*Cómputo de Alto Rendimiento*

*Actividad 7: Ejercicios sobre comunicación entre procesos*

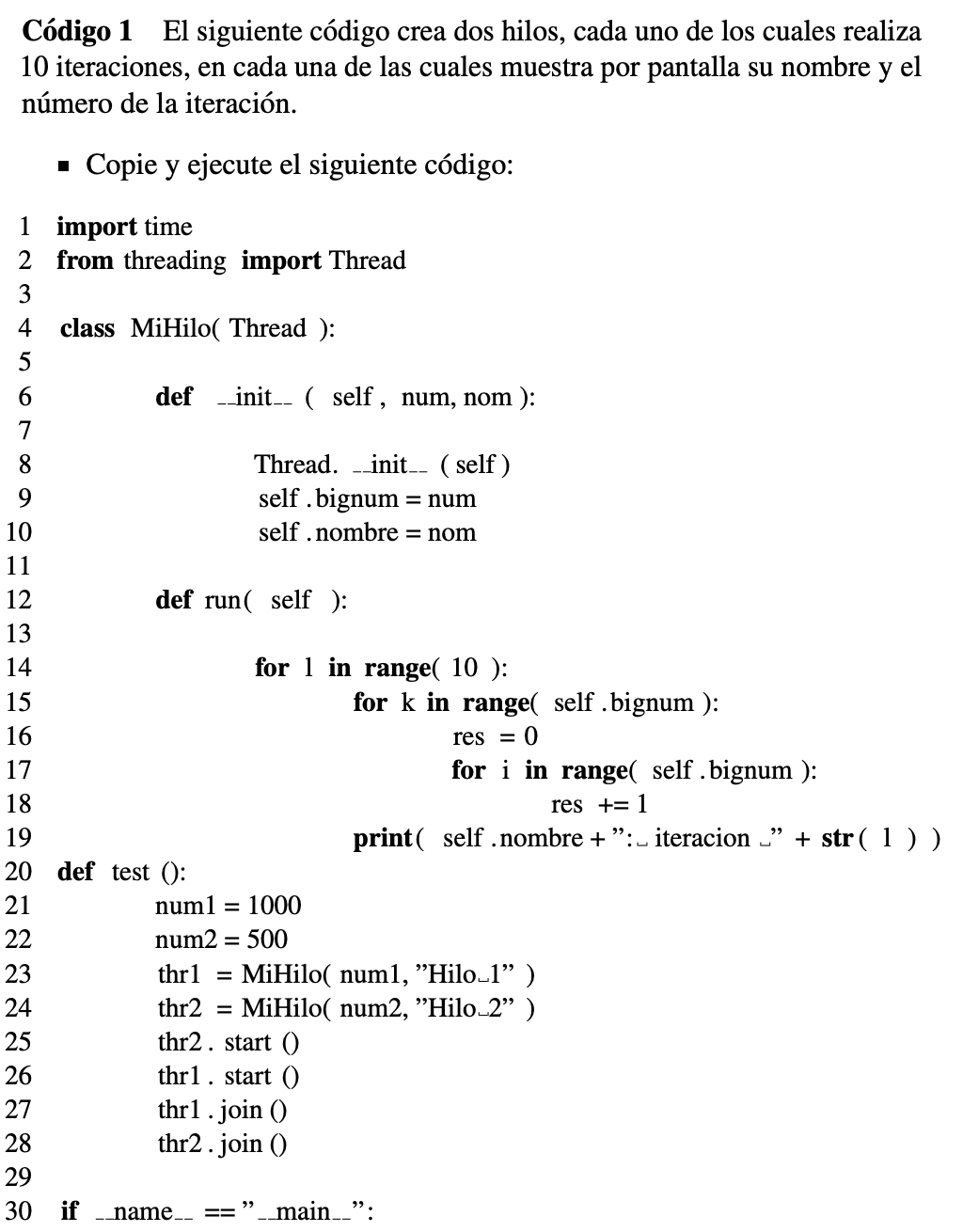
*Nombre: David Aaron Ramirez Olmeda*

*Programa: Maestría en Ciencia de Datos e Información*

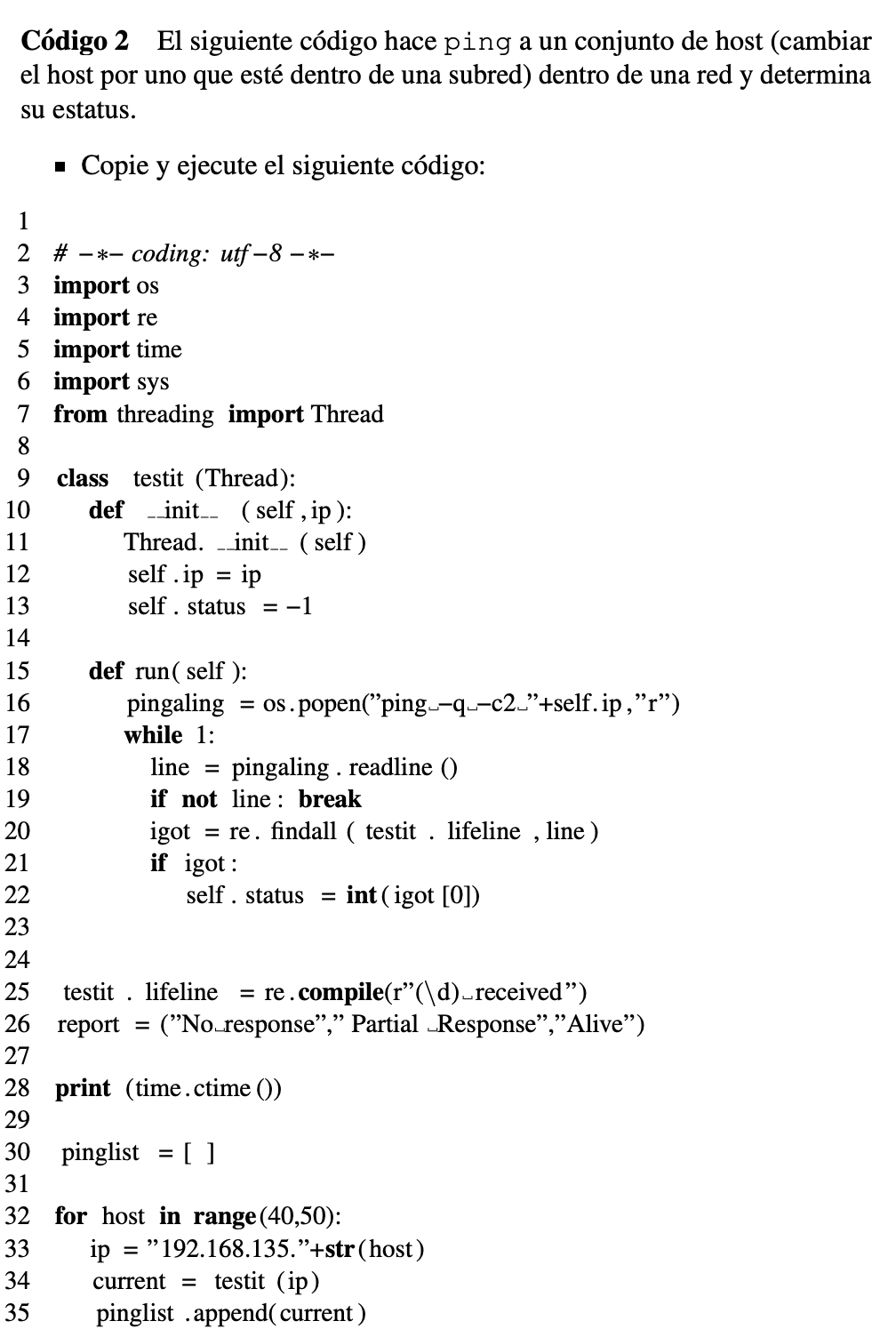
**Introducción**

**1. Reporte**

Analice la ejecucion de cada uno de estos códigos. Elabore un reporte y diga cual es la diferencia entre ejecutar esto con hilos y con ciclos for u otra estructura de control ¿qué utilidad le ve a los hilos?



El Código 1 crea dos hilos, "Hilo 1" y "Hilo 2", que realizan 10 iteraciones cada uno. En cada iteración, muestran su nombre y el número de la iteración. Este enfoque permite realizar cálculos en paralelo. La ventaja de utilizar hilos es que ambas tareas se ejecutan simultáneamente, aprovechando la capacidad de procesamiento multicore. Esto puede acelerar el proceso y es especialmente útil para tareas sensibles al tiempo de respuesta, como las aplicaciones financieras.



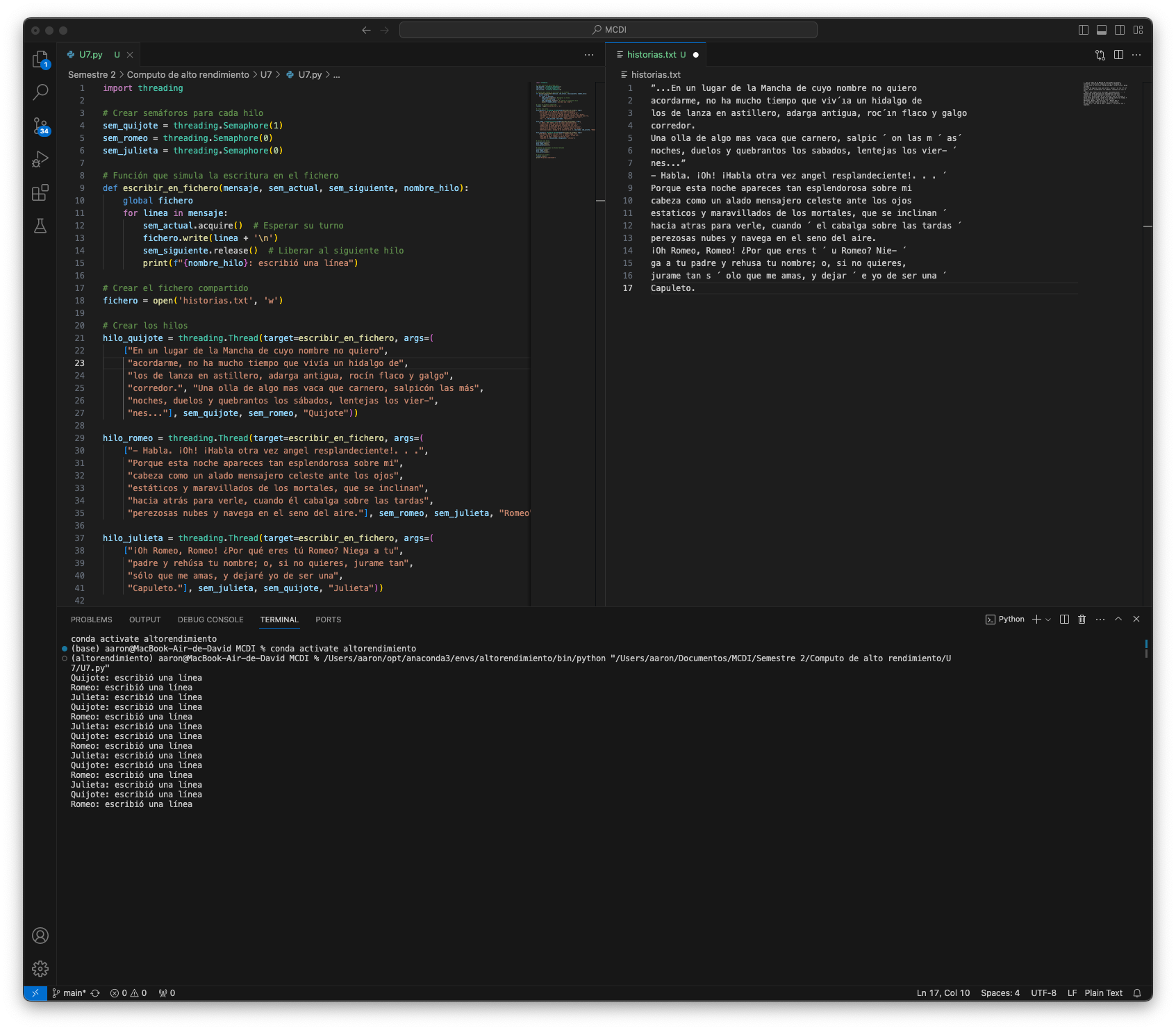
El Código 2 realiza un escaneo de ping en una serie de hosts dentro de una red para determinar su estado. Se utilizan hilos para realizar los escaneos en paralelo, lo que acelera el proceso de escaneo. La ventaja de usar hilos es evidente al escanear múltiples hosts, ya que los resultados se obtienen de manera más eficiente y rápida. Esto es útil en situaciones dispositivos de red para monitorear su disponibilidad o algo similar.

En ambos casos, la ventaja de utilizar hilos radica en la concurrencia, lo que permite realizar múltiples tareas de manera simultánea, acelerando el procesamiento y mejorando el rendimiento.

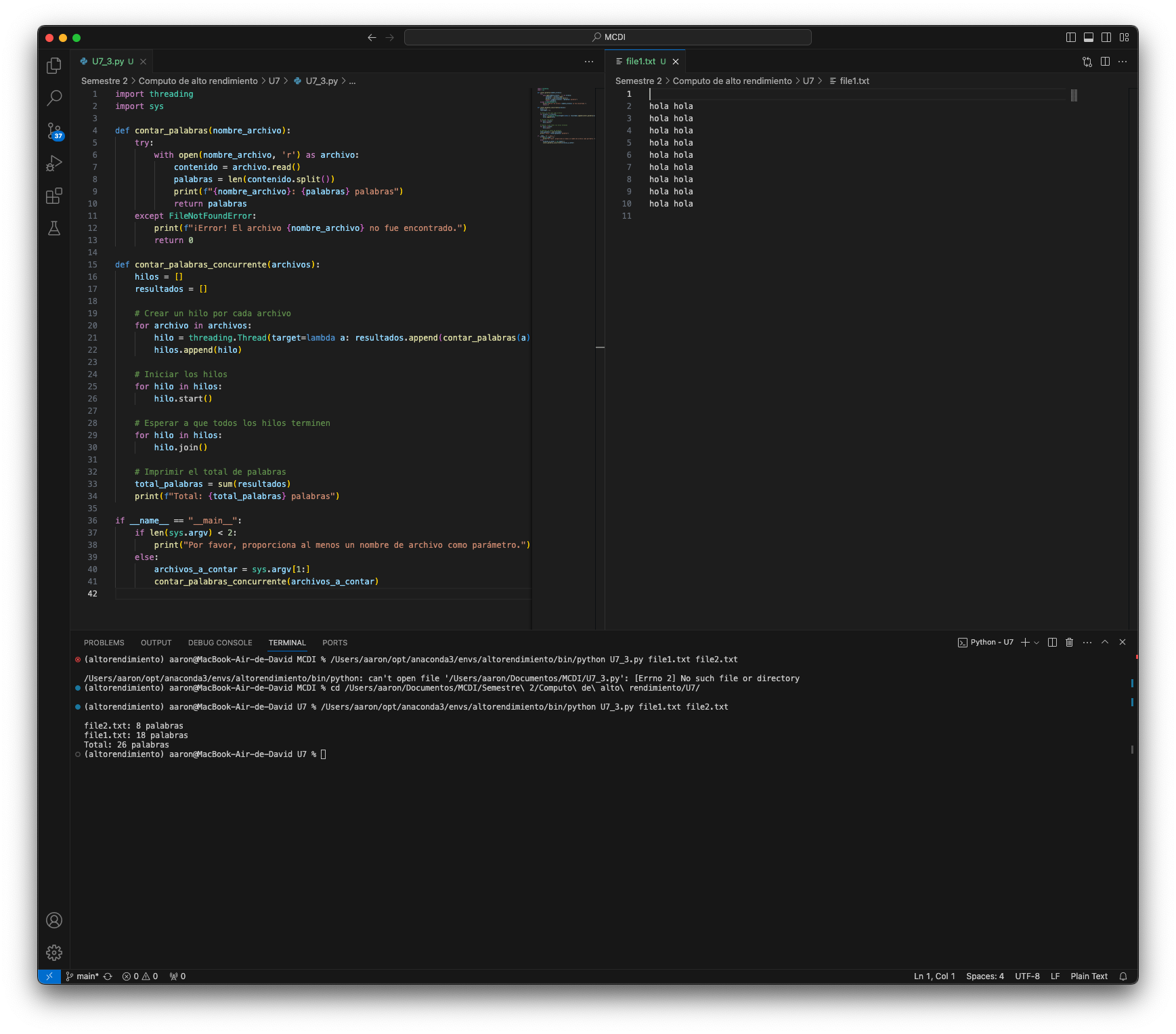
**2. Hilos literarios**

Para lograr la sincronización de los hilos que escriben en un mismo fichero en un orden específico, podemos utilizar semáforos. En este caso, necesitaremos un semáforo para cada hilo, y cada hilo deberá esperar su turno antes de escribir en el fichero compartido.

Ojo, creo que no salió tan bien y tuve que modificar un poco el archivo a mano.



**3. Disenar y codificar un programa que resuelva cuenta palabras**

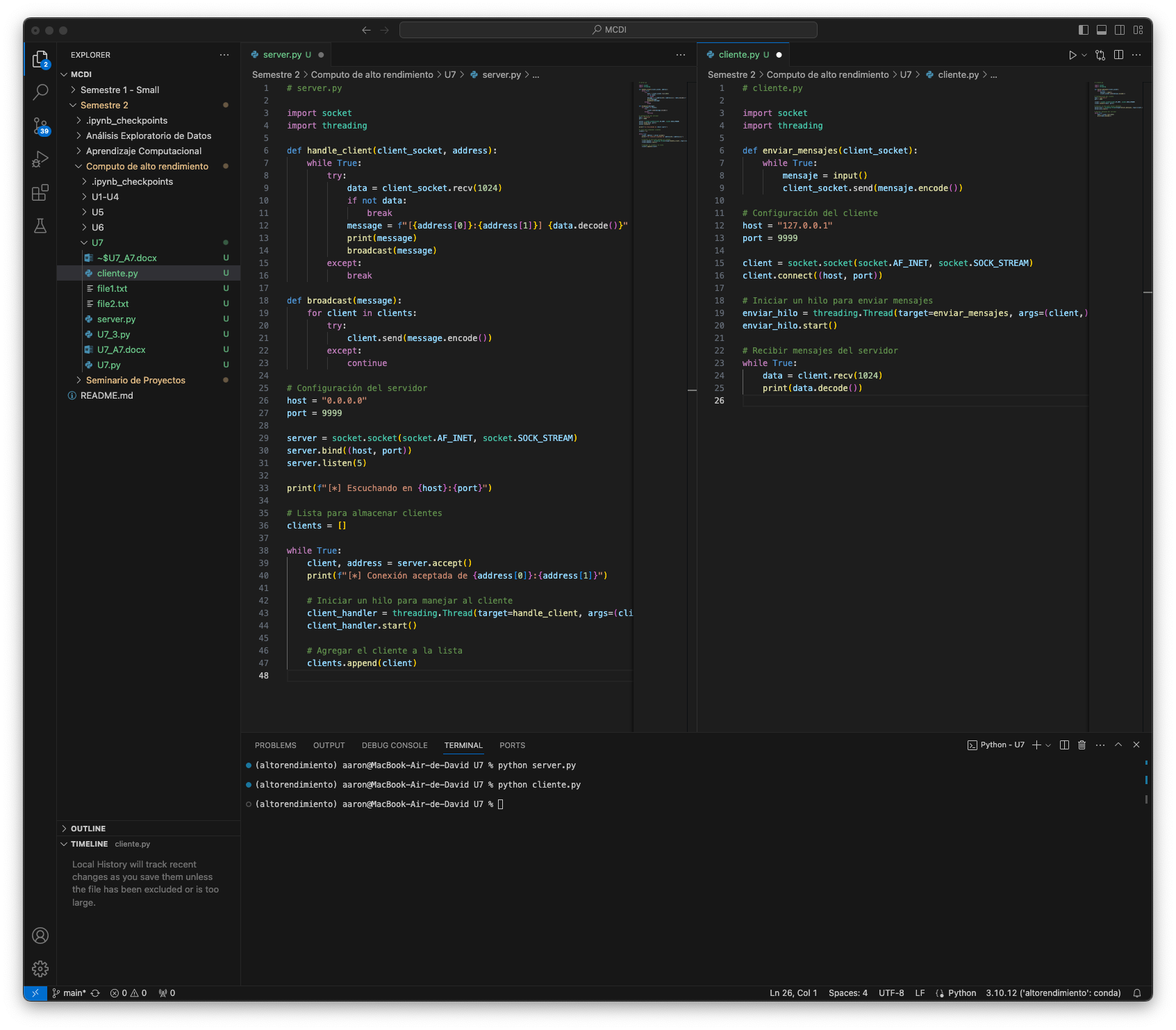
****

**4. Se les proporcionaran 2 códigos…**

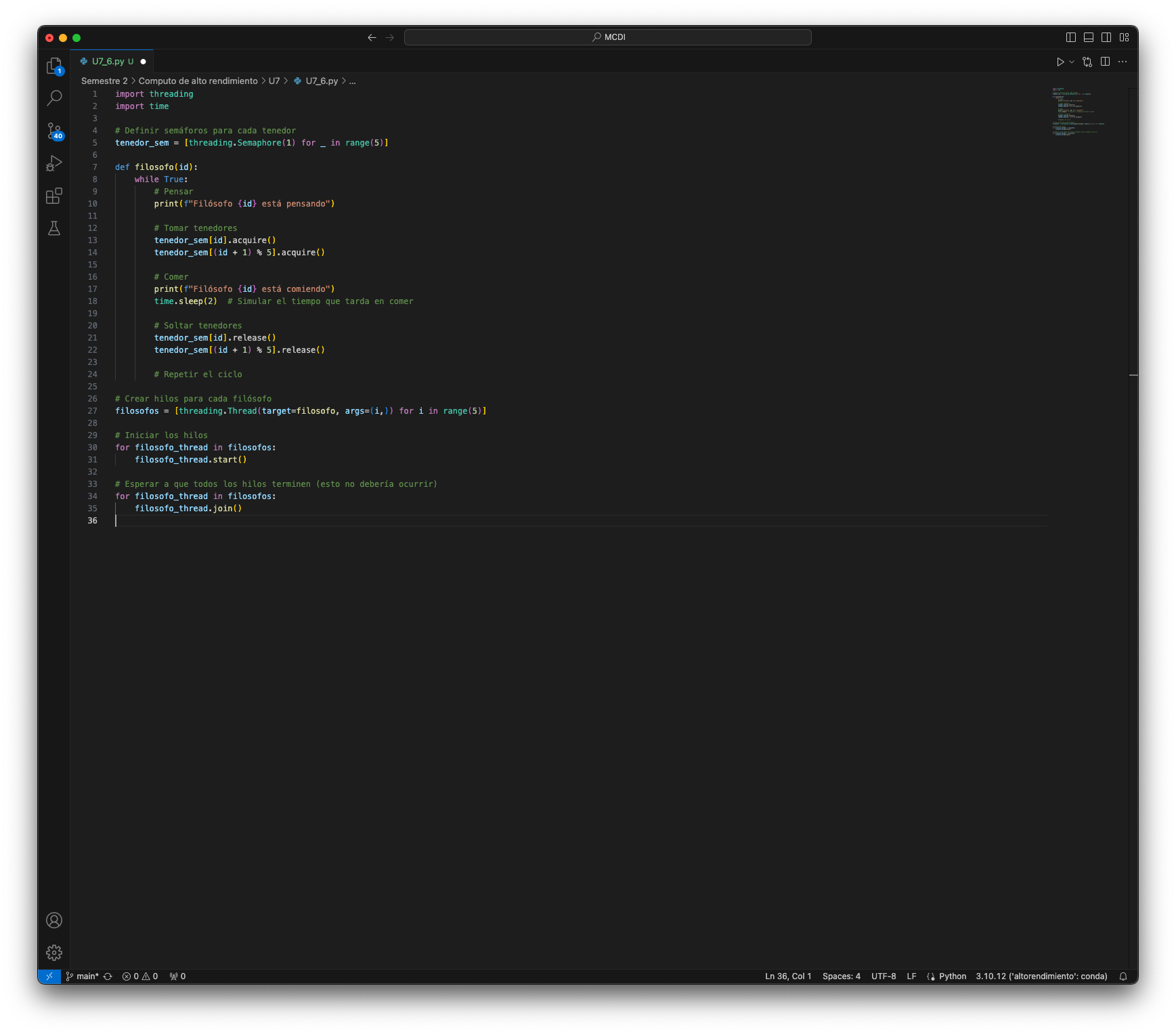
Primero, el server.py debería manejar múltiples conexiones de clientes.

Ahora, el cliente.py debe permitir al usuario enviar mensajes al servidor.

Aquí hay una versión de esto



**6. El problema de los filósofos comensales** es un clásico problema de concurrencia que se puede resolver utilizando semáforos o monitores para garantizar la exclusión mutua y evitar condiciones de carrera. A continuación, presento un pseudocódigo que implementa una solución al problema:



Este algoritmo utiliza semáforos para representar los tenedores y garantizar que cada filósofo pueda tomar dos tenedores contiguos para comer. La ventaja de esta solución es que evita el deadlock, ya que ningún filósofo puede quedar atrapado esperando un tenedor que otro filósofo está sosteniendo.

Sin embargo, esta solución puede tener el problema de inanición si un filósofo no puede adquirir ambos tenedores debido a la competencia con otros filósofos. Para mitigar esto, se pueden implementar soluciones más avanzadas, como permitir que los filósofos tomen tenedores solo cuando ambos estén disponibles o introducir un coordinador que regule el acceso a los tenedores.

**7. Algoritmo de Dekker:**

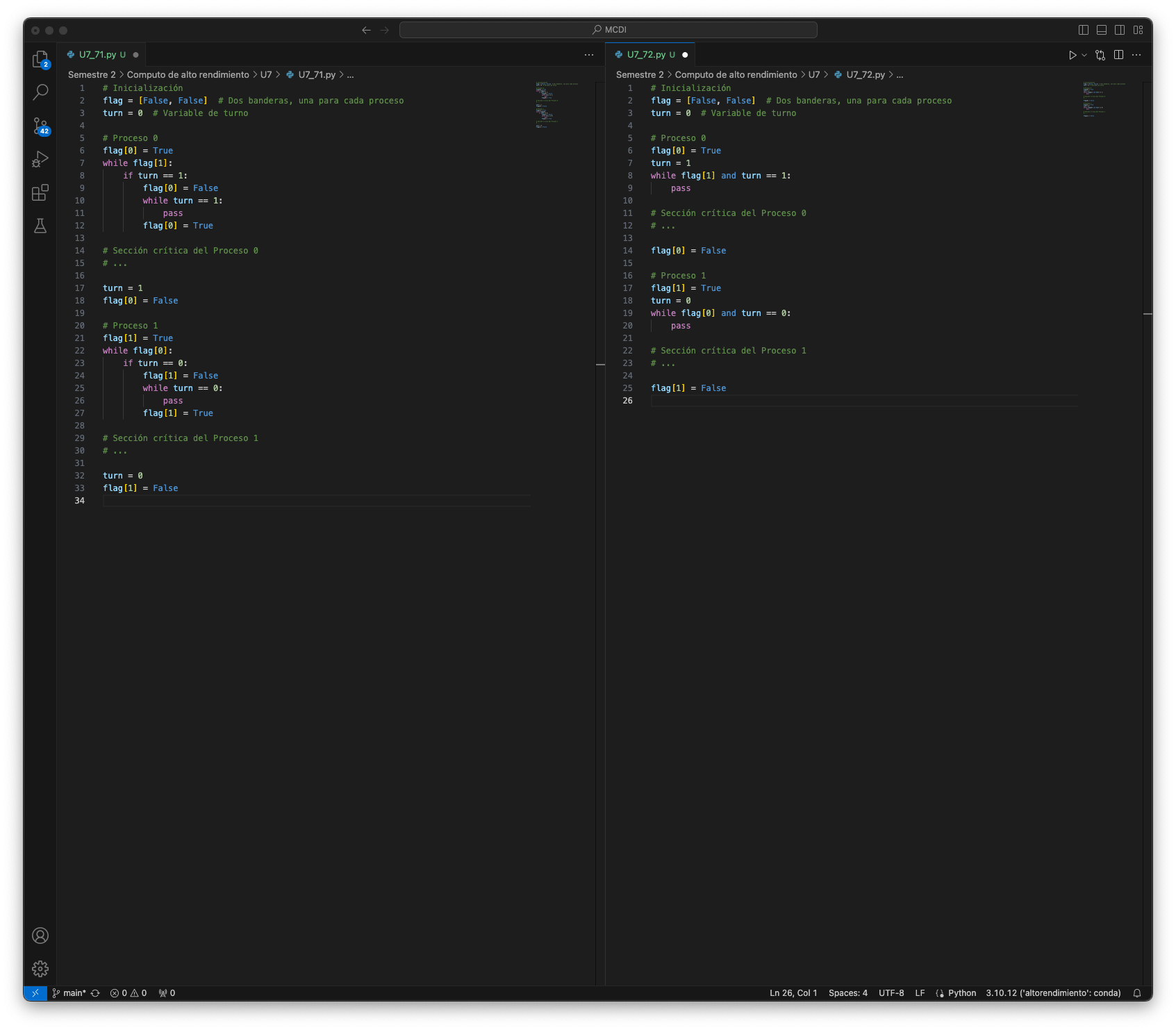
El algoritmo de Dekker es un algoritmo clásico de exclusión mutua diseñado para dos procesos que comparten recursos en un entorno de programación concurrente. Fue propuesto por Edsger W. Dijkstra en 1965. El objetivo principal del algoritmo es permitir que dos procesos accedan a una sección crítica de manera alternativa, sin interferencias.

En este algoritmo, cada proceso intenta entrar en la sección crítica alternativamente. Si un proceso quiere entrar y la bandera del otro proceso está activa, se pone en espera hasta que el otro proceso complete su sección crítica y libere la bandera.

**Solución de Peterson:**

La solución de Peterson es otro algoritmo para la exclusión mutua, pero a diferencia del algoritmo de Dekker, la solución de Peterson es más general y se puede aplicar a cualquier número de procesos. Fue propuesta por Gary L. Peterson en 1981.

En este algoritmo, cada proceso utiliza una bandera y una variable de turno. Los procesos esperan pacientemente hasta que su bandera está activa y es su turno antes de entrar en la sección crítica. La solución de Peterson garantiza que solo un proceso pueda estar en la sección crítica en un momento dado y evita la posibilidad de interbloqueo.



Ambos algoritmos (Dekker y Peterson) son ejemplos clásicos de cómo abordar el problema de la exclusión mutua en entornos concurrentes, y se han utilizado como base para el diseño de algoritmos más avanzados.