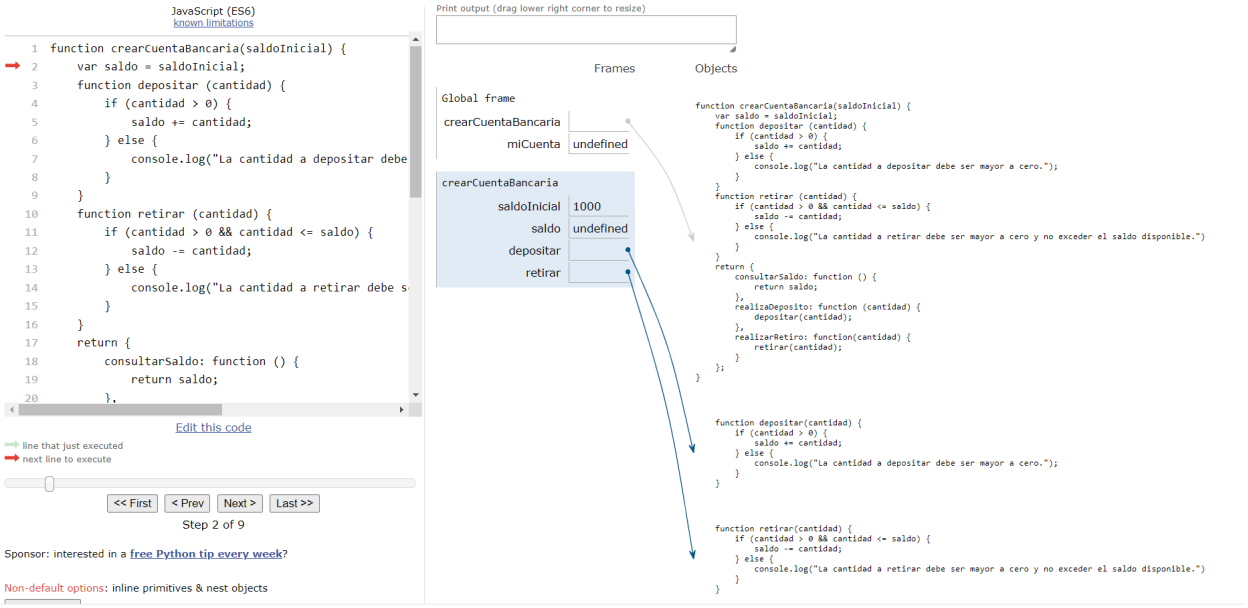
Tarea #8

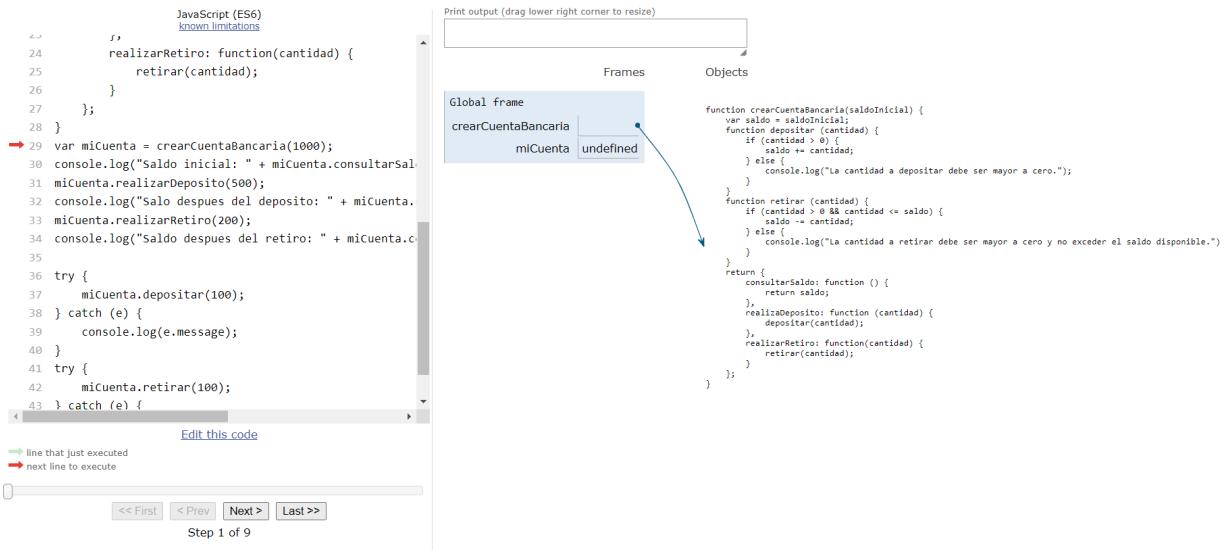
Función `crearCuentaBancaria(saldoInicial)`:

1. **Crea una variable `saldo` y la inicializa con el valor de `saldoInicial` recibido como parámetro.** Esta variable representa el saldo actual de la cuenta bancaria.

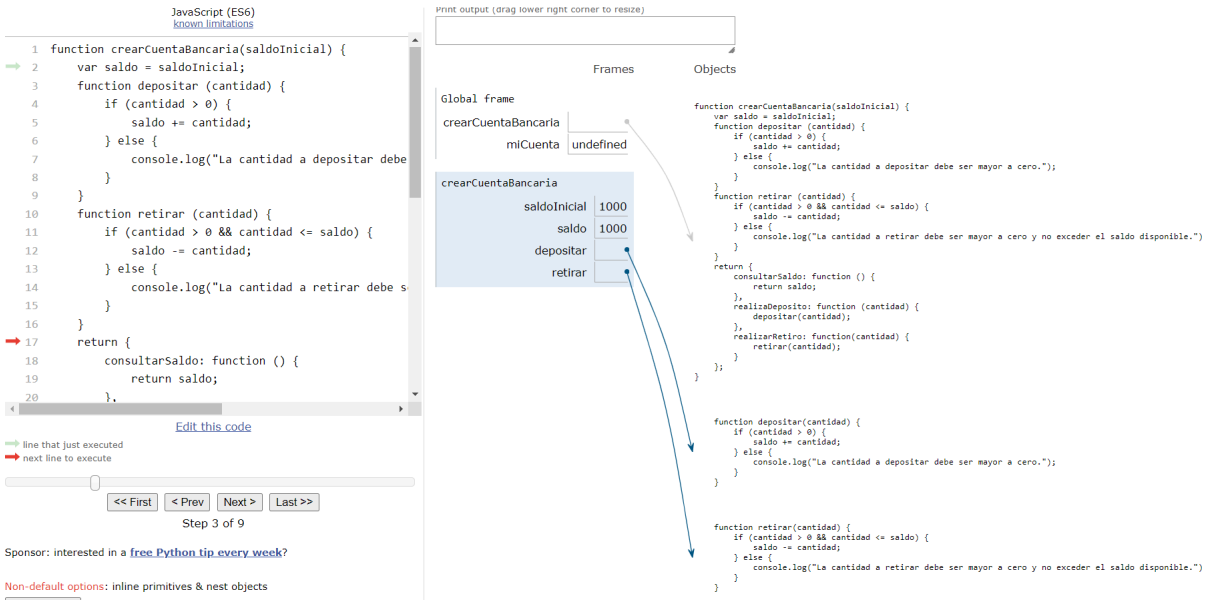


2. **Define dos funciones anidadas dentro de la función `crearCuentaBancaria`:**
   - **`depositar(cantidad)`:**
     - **Verifica si la `cantidad` a depositar es mayor que cero.** Si lo es, **suma la cantidad al saldo actual** utilizando el operador `+=`.
     - Si la `cantidad` es menor o igual a cero, **imprime un mensaje de error** indicando que la cantidad a depositar debe ser mayor a cero.
   - **`retirar(cantidad)`:**
     - **Verifica si la `cantidad` a retirar es mayor que cero y no excede el saldo actual.** Si lo es, **resta la cantidad al saldo actual** utilizando el operador `-=`.
     - Si la `cantidad` es menor o igual a cero, o si supera el saldo disponible, **imprime un mensaje de error** indicando que la cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

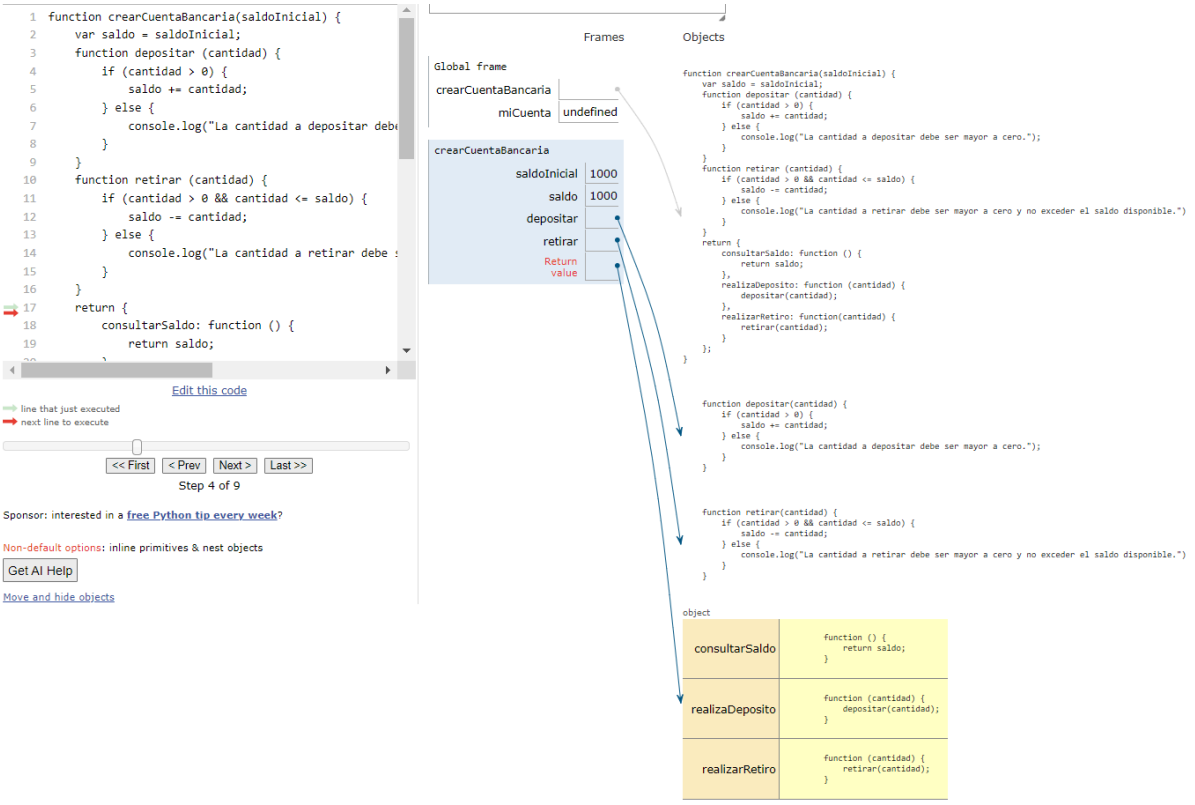


3. **Devuelve un objeto con las siguientes propiedades:**
    - **`consultarSaldo`:** Una función que **devuelve el valor actual del saldo**.
    - **`realizaDeposito(cantidad)`:** Una función que **llama a la función `depositar`** para depositar la `cantidad` especificada.
    - **`realizarRetiro(cantidad)`:** Una función que **llama a la función `retirar`** para retirar la `cantidad` especificada.

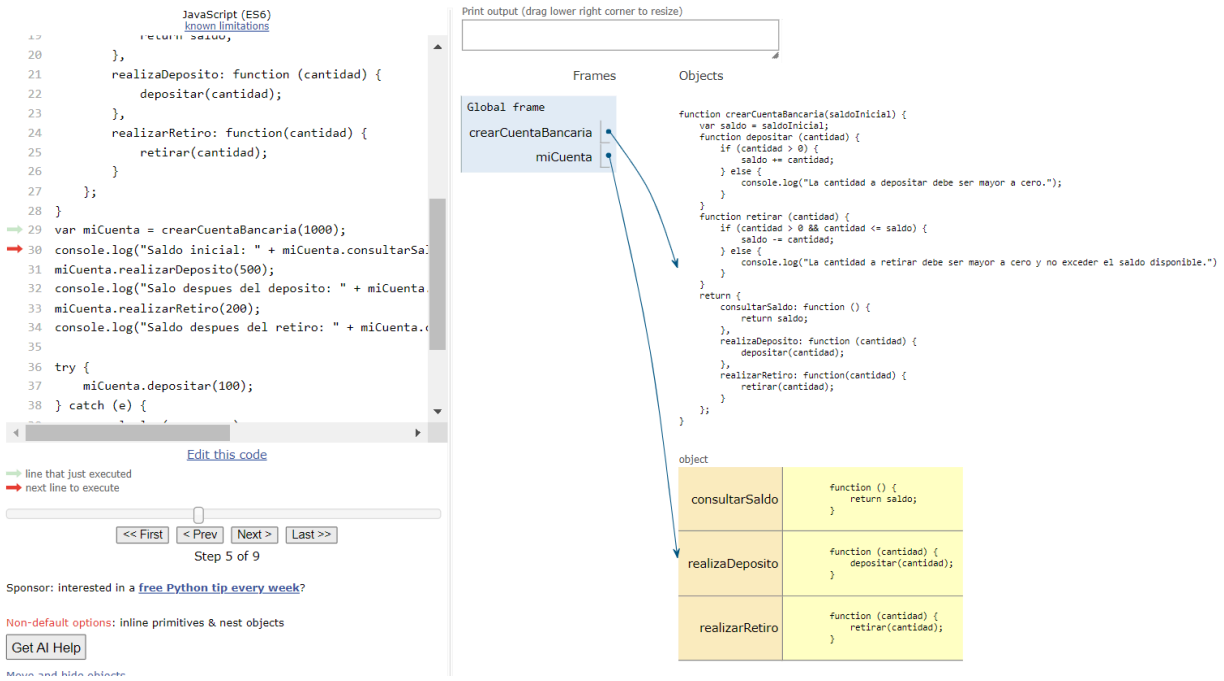


Creación de la cuenta y operaciones:

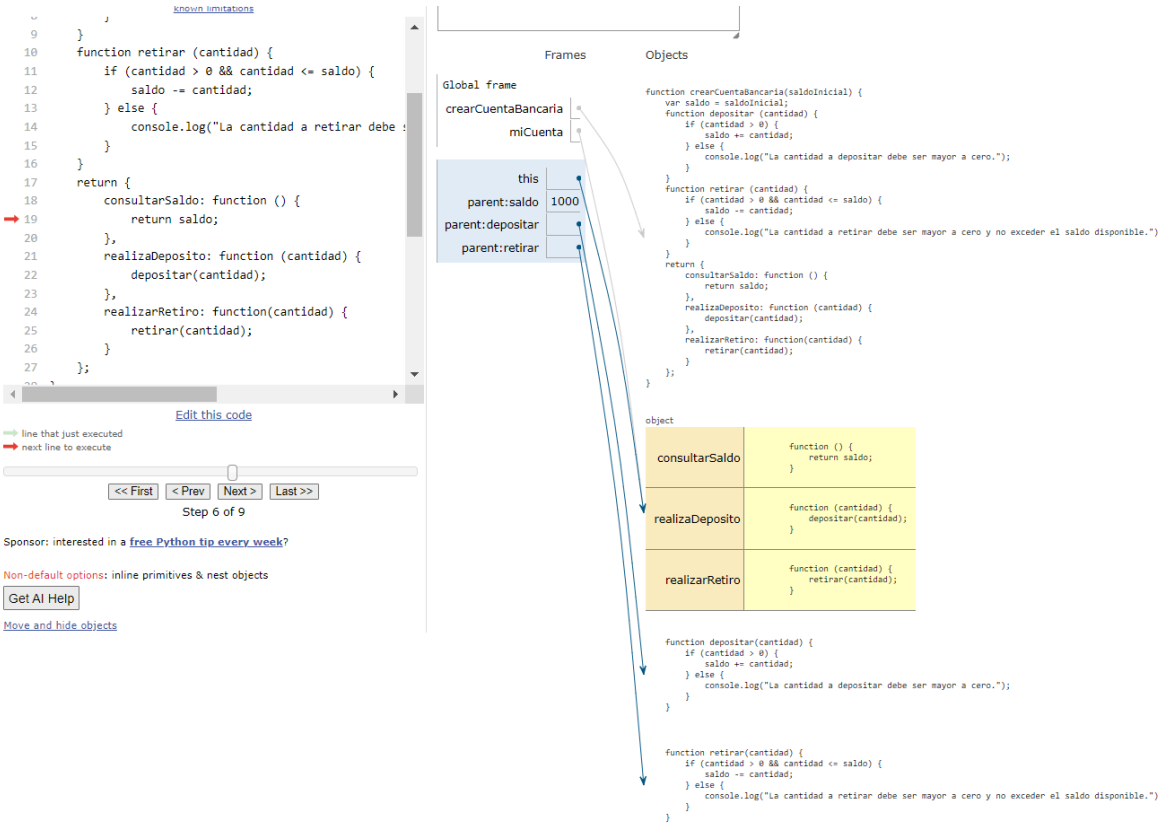
1. **Se crea una variable `miCuenta` llamando a la función `crearCuentaBancaria` y pasando un saldo inicial de 1000.** Esto crea una nueva cuenta bancaria con un saldo inicial de $1000.



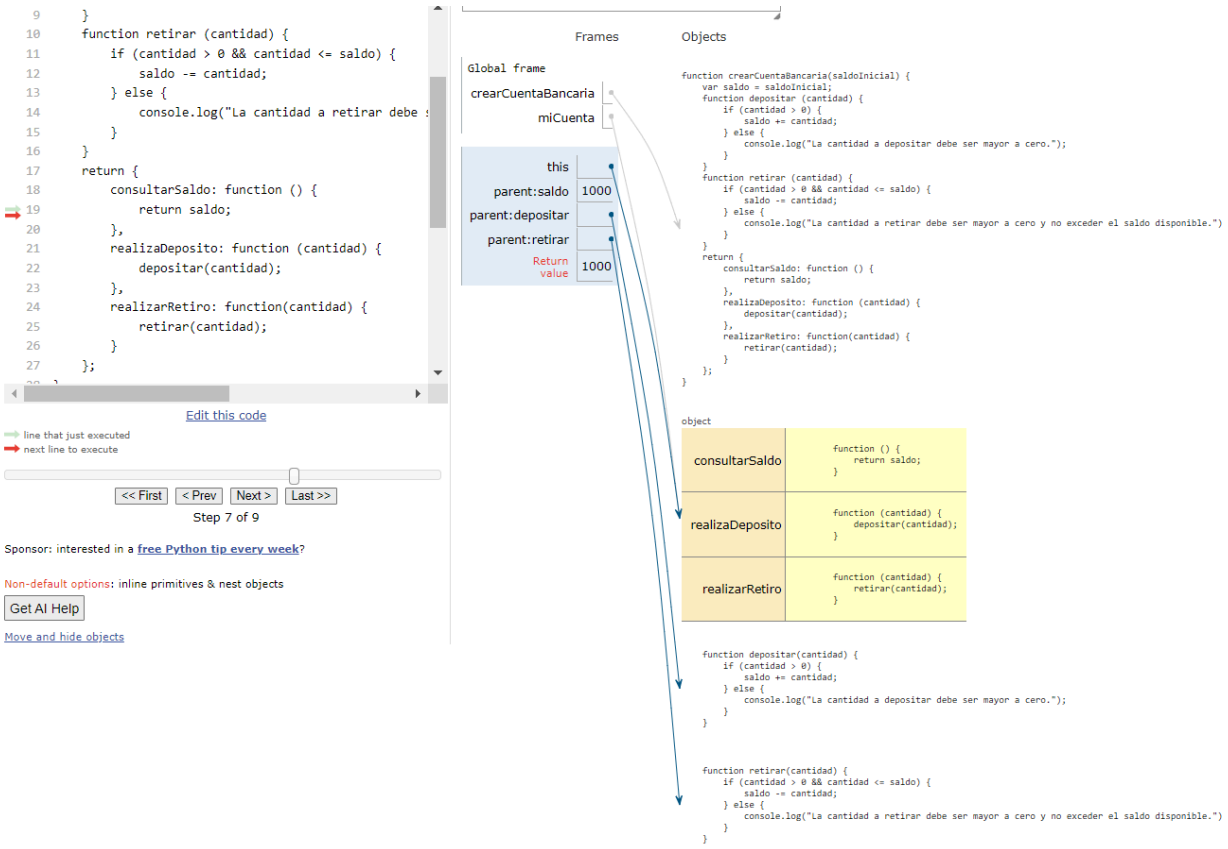
2. **Se imprime el saldo inicial:**
    - Se utiliza la propiedad `consultarSaldo` de la variable `miCuenta` para obtener el saldo actual, que es de $1000.
    - Se imprime un mensaje que muestra el saldo inicial: "Saldo inicial: $1000".



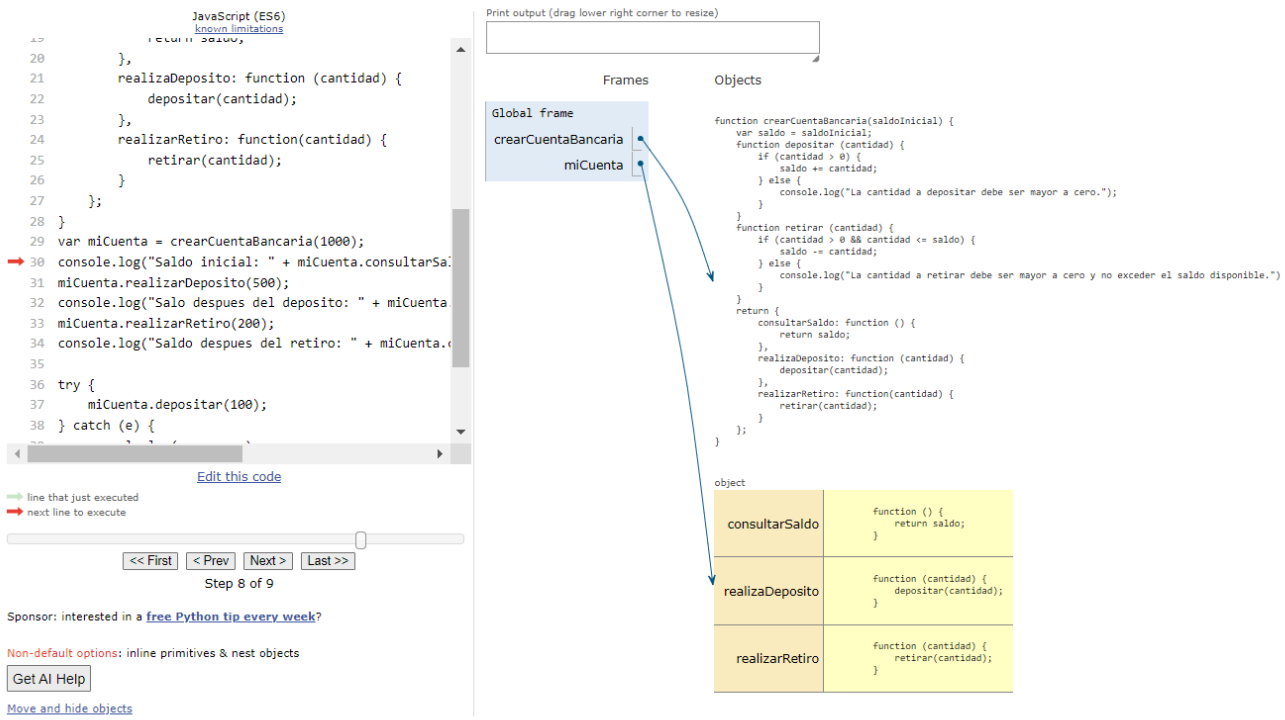
3. **Se realiza un depósito de $500:**
    - Se llama a la propiedad `realizaDeposito` de la variable `miCuenta` y se le pasa la cantidad $500 como argumento.
    - Esto llama a la función `depositar` internamente, aumentando el saldo en $500.



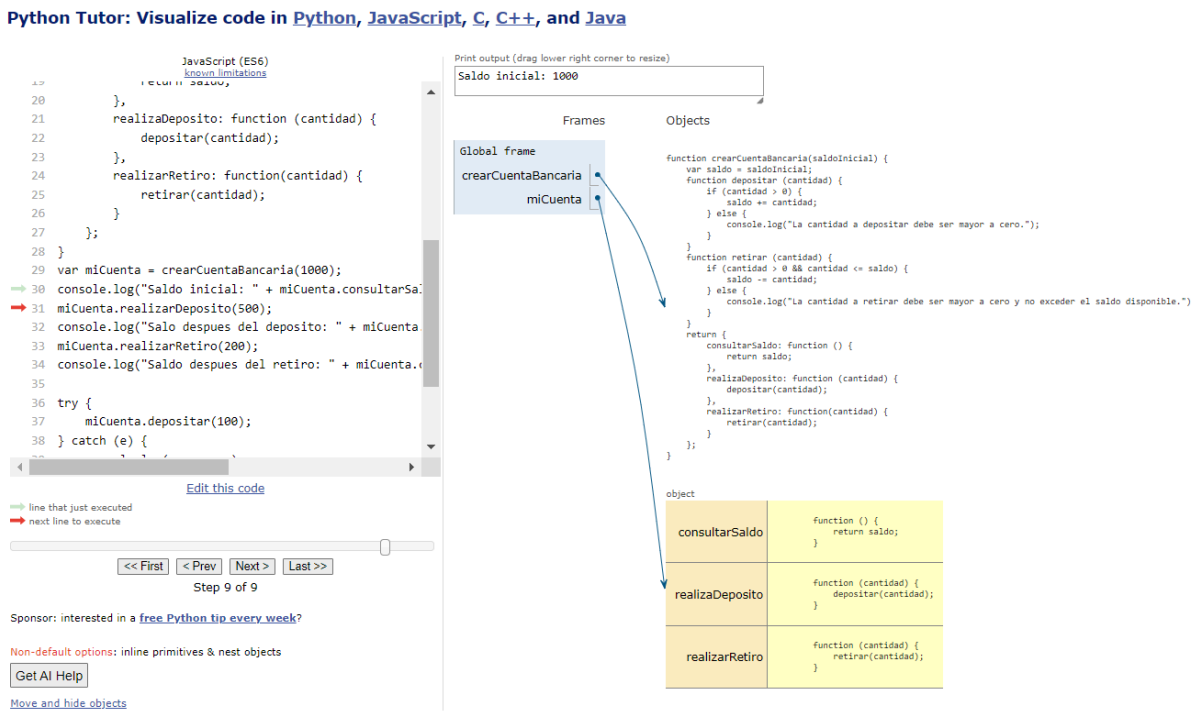
4. **Se imprime el saldo después del depósito:**
    - Se utiliza nuevamente la propiedad `consultarSaldo` para obtener el saldo actual, que ahora es de $1500.
    - Se imprime un mensaje que muestra el saldo después del depósito: "Saldo después del depósito: $1500".



5. **Se realiza un retiro de $200:**
    - Se llama a la propiedad `realizarRetiro` de la variable `miCuenta` y se le pasa la cantidad $200 como argumento.
    - Esto llama a la función `retirar` internamente, disminuyendo el saldo en $200.



6. **Se imprime el saldo después del retiro:**
    - Se utiliza nuevamente la propiedad `consultarSaldo` para obtener el saldo actual, que ahora es de $1300.
    - Se imprime un mensaje que muestra el saldo después del retiro: "Saldo después del retiro: $1300".



Manejo de errores:

1. **Se intenta realizar un depósito de $100 dentro de un bloque `try-catch`:**
   - Se llama a la propiedad `depositar` de la variable `miCuenta` y se le pasa la cantidad $100 como argumento.
   - Como la cantidad es mayor a cero, se debería depositar correctamente.
   - Sin embargo, dentro del bloque `catch`, se captura cualquier excepción que pueda ocurrir durante la ejecución de la función `depositar`.
2. **Se intenta realizar un retiro de $100 dentro de otro bloque `try-catch`:**
   - Se llama a la propiedad `realizarRetiro` de la variable `miCuenta` y se le pasa la cantidad $100 como argumento.
   - Como la cantidad es mayor a cero, pero supera el saldo disponible, se debería generar una excepción.
   - El bloque `catch` capturará la excepción y **imprimirá el mensaje de error correspondiente** indicando que la cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.