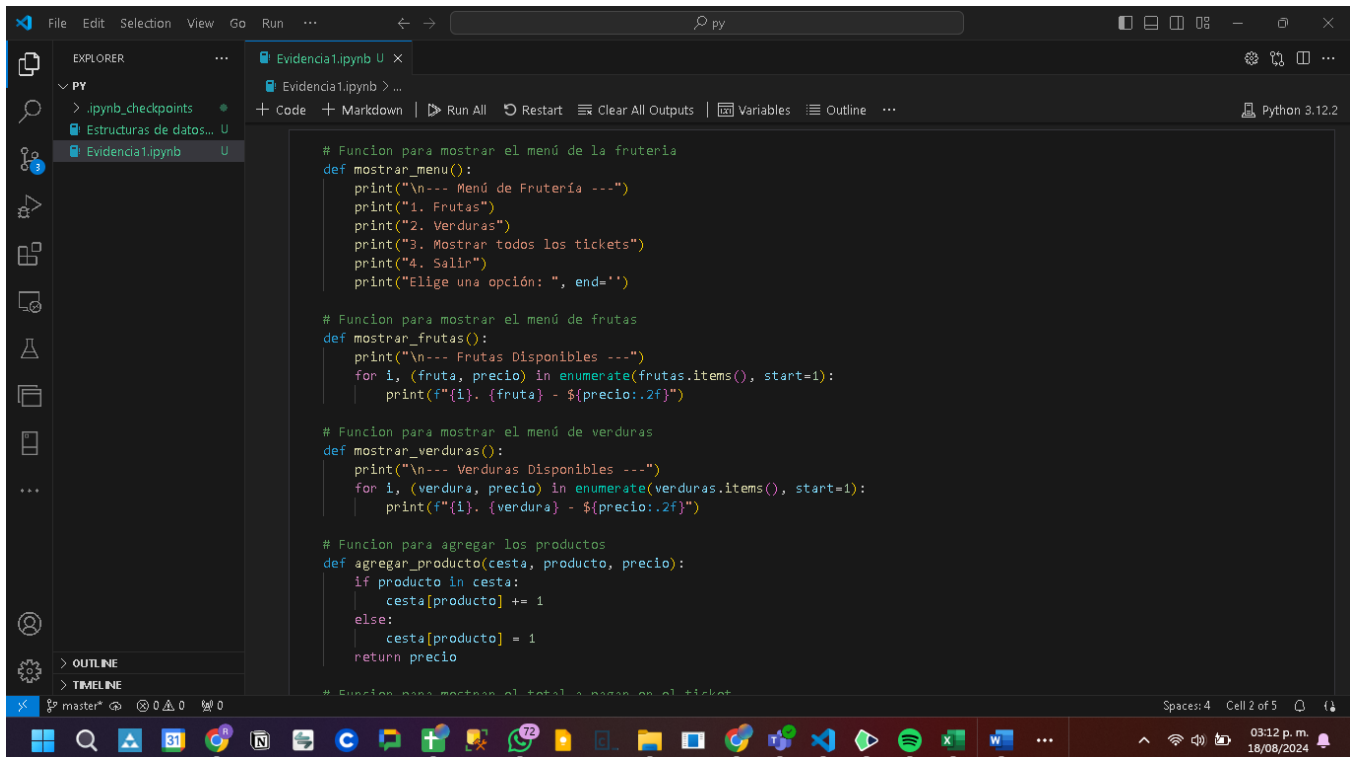
--- Menú de Frutería ---
1. Frutas
2. Verduras
3. Mostrar todos los tickets
4. Salir
Elige una opción:
--- Frutas Disponibles ---
1. Manzana - $1.00
2. Banana - $0.50
3. Naranja - $0.80
4. Fresa - $1.20
5. Mango - $1.50
6. Durazno - $1.00
```

DIAGRAMA DE FLUJO



VALIDACIÓN PYTHON

CAPTURAS CÓDIGO PYTHON USANDO JUPYTER NOTEBOOK EN VISUAL STUDIO



This screenshot shows the Visual Studio Code interface with a Jupyter Notebook open. The Explorer panel on the left shows the file structure with 'Evidencia1.ipynb' selected. The main editor displays the following Python code:

```
# Funcion para mostrar el menú de la fruteria
def mostrar_menu():
    print("\n--- Menú de Frutería ---")
    print("1. Frutas")
    print("2. Verduras")
    print("3. Mostrar todos los tickets")
    print("4. Salir")
    print("Elige una opción: ", end='')

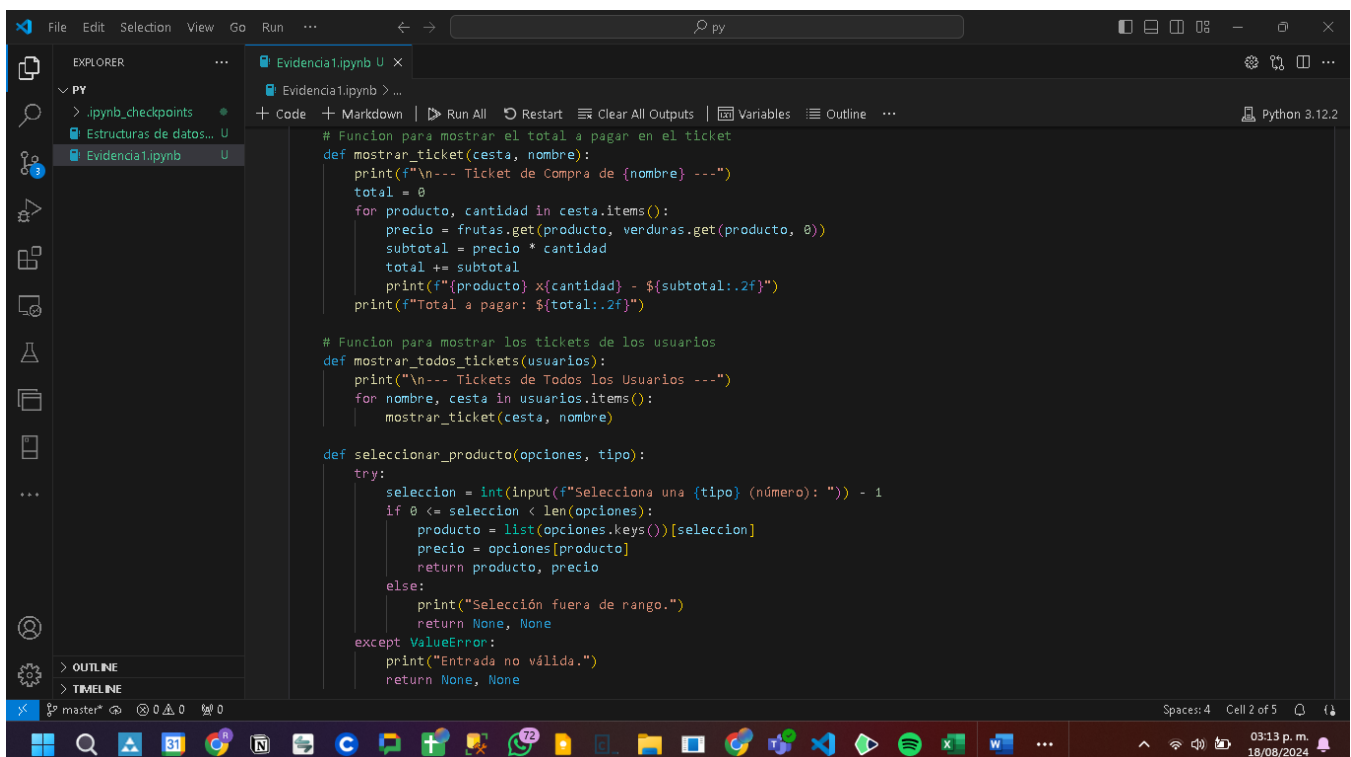
# Funcion para mostrar el menú de frutas
def mostrar_frutas():
    print("\n--- Frutas Disponibles ---")
    for i, (fruta, precio) in enumerate(frutas.items(), start=1):
        print(f"{i}. {fruta} - ${precio:.2f}")

# Funcion para mostrar el menú de verduras
def mostrar_verduras():
    print("\n--- Verduras Disponibles ---")
    for i, (verdura, precio) in enumerate(verduras.items(), start=1):
        print(f"{i}. {verdura} - ${precio:.2f}")

# Funcion para agregar los productos
def agregar_producto(cesta, producto, precio):
    if producto in cesta:
        cesta[producto] += 1
    else:
        cesta[producto] = 1
    return precio

# Funcion para mostrar el total a pagar en el ticket
```

The status bar at the bottom indicates 'Python 3.12.2' and 'Cell 2 of 5'.



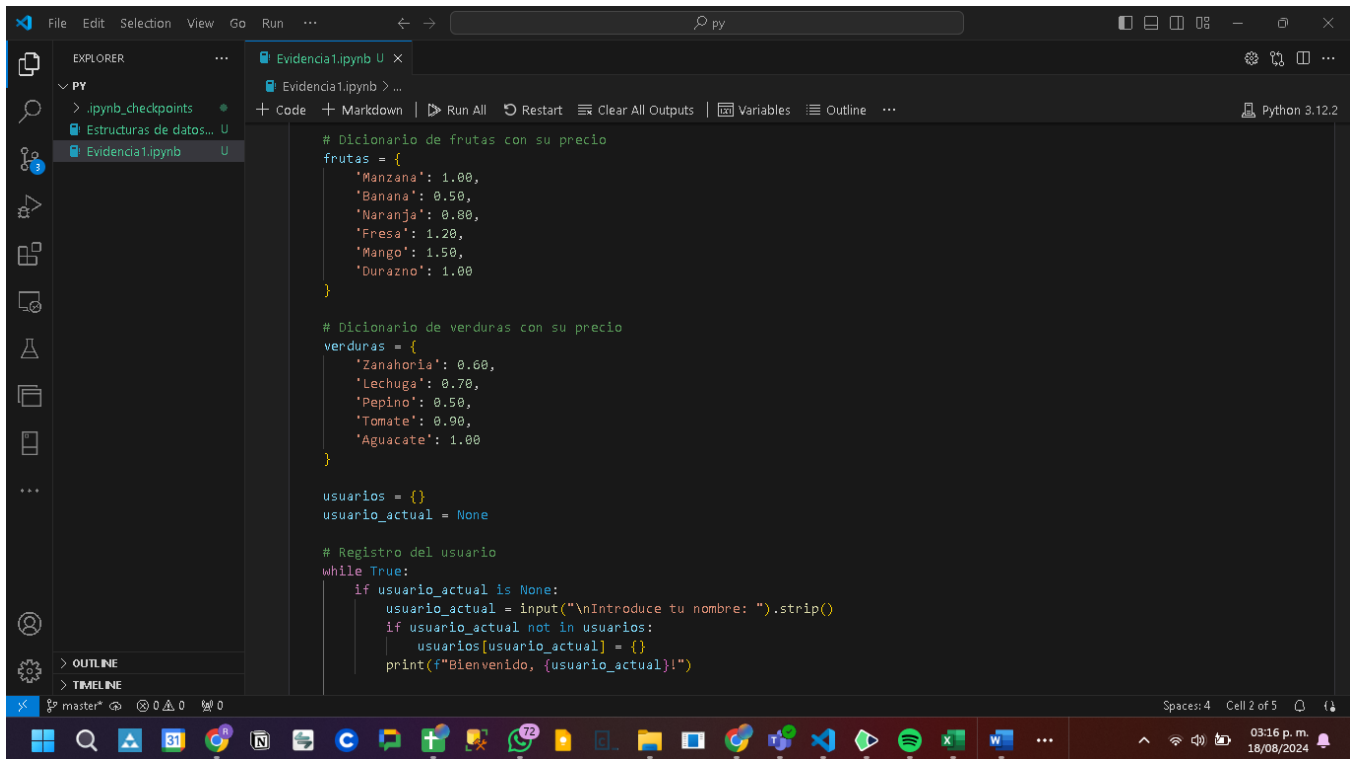
This screenshot shows the Visual Studio Code interface with a Jupyter Notebook open, continuing the code from the previous image. The main editor displays the following Python code:

```
# Funcion para mostrar el total a pagar en el ticket
def mostrar_ticket(cesta, nombre):
    print(f"\n--- Ticket de Compra de {nombre} ---")
    total = 0
    for producto, cantidad in cesta.items():
        precio = frutas.get(producto, 0) + verduras.get(producto, 0)
        subtotal = precio * cantidad
        total += subtotal
    print(f"{producto} x{cantidad} = ${subtotal:.2f}")
    print(f"Total a pagar: ${total:.2f}")

# Funcion para mostrar los tickets de los usuarios
def mostrar_todos_tickets(usuarios):
    print("\n--- Tickets de Todos los Usuarios ---")
    for nombre, cesta in usuarios.items():
        mostrar_ticket(cesta, nombre)

def seleccionar_producto(opciones, tipo):
    try:
        seleccion = int(input(f"Selecciona una {tipo} (número): ")) - 1
        if 0 <= seleccion < len(opciones):
            producto = list(opciones.keys())[seleccion]
            precio = opciones[producto]
            return producto, precio
        else:
            print("Selección fuera de rango.")
            return None, None
    except ValueError:
        print("Entrada no válida.")
        return None, None
```

The status bar at the bottom indicates 'Python 3.12.2' and 'Cell 2 of 5'.

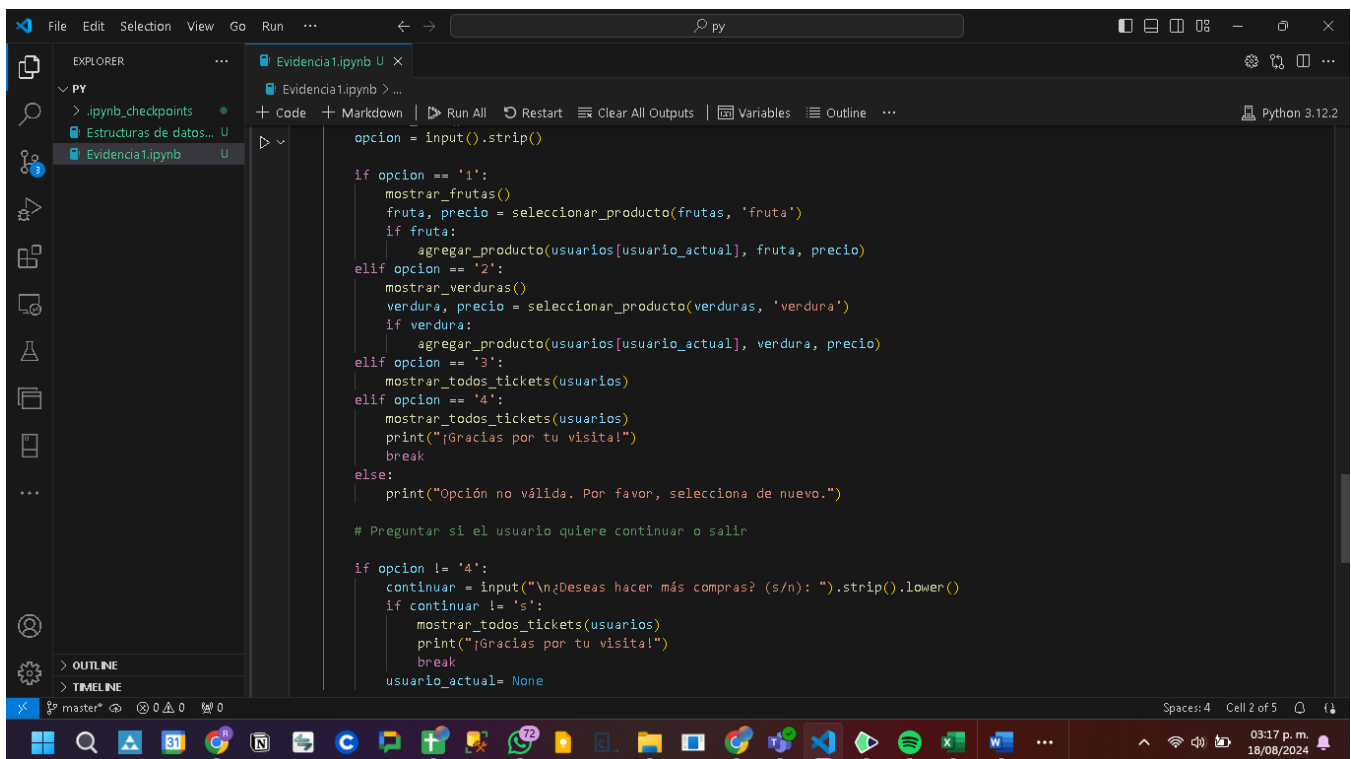


```
# Dicionario de frutas con su precio
frutas = {
    'Manzana': 1.00,
    'Banana': 0.50,
    'Naranja': 0.80,
    'Fresa': 1.20,
    'Mango': 1.50,
    'Durazno': 1.00
}

# Dicionario de verduras con su precio
verduras = {
    'Zanahoria': 0.60,
    'Lechuga': 0.70,
    'Pepino': 0.50,
    'Tomate': 0.90,
    'Aguacate': 1.00
}

usuarios = {}
usuario_actual = None

# Registro del usuario
while True:
    if usuario_actual is None:
        usuario_actual = input("\nIntroduce tu nombre: ").strip()
        if usuario_actual not in usuarios:
            usuarios[usuario_actual] = {}
            print(f"Bienvenido, {usuario_actual}!")
```



```
opcion = input().strip()

if opcion == '1':
    mostrar_frutas()
    fruta, precio = seleccionar_producto(frutas, 'fruta')
    if fruta:
        agregar_producto(usuarios[usuario_actual], fruta, precio)
elif opcion == '2':
    mostrar_verduras()
    verdura, precio = seleccionar_producto(verduras, 'verdura')
    if verdura:
        agregar_producto(usuarios[usuario_actual], verdura, precio)
elif opcion == '3':
    mostrar_todos_tickets(usuarios)
elif opcion == '4':
    mostrar_todos_tickets(usuarios)
    print("\n¡Gracias por tu visita!")
    break
else:
    print("Opción no válida. Por favor, selecciona de nuevo.")

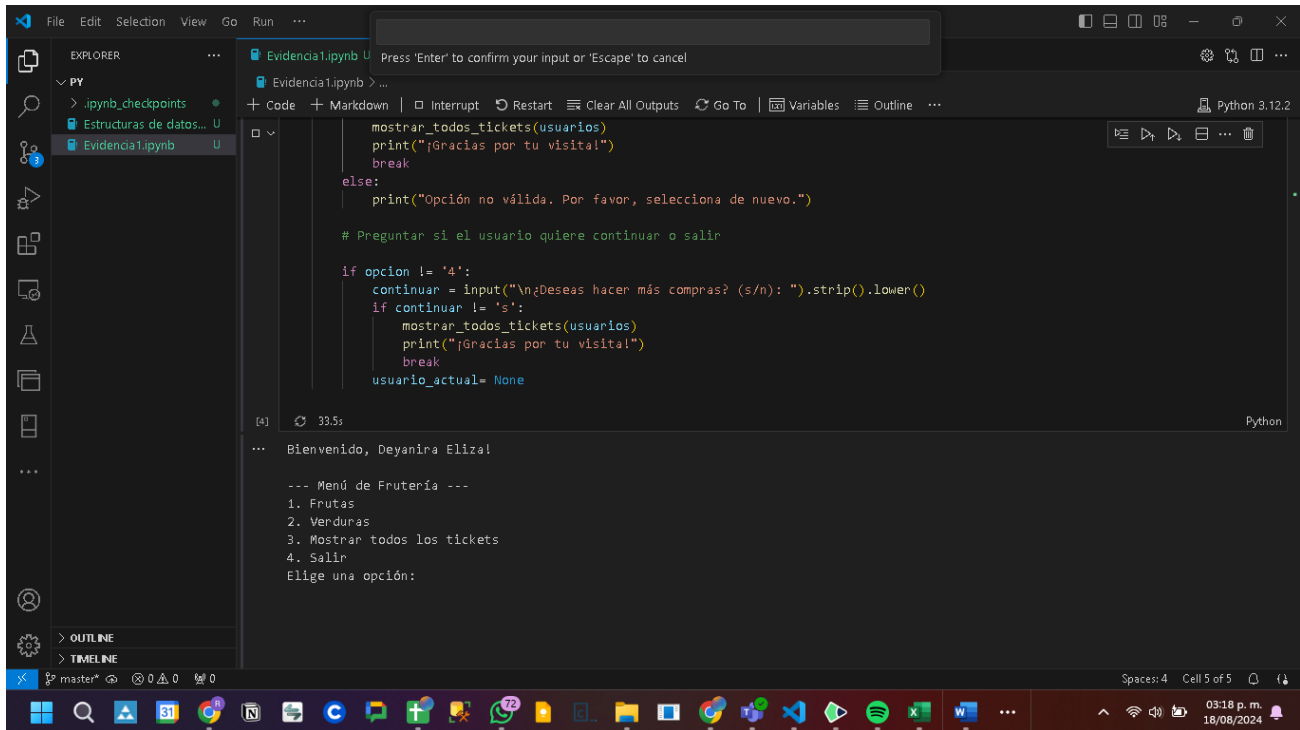
# Preguntar si el usuario quiere continuar o salir

if opcion != '4':
    continuar = input("\n¿Deseas hacer más compras? (s/n): ").strip().lower()
    if continuar != 's':
        mostrar_todos_tickets(usuarios)
        print("\n¡Gracias por tu visita!")
        break
    usuario_actual = None
```

VALIDACIÓN PYTHON

Ejecución del programa

Se ingresa el nombre en el input del programa



```
File Edit Selection View Go Run ...
Evidencia1.ipynb U Press 'Enter' to confirm your input or 'Escape' to cancel
Evidencia1.ipynb > ...
+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ...
Python 3.12.2

mostrar_todos_tickets(usuarios)
print("¡Gracias por tu visita!")
break
else:
    print("Opción no válida. Por favor, selecciona de nuevo.")

# Preguntar si el usuario quiere continuar o salir

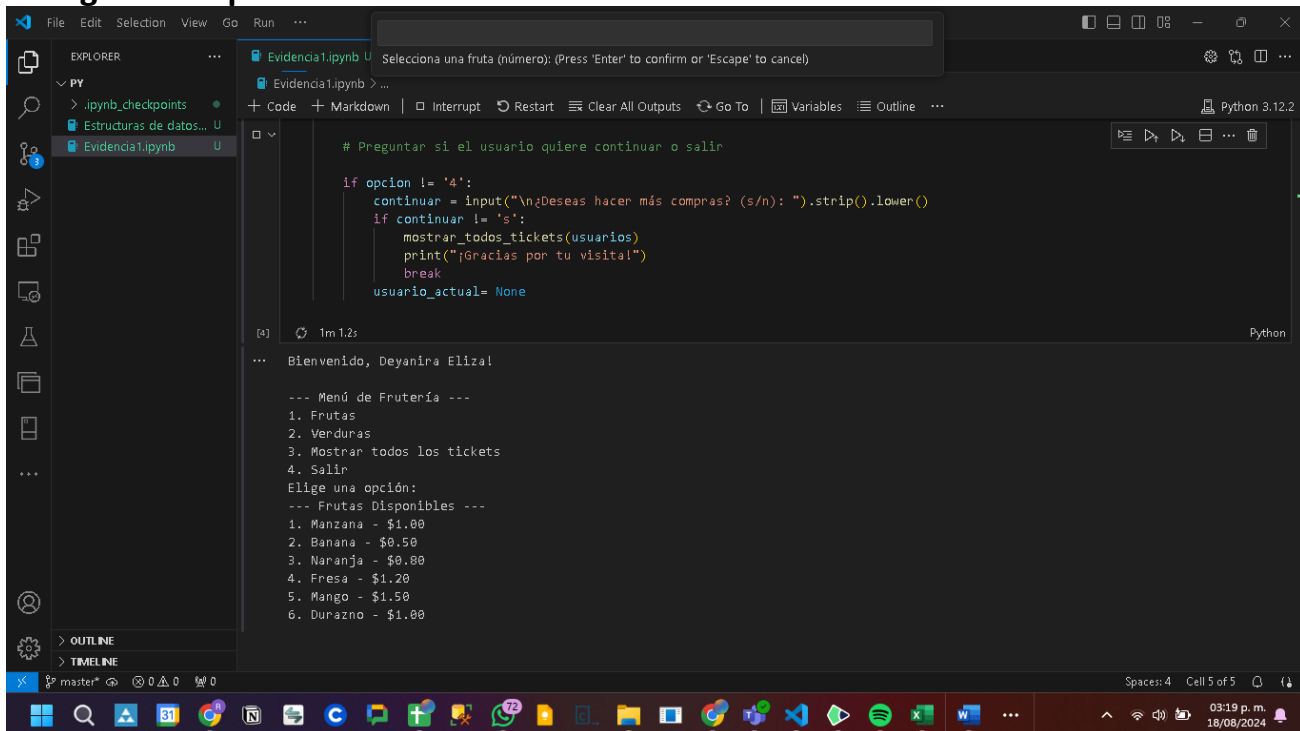
if opcion != '4':
    continuar = input("\n¿Deseas hacer más compras? (s/n): ").strip().lower()
    if continuar != 's':
        mostrar_todos_tickets(usuarios)
        print("¡Gracias por tu visita!")
        break
    usuario_actual= None

[4] 33.5s Python

... Bienvenido, Deyanira Elizal

--- Menú de Frutería ---
1. Frutas
2. Verduras
3. Mostrar todos los tickets
4. Salir
Elige una opción:
```

Se ingresa la opción 1



```
File Edit Selection View Go Run ...
Evidencia1.ipynb U Selecciona una fruta (número): (Press 'Enter' to confirm or 'Escape' to cancel)
Evidencia1.ipynb > ...
+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ...
Python 3.12.2

# Preguntar si el usuario quiere continuar o salir

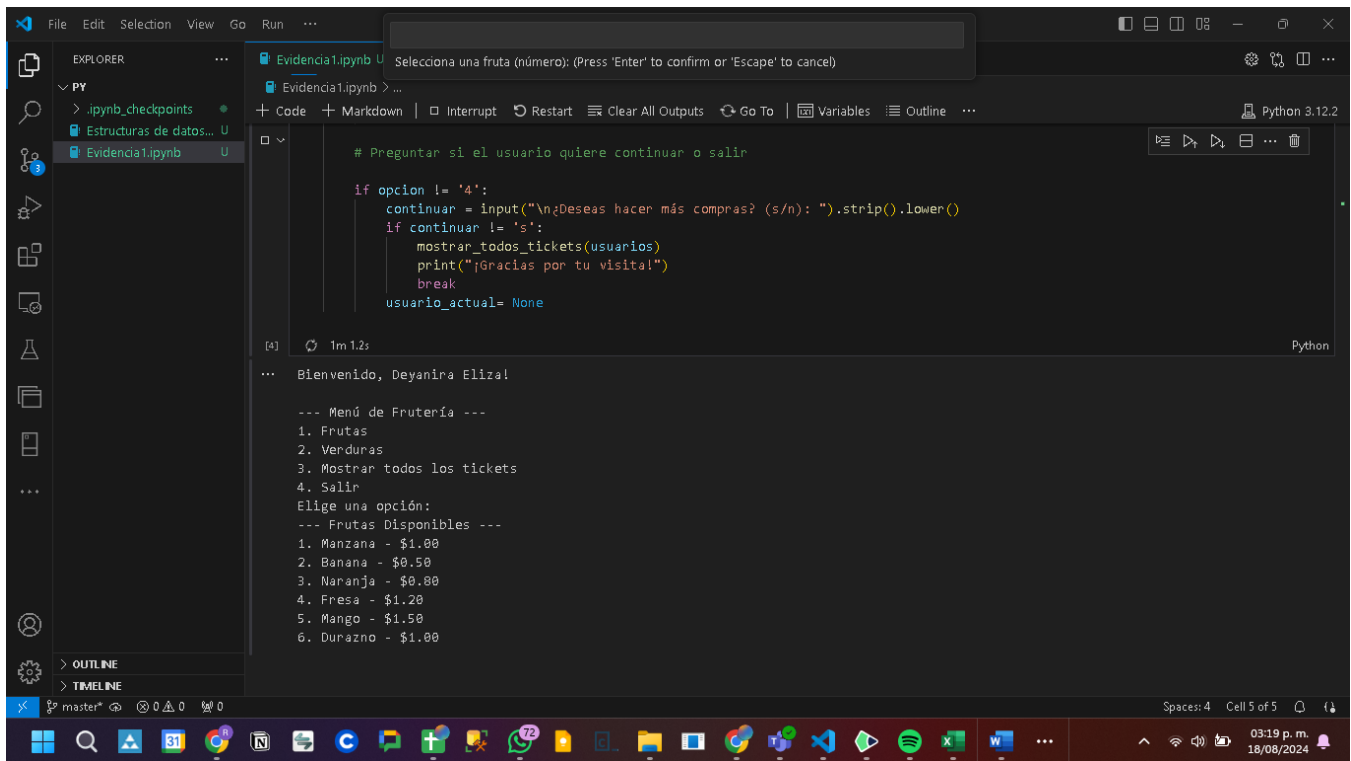
if opcion != '4':
    continuar = input("\n¿Deseas hacer más compras? (s/n): ").strip().lower()
    if continuar != 's':
        mostrar_todos_tickets(usuarios)
        print("¡Gracias por tu visita!")
        break
    usuario_actual= None

[4] 1m12s Python

... Bienvenido, Deyanira Elizal

--- Menú de Frutería ---
1. Frutas
2. Verduras
3. Mostrar todos los tickets
4. Salir
Elige una opción:
--- Frutas Disponibles ---
1. Manzana - $1.00
2. Banana - $0.50
3. Naranja - $0.80
4. Fresa - $1.20
5. Mango - $1.50
6. Durazno - $1.00
```

Se ingresa en el input 1



```
Selecciona una fruta (número): (Press 'Enter' to confirm or 'Escape' to cancel)

# Preguntar si el usuario quiere continuar o salir

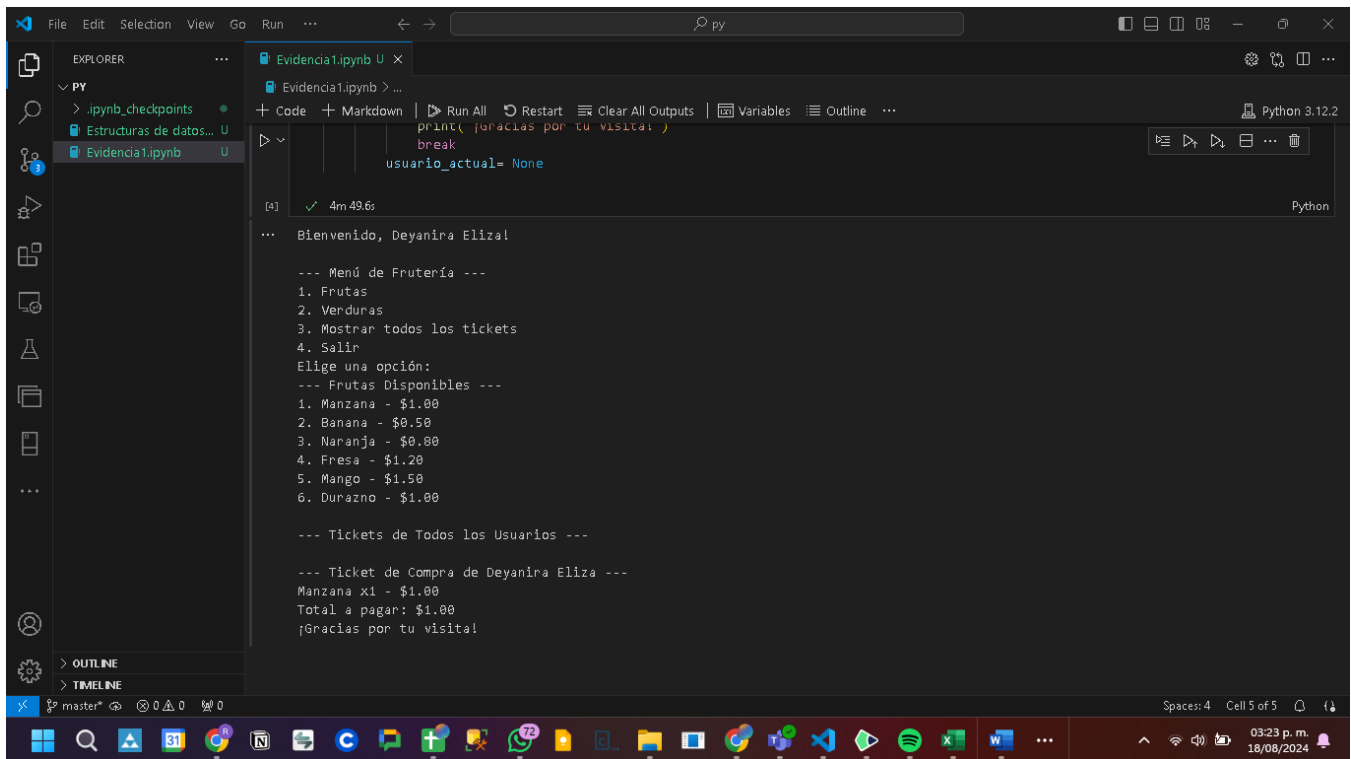
if opcion != '4':
    continuar = input("\n¿Deseas hacer más compras? (s/n): ").strip().lower()
    if continuar != 's':
        mostrar_todos_tickets(usuarios)
        print("¡Gracias por tu visita!")
        break
    usuario_actual= None

[4] 1m 12s Python

...
Bienvenido, Deyanira Eliza!

--- Menú de Frutería ---
1. Frutas
2. Verduras
3. Mostrar todos los tickets
4. Salir
Elige una opción:
--- Frutas Disponibles ---
1. Manzana - $1.00
2. Banana - $0.50
3. Naranja - $0.80
4. Fresa - $1.20
5. Mango - $1.50
6. Durazno - $1.00
```

Se ingresa N porque ya no quiere hacer compras



```
[4] 4m 49.6s Python

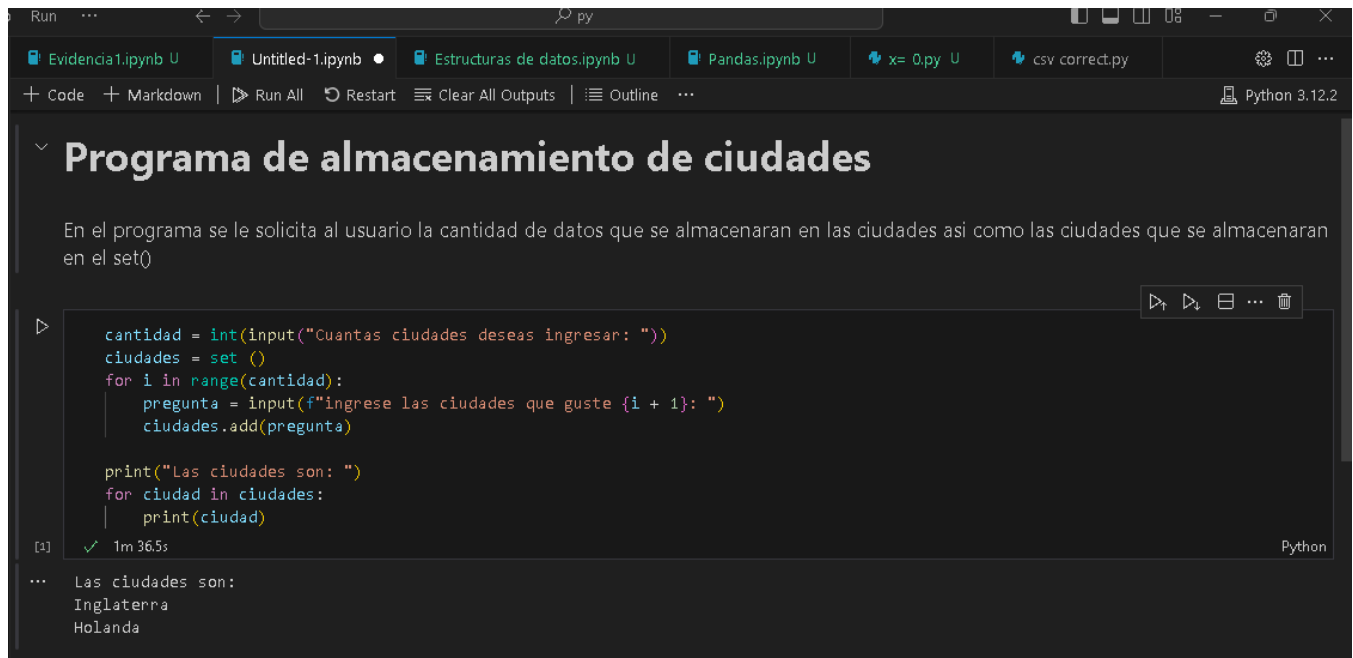
...
Bienvenido, Deyanira Eliza!

--- Menú de Frutería ---
1. Frutas
2. Verduras
3. Mostrar todos los tickets
4. Salir
Elige una opción:
--- Frutas Disponibles ---
1. Manzana - $1.00
2. Banana - $0.50
3. Naranja - $0.80
4. Fresa - $1.20
5. Mango - $1.50
6. Durazno - $1.00

--- Tickets de Todos los Usuarios ---

--- Ticket de Compra de Deyanira Eliza ---
Manzana x1 - $1.00
Total a pagar: $1.00
¡Gracias por tu visita!
```


Programa de almacenamiento de ciudades



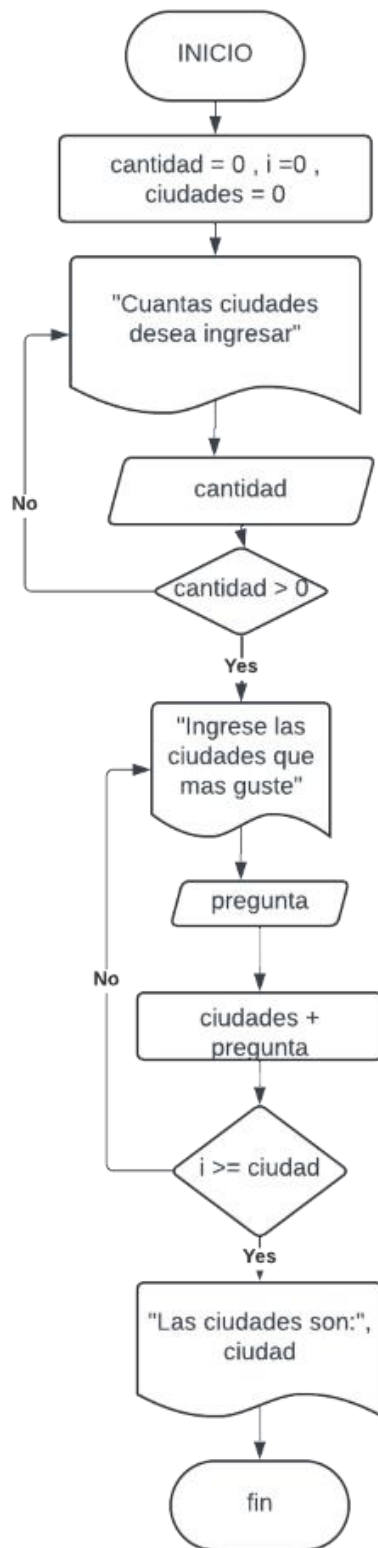
The screenshot shows a Jupyter Notebook window with several tabs. The active tab is 'Untitled-1.ipynb'. The notebook contains a title 'Programa de almacenamiento de ciudades' and a description: 'En el programa se le solicita al usuario la cantidad de datos que se almacenaran en las ciudades asi como las ciudades que se almacenaran en el set()'. Below the text is a code cell with the following Python code:

```
cantidad = int(input("Cuántas ciudades deseas ingresar: "))
ciudades = set ()
for i in range(cantidad):
    pregunta = input(f"ingrese las ciudades que guste {i + 1}: ")
    ciudades.add(pregunta)

print("Las ciudades son: ")
for ciudad in ciudades:
    print(ciudad)
```

The code cell shows a successful execution with a green checkmark and a time of 1m 365s. Below the code cell, the output is displayed: 'Las ciudades son: Inglaterra Holanda'.

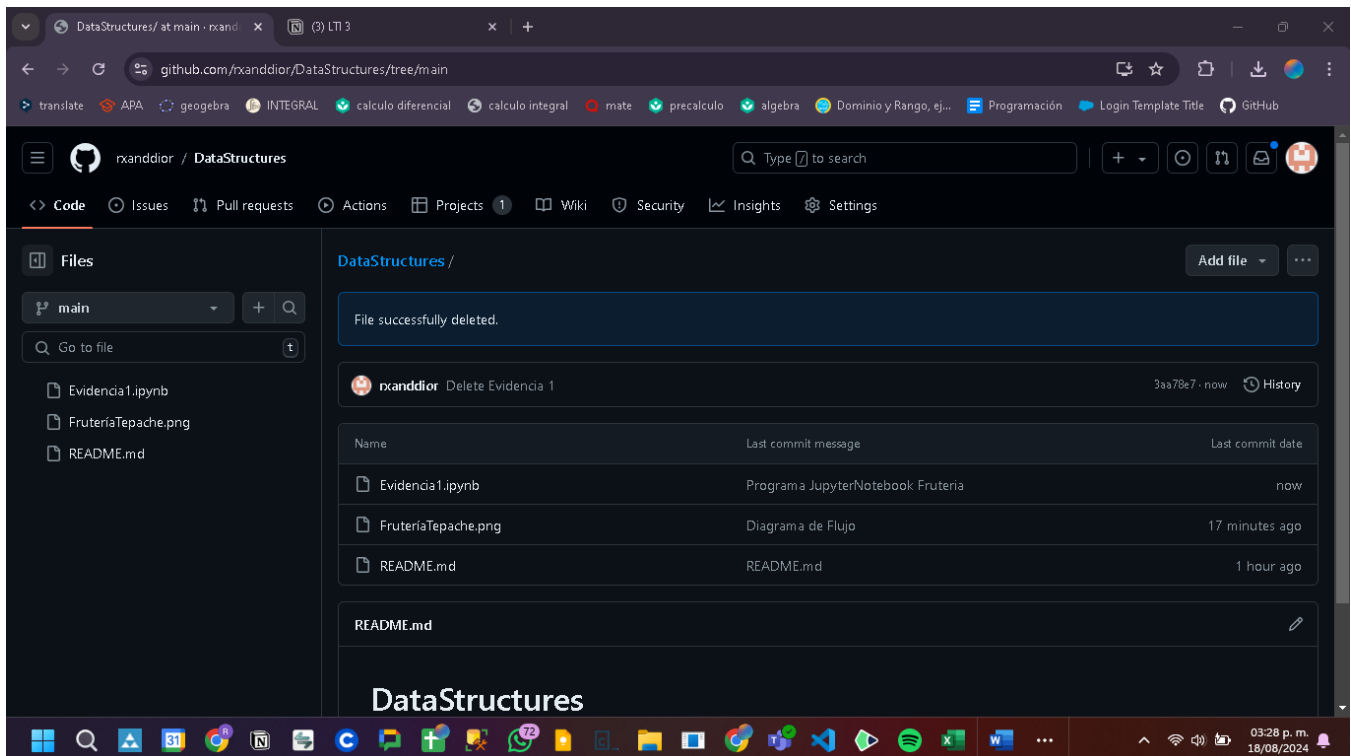
Diagrama de flujo



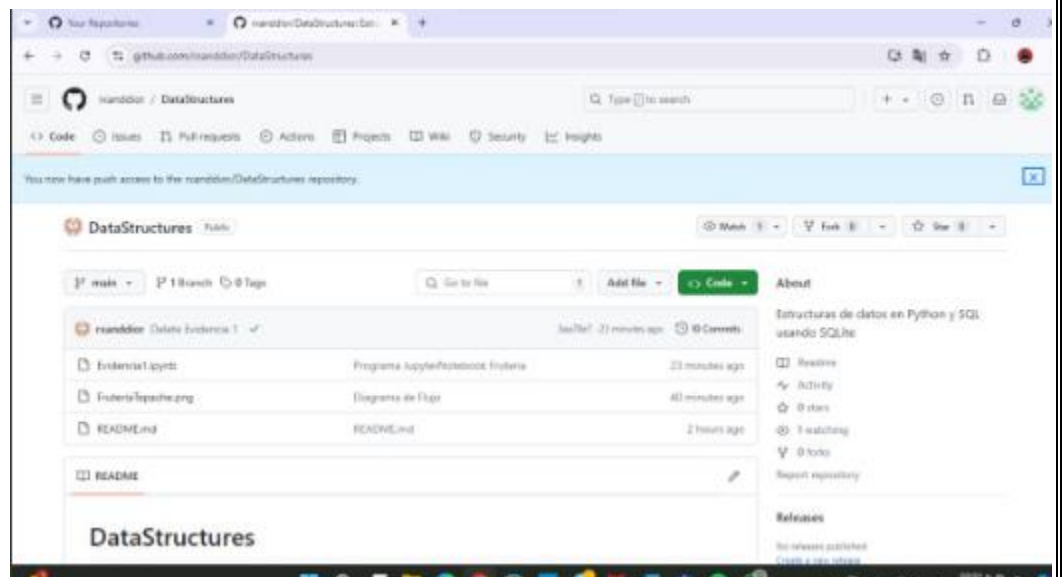
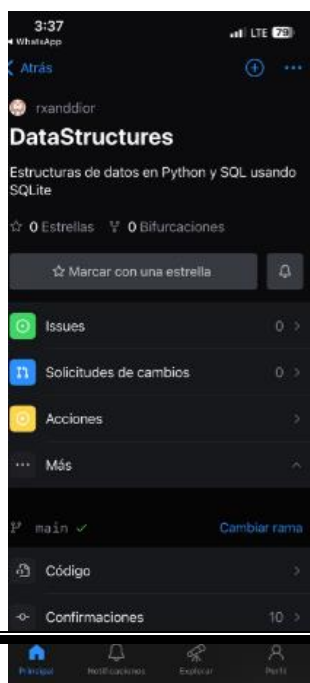
VALIDACION DE REPOSITORIO EN GIT

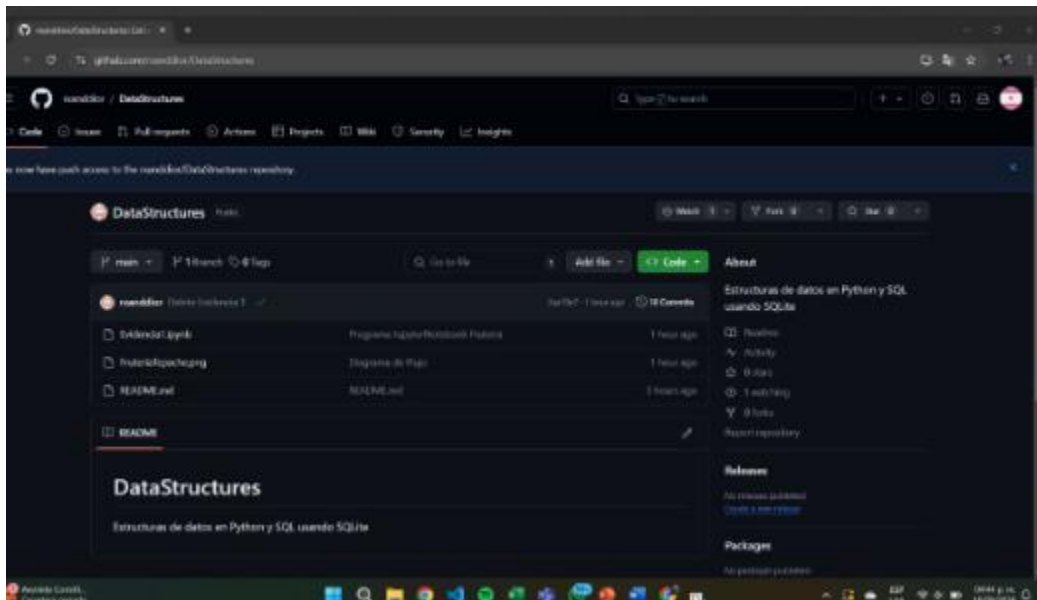
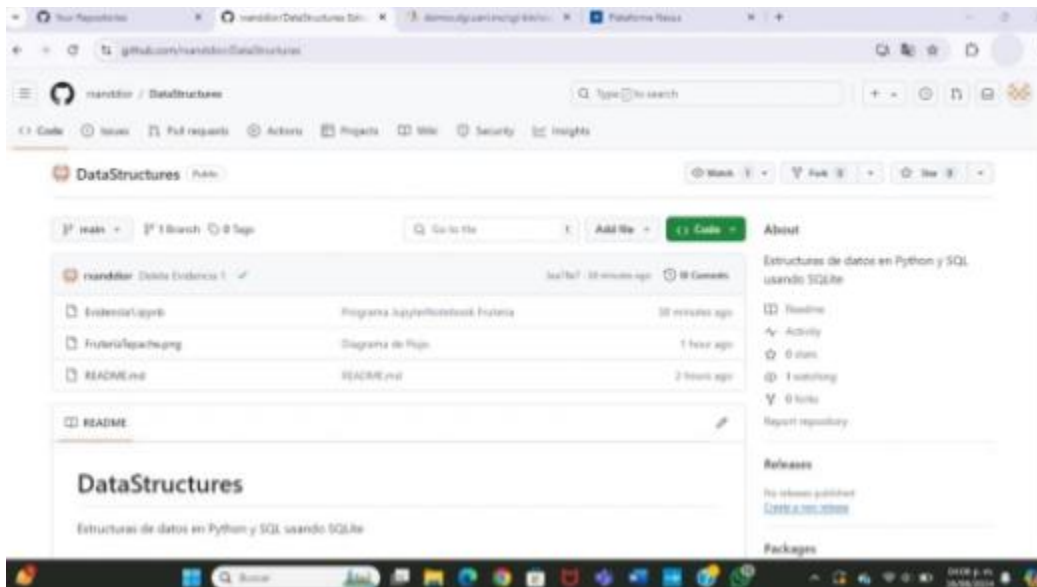
Enlace del Repositorio GitHub

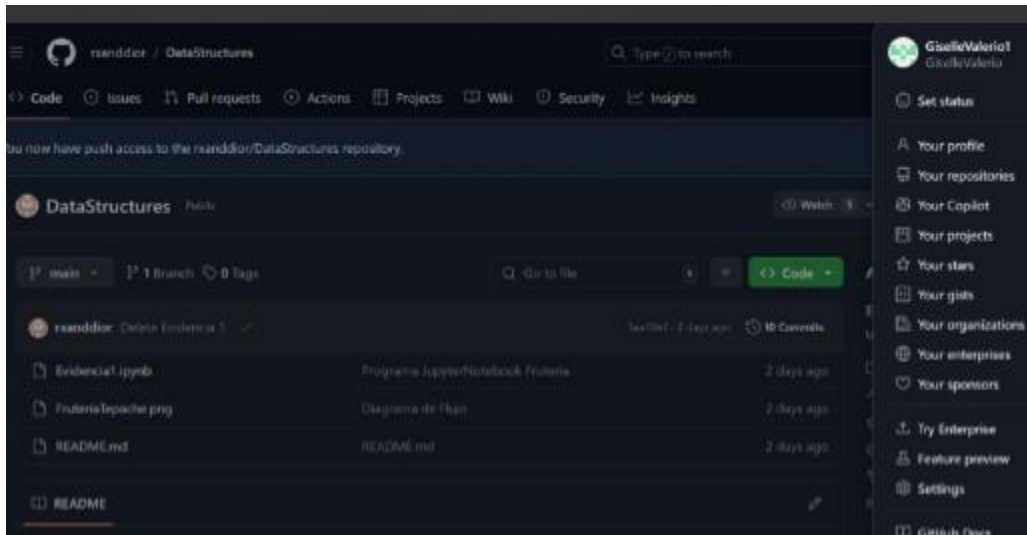
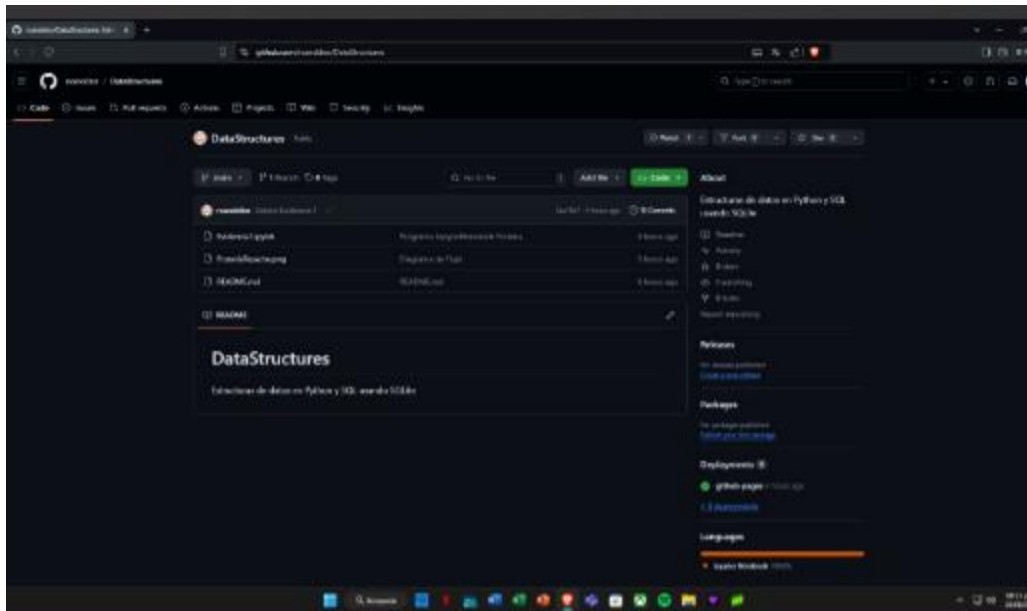
<https://github.com/rxanddior/DataStructures>



Compañeros con acceso a repositorio grupal GitHub







CONCLUSION GRUPAL

En este documento, hemos explorado en profundidad el modelo de negocio de una frutería, abordando tanto los aspectos técnicos como los conceptuales. A través de la comparación entre tuplas, diccionarios y listas, hemos identificado cómo cada una de estas estructuras de datos puede ser aplicada para la gestión eficiente de la información en el contexto de nuestro negocio. Las tuplas ofrecen inmutabilidad, lo que es ideal para datos constantes, mientras que los diccionarios permiten un acceso rápido a los datos mediante claves, y las listas proporcionan flexibilidad en la organización y manipulación de conjuntos de datos.

La presentación del problema de estudio nos permitió contextualizar las necesidades y desafíos que enfrenta la frutería en la gestión de su inventario, ventas y operaciones diarias. Este análisis nos sirvió como base para la construcción del diagrama de flujo, que ilustra claramente los procesos clave y las decisiones críticas en el manejo del negocio, también hemos documentado y centralizado todo el código y los recursos en un repositorio de GitHub, facilitando la colaboración y la evolución del proyecto.

CONCLUSION INDIVIDUAL

Rodríguez Sánchez Giovanni Missael: En este proyecto, me enfoqué en analizar las diferentes estructuras de datos existentes ya sea tuplas, diccionarios, etc. para determinar cuál sería la más efectiva en la gestión de información para nuestra frutería, esta tarea me permitió comprender mejor cómo elegir las herramientas adecuadas puede simplificar y optimizar procesos clave del negocio. Además, participar en la elaboración del diagrama de flujo me ayudó a visualizar de manera clara y precisa las etapas del modelo de negocio. Estoy satisfecho con lo logrado y con mis compañeros de equipo, considero que este proyecto nos ha proporcionado una base sólida para continuar desarrollando soluciones más eficientes y adaptadas a nuestras necesidades

Zambrano Alcorta Roxana Dior: En la construcción de esta evidencia he aprendido a profundidad como se pueden emplear las estructuras de datos en Python y sus distintas funcionalidades para lo que son los proyectos de análisis y almacén de datos, así como para catalogar diversa información utilizando listas, tuplas, diccionarios y conjuntos, así mismo las diferencias entre ellos y como se pueden aplicar los métodos en cada estructura de datos para añadir elementos como en las listas se usa `.append` en cambio en los conjuntos utilizamos `.add` y así en cada uno tienen sus diferencias, los cuales en conjunto nos ayudan a la creación de programas que nos permiten almacenar y procesar los datos que requerimos así como consultar entre ellos.

Rodríguez Delgado Emerson Aldair: En conclusión, sobre esta evidencia aplicamos los diferentes conceptos vistos en clase sobre las estructuras de datos y las implementamos en un pequeño programa que pueda realizar dichos conceptos el que no sabía mucho específicamente es el de las tuplas ya que tiene un numero específico y una secuencia de valores las cuales nos ayuda a crear diferentes programas avanzados que podemos utilizar y poner a prueba.

Rentería Flores Yaressi Abigail: En conclusión, GitHub es una plataforma poderosa para la colaboración y gestión de proyectos de desarrollo de software. A través de GitHub, los programadores pueden compartir y controlar versiones de su código, colaborar en proyectos en equipo y mantener un historial detallado de los cambios realizados en el código fuente.

Por otro lado, en cuanto a los tipos de datos en Python, los diccionarios, tuplas y conjuntos ofrecen estructuras de datos únicas que permiten a los desarrolladores almacenar y manipular información de manera eficiente.

Valerio Aguirre Giselle Nahomi: En conclusión, estas herramientas y conceptos fortalecen la capacidad de un programador para desarrollar, gestionar y colaborar en proyectos de software de manera más efectiva y organizada.

Ramos Espinosa Luis Daniel: En resumen, este trabajo ha mostrado cómo usar Git para gestionar y compartir código Python que emplea estructuras de datos nativas del lenguaje. Utilizando Git, hemos podido llevar un control preciso de las versiones y colaborar de manera más efectiva. Las estructuras de datos en Python, como listas, diccionarios, conjuntos y tuplas, se han aplicado para resolver diferentes problemas, demostrando su versatilidad y utilidad.

Molina Ramírez Fernando Nicolas: El repositorio en Git ha sido clave para documentar y seguir las mejores prácticas en el uso de estas estructuras, sirviendo como una herramienta valiosa para quienes buscan mejorar en Python.

Hernández González Angela Michelle: En conclusión, combinar Git con desarrollo en Python no solo facilita la gestión del proyecto, sino que también ayuda a aprender y aplicar conceptos importantes en programación, creando soluciones más sólidas y eficientes.