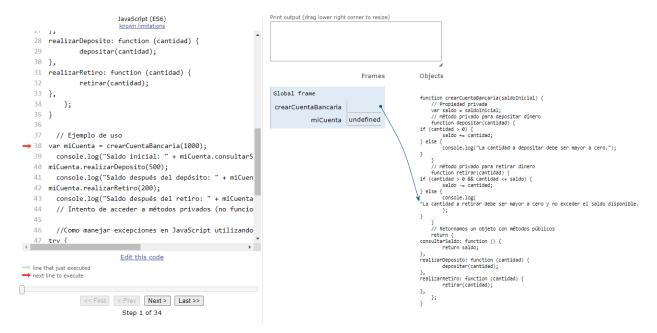
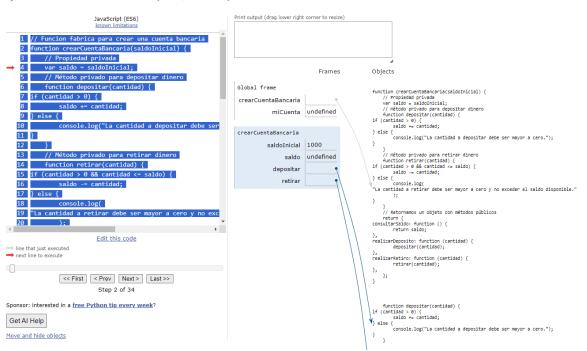
- 1. Inicia el recorrido del codigo
- Llamada a la Función crearCuentaBancaria: Llama a la función crearCuentaBancaria con el argumento 1000, que es el saldo inicial de la cuenta bancaria.

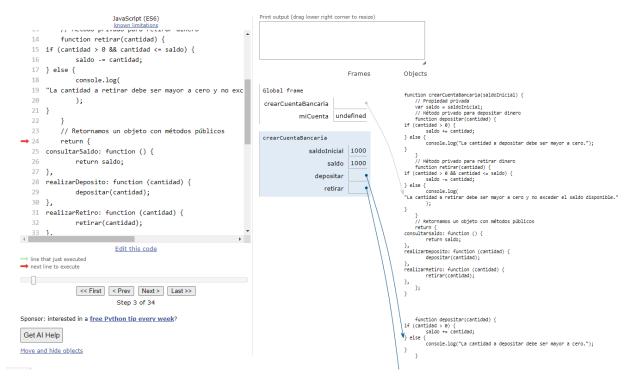


3. Declaración de la Variable saldo:

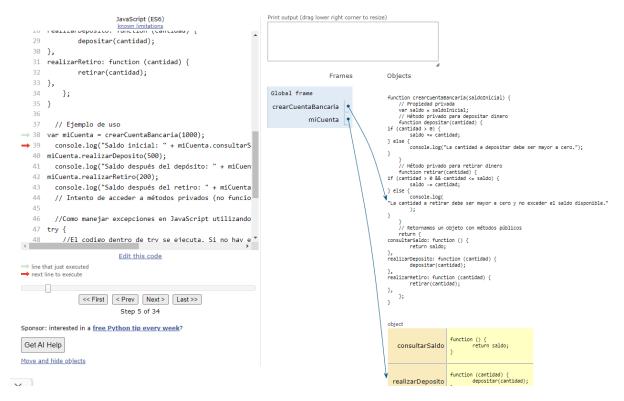


4. La línea 24 inicia el retorno de un objeto literal que contiene métodos públicos para interactuar con la cuenta bancaria. Este objeto permite a los usuarios consultar el saldo, realizar depósitos y retiros, mientras mantiene encapsulados los métodos privados depositar y retirar y la variable saldo.

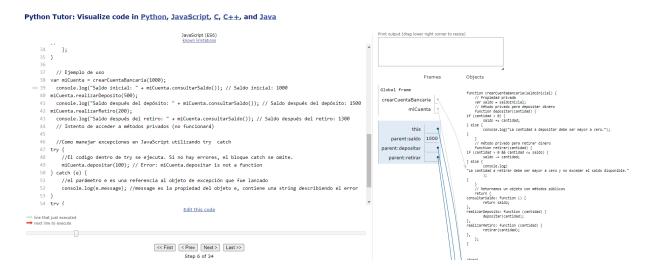




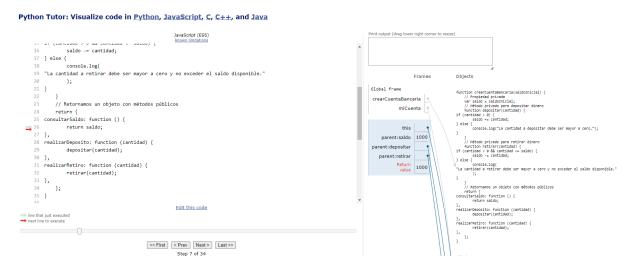
5. Ejecuta funcion crearCuentabancaria



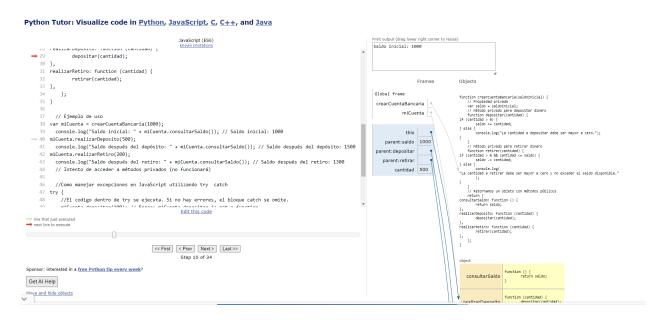
6. & 9La línea 39 imprime el saldo inicial de la cuenta bancaria en la consola. Utiliza el método consultarSaldo para obtener el saldo actual y luego imprime el resultado con un mensaje descriptivo. Esto confirma que la cuenta bancaria se ha creado correctamente con el saldo inicial especificado.



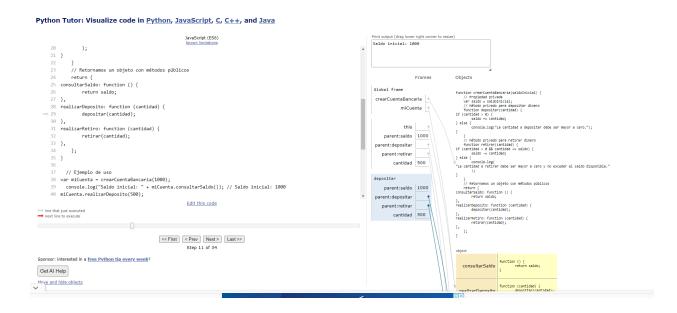
7. & 8La línea 26 devuelve el valor actual del saldo de la cuenta bancaria cuando se llama al método consultarSaldo. Este valor es utilizado por cualquier llamada a miCuenta.consultarSaldo() para obtener el saldo actual de la cuenta bancaria.



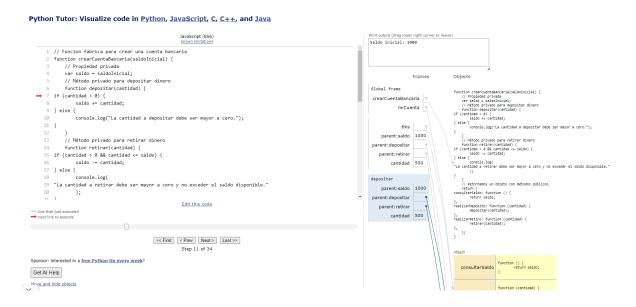
10. La línea 40 llama al método realizarDeposito del objeto miCuenta para depositar 500 unidades de dinero en la cuenta bancaria. Esta llamada actualiza el saldo de la cuenta sumando 500 al saldo actual, pasando de 1000 a 1500.



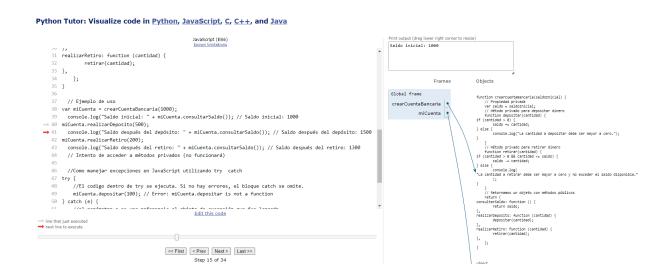
11. La línea 29 dentro del método realizarDeposito llama a la función privada depositar con la cantidad especificada, lo que permite agregar dinero al saldo de la cuenta bancaria. Esta llamada ejecuta la lógica de la función depositar para verificar y actualizar el saldo según la cantidad proporcionada.



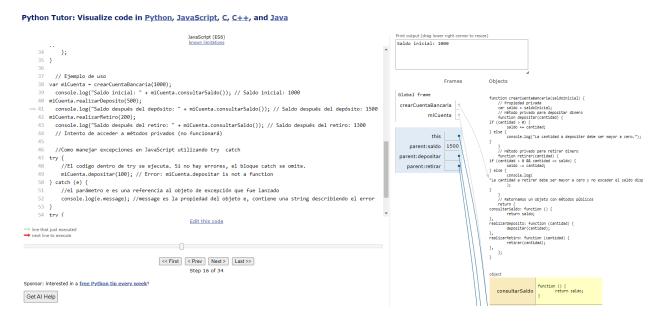
12. La línea 7 verifica si la cantidad a depositar es mayor que 0. Esta verificación asegura que solo se puedan realizar depósitos positivos en la cuenta bancaria. Si la cantidad es mayor que 0, se incrementa el saldo. Si no, se muestra un mensaje de error indicando que la cantidad a depositar debe ser mayor a cero.



- 13. Realiza confirmacion si cantidad es mayor a 0
- 14. Al confirmar que la cantidad es mayor a 0 agrega los 500
- 15. La línea 40 llama al método realizarDeposito del objeto miCuenta para depositar 500 unidades de dinero en la cuenta bancaria. Esta llamada actualiza el saldo de la cuenta sumando 500 al saldo actual, pasando de 1000 a 1500.



16. La línea 41 imprime el saldo actual de la cuenta bancaria en la consola después de realizar un depósito. Utiliza el método consultarSaldo para obtener el saldo actualizado y luego imprime el resultado con un mensaje descriptivo. Esto confirma que el depósito realizado en la línea 40 se ha aplicado correctamente y que el saldo ha aumentado de 1000 a 1500.



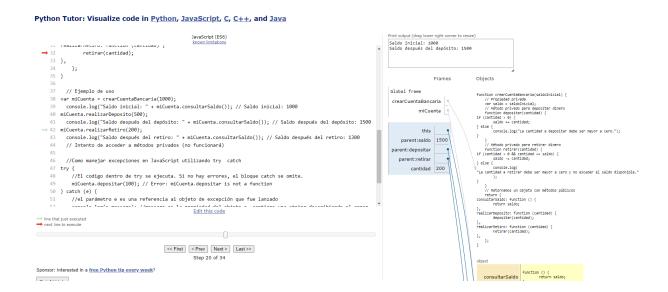
- 17. Ingresa a consultarSaldo para acceder al 'saldo'
- 18. Retorna el valor de 'consultarSaldo' y el valor de 'saldo'



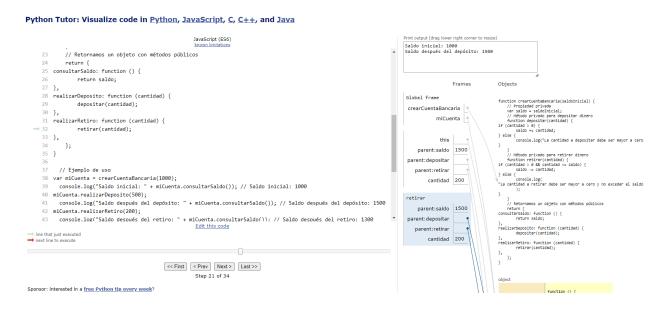
19. Imprime en consola el saldo



20. La línea 42 llama al método realizarRetiro del objeto miCuenta para retirar 200 unidades de dinero de la cuenta bancaria. Esta llamada actualiza el saldo de la cuenta restando 200 del saldo actual, pasando de 1500 a 1300.



21. La línea 32 dentro del método realizarRetiro llama a la función privada retirar con la cantidad especificada, lo que permite restar dinero del saldo de la cuenta bancaria. Esta llamada ejecuta la lógica de la función retirar para verificar y actualizar el saldo según la cantidad proporcionada.



22. La línea 15 verifica si la cantidad a retirar es mayor que 0 y no excede el saldo disponible en la cuenta bancaria. Si ambas condiciones se cumplen, permite la retirada del saldo restando la cantidad especificada. Si alguna condición no se cumple, imprime un mensaje de error. Esta verificación asegura que solo se puedan realizar retiros válidos y que no se pueda retirar más dinero del que hay en la cuenta.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java Saldo inicial: 1000 Saldo después del depósito: 1500 function depositar(cantidad) { 7 if (cantidad > 0) { 8 saldo += cantidad; 9 } else { console.log("La cantidad a depositar debe ser mayor a cero."); Global frame function crearcuenteBancaria(saldotnicial) (// Projeteda prisons // Pro crearCuentaBancaria | 9 12 } 13 // Método privado para retirar dinero 14 function retirar(cantidad) { 15 if (cantidad > 0 && cantidad <- saldo) { 16 saldo -- cantidad; 17 } else { 18 console.log(miCuenta 📍 parent:saldo 1500 // Método privado para retirar dinero function retirar(cintidad) { if (camidad > 8 di camidad < saldo) { saldo - camidad > saldo | s parent:depositar 19 "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible." 20 21 } parent:retirar cantidad 200 // Retornamos un objeto con métodos públicos } // Retornamos un objeto con métodos públicos 24 return { 25 consultarSaldo: function () { parent:saldo 1500 return { consultarSaldo: function () { return saldo; } parent:depositar parent:retirar }, realizarDeposito: function (cantidad) { depositar(cantidad); cantidad 200 }, realizarRetiro: function (cantidad) { retirar(cantidad); Step 22 of 34 Sponsor: interested in a <u>free Python tip every week</u>?

23. La línea 16 resta la cantidad especificada del saldo actual de la cuenta bancaria, actualizando el saldo después de una operación de retiro. Esta operación solo se realiza si la cantidad a retirar es válida y no excede el saldo disponible, según lo verificado en la línea 15. 24. El decremento se realiza y se imprime

Get Al Help

Move and hide objects

```
JavaScript (ES6)
                                                    known limitations
           // riccodo priivado para acposicar ainero
    6
           function depositar(cantidad) {
    7 if (cantidad > 0) {
    8
             saldo += cantidad;
    9 } else {
              console.log("La cantidad a depositar debe ser mayor a cero.");
    11 }
    12 }
    13 // Método privado para retirar dinero
    14 function retirar(cantidad) {
    15 if (cantidad > 0 && cantidad <= saldo) {
 16
             saldo -= cantidad;
    17 } else {
    18 console.log(
    19 "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
    20
    21 }
    22 }
    23 // Retornamos un objeto con métodos públicos
    24 return {
    25 consultarSaldo: function () {
               noturn colder
                                                   Edit this code
 ine that just executed
next line to execute
                                          << First | < Prev | Next > Last >>
                                                    Step 23 of 34
```

24. La línea 32 dentro del método realizarRetiro llama a la función privada retirar con la cantidad especificada, lo que permite restar dinero del saldo de la cuenta bancaria. Esta llamada ejecuta la lógica de la función retirar para verificar y actualizar el saldo según la cantidad proporcionada.

```
JavaScript (ES6)
                                                    known limitations
    23
          // Retornamos un objeto con métodos públicos
    24
          return {
    25 consultarSaldo: function () {
              return saldo;
    27 },
    28 realizarDeposito: function (cantidad) {
    29
              depositar(cantidad);
    30 },
    31 realizarRetiro: function (cantidad) {
 → 32
             retirar(cantidad);
 → 33 },
    34
           };
    35 }
    36
    37 // Ejemplo de uso
    38 var miCuenta = crearCuentaBancaria(1000);
    39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
    40 miCuenta.realizarDeposito(500);
    41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
    42 miCuenta.realizarRetiro(200);
        consolo log/"Saldo dospués dal notino: " : miCuanta consultanSaldo()): // Saldo dospués dal notino: 1300
                                                    Edit this code
ine that just executed
next line to execute
                                          Step 24 of 34
```

25. La línea 42 imprime el saldo actual de la cuenta bancaria en la consola después de realizar un retiro. Utiliza el método consultarSaldo para obtener el saldo actualizado y luego imprime el resultado con un mensaje descriptivo. Esto confirma que el retiro realizado en la línea 42 se ha aplicado correctamente y que el saldo ha disminuido de 1500 a 1300.

```
JavaScript (ES6)
                                                       known limitations
               rectrar (carretuau),
    33 },
    34
           };
    35 }
    36
    37 // Ejemplo de uso
    38 var miCuenta = crearCuentaBancaria(1000);
    39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
    40 miCuenta.realizarDeposito(500);
    41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500

→ 42 miCuenta.realizarRetiro(200);

 → 43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
    44 // Intento de acceder a métodos privados (no funcionará)
    45
    46 //Como manejar excepciones en JavaScript utilizando try catch
    47 try {
    48
           //El codigo dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
    49
           miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
    50 } catch (e) {
          //el parámetro e es una referencia al objeto de excepción que fue lanzado
    52
           console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
                                                       Edit this code
ine that just executed
next line to execute
                                             << First | < Prev | Next > | Last >> |
```

26. La línea 43 es un comentario que indica que los intentos de acceder a métodos privados (depositar y retirar) desde fuera del objeto miCuenta no funcionarán. El comentario sirve como explicación para el bloque de código que sigue, el cual demuestra que estos métodos no son accesibles directamente y cómo manejar los errores resultantes.

```
JavaScript (ES6)
                                                      known limitations
    34
            };
    35 }
    36
    37 // Ejemplo de uso
    38 var miCuenta = crearCuentaBancaria(1000);
    39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
    40 miCuenta.realizarDeposito(500);
    41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
    42 miCuenta.realizarRetiro(200);
  → 43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
         // Intento de acceder a métodos privados (no funcionará)
    44
    46
         //Como manejar excepciones en JavaScript utilizando try catch
    47 try {
           //El codigo dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
    48
    49
           miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
    50 } catch (e) {
          //el parámetro e es una referencia al objeto de excepción que fue lanzado
    52
            console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
    53 }
    54 try {
                                                       Edit this code
ine that just executed
next line to execute
```

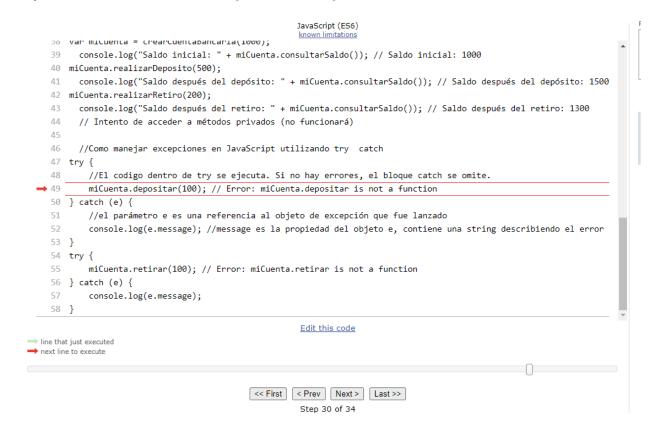
- 27. Accede a 'consultarSaldo' para verificar 'saldo'
- 28. Retorna el valor actual de 'saldo'



29. Devuelve el saldo en consola luego del retiro

```
JavaScript (ES6)
                                                    known limitations
    oo var micuenta = crearcuentabancaria(שטשו);
    39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
    40 miCuenta.realizarDeposito(500);
    41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
    42 miCuenta.realizarRetiro(200);
 → 43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
    44
        // Intento de acceder a métodos privados (no funcionará)
    46
        //Como manejar excepciones en JavaScript utilizando try catch
    47 try {
    48
        //El codigo dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
          miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
 → 49
    50 } catch (e) {
         //el parámetro e es una referencia al objeto de excepción que fue lanzado
          console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
    53 }
    54 try {
    55 miCuenta.retirar(100); // Error: miCuenta.retirar is not a function
    56 } catch (e) {
    57
        console.log(e.message);
    58 }
                                                     Edit this code
ine that just executed
next line to execute
                                           Step 29 of 34
```

30 & 31. Evalua Try y Catch para verificar si hubo algún error que se tenga que exceptuar de la recorrido del codigo



- 32. La línea 52 imprime el mensaje de error asociado con el objeto de error e en la consola. Este mensaje proporciona una descripción del error que ocurrió, ayudando a entender por qué falló la llamada dentro del bloque try. Esta línea es crucial para el manejo de errores y depuración en JavaScript.
- 33. Hace la captura e imprime

```
JavaScript (ES6)
                                                   known limitations
  o var micuenca = crearcuencabancaria(שמש);
  39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
  40 miCuenta.realizarDeposito(500);
  41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
  42 miCuenta.realizarRetiro(200);
       console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
       // Intento de acceder a métodos privados (no funcionará)
       //Como manejar excepciones en JavaScript utilizando try catch
  47 try {
  48
         //El codigo dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
  49
         miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
  50 } catch (e) {
        //el parámetro e es una referencia al objeto de excepción que fue lanzado

⇒ 52

         console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
 53 }
  54 try {
→ 55 miCuenta.retirar(100); // Error: miCuenta.retirar is not a function
  56 } catch (e) {
  57
         console.log(e.message);
  58 }
```

Edit this code

→ line that just executed
→ next line to execute

34. La línea 55 intenta llamar al método retirar del objeto miCuenta directamente. Dado que retirar es un método privado y no está expuesto como un método público del objeto miCuenta, esta llamada genera un error. Este error es capturado por el bloque catch, que luego imprime el mensaje de error correspondiente. Este ejemplo ilustra cómo los métodos privados no pueden ser accedidos directamente desde fuera de la función crearCuentaBancaria y cómo manejar tales errores en JavaScript.

```
console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
    40 miCuenta.realizarDeposito(500);
    41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
    42 miCuenta.realizarRetiro(200);
    43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
    44 // Intento de acceder a métodos privados (no funcionará)
    45
    46
        //Como manejar excepciones en JavaScript utilizando try catch
    47 try {
    48
           //El codigo dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
    49
           miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
    50 } catch (e) {
          //el parámetro e es una referencia al objeto de excepción que fue lanzado
    51
            console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
    53 }
    54 try {
 → 55
            miCuenta.retirar(100); // Error: miCuenta.retirar is not a function
    56 } catch (e) {
    57
            console.log(e.message);
    58 }
                                                       Edit this code
ine that just executed
next line to execute
                                            << First | < Prev | Next > | Last >>
                                                      Step 33 of 34
```

TypeFrror: miCuenta retirar is not a function