

## Depuración de Node.js (práctica)

Tiempo estimado: 45min

El objeto de esta práctica es afianzar, reforzar y consolidar los conocimientos teóricos presentados en la lección.

Al finalizarla, el estudiante:

- Habrá instalado la extensión **Node.js V8 Inspector** de **Chrome**.
- Habrá trabajado con el **inspector V8**, definido puntos de interrupción y observadores, ejecutado paso a paso, consultado la pila de llamadas y usado la consola de consulta.

### Objetivos

En esta práctica, vamos a crear un pequeño *script* **Node** de dos módulos con el que trabajar la depuración.

### Prerrequisitos

Para la realización de la práctica, es necesario **Chrome**. Recomendándose la última versión estable del navegador.

### Creación del proyecto

Para comenzar, vamos a crear el proyecto de la aplicación.

1. Abrir una consola.
2. Crear el directorio de la práctica e ir a él.
3. Crear el archivo **index.js** con el siguiente contenido:

```
console.log("¡Hola Mundo!");
```

4. Ejecutar el *script*:

```
$ node index.js
¡Hola Mundo!
$
```

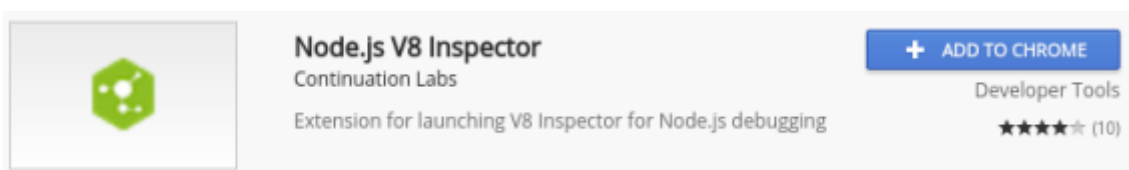
### Instalación de la extensión Node.js V8 Inspector

A continuación, vamos a instalar la extensión **Node.js V8 Inspector** para facilitar la conexión al inspector:

1. Abrir **Chrome**.
2. Ir a las extensiones: <chrome://extensions>.
3. Hacer clic en **Get more extensions** al final de la página:

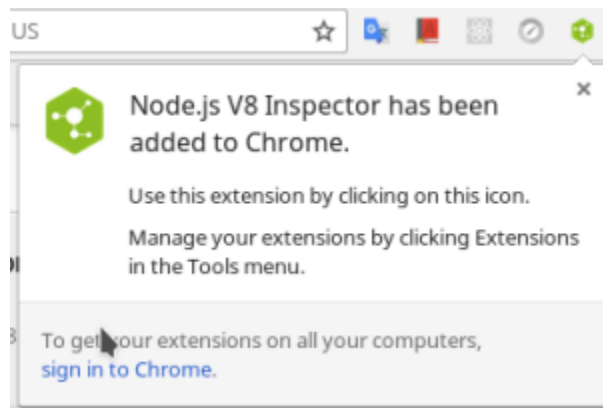


4. Buscar la extensión y hacer clic en **+ ADD TO CHROME**:



5. Confirme la solicitud de instalación.

A continuación, se muestra el botón que hay que usar para acceder a la extensión:



## Creación de un módulo de prueba

Vamos a crear un módulo de prueba que exporte una función con la que calcular el número de Fibonacci:

1. Crear el archivo `fibo.js`:

```
module.exports = exports = function fibo(n) {
  var res;

  if (n < 2) res = 1;
  else res = fibo(n - 1) + fibo(n - 2);

  return res;
};
```

## Prueba de depuración

A continuación, vamos a preparar el módulo principal para que dado un número, pasado en la línea de comandos, calcule su número de Fibonacci:

1. Modificar el archivo `index.js`:

```
//imports
const fibo = require("./fibo");

//data
const n = Number(process.argv[2]);

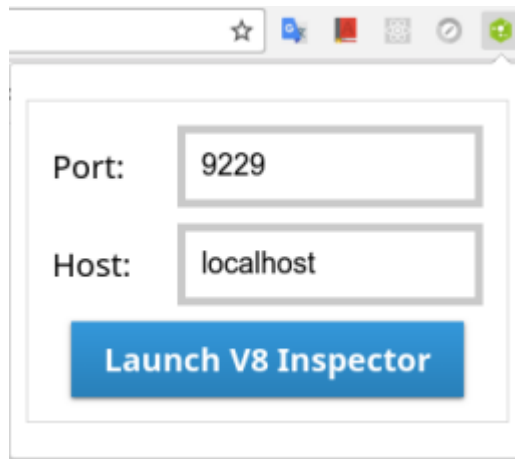
//main
console.log(fibo(n));
```

2. Guardar cambios.

3. Invocar el `script index.js` en modo depuración, esto es, bajo la monitorización del `inspector V8`:

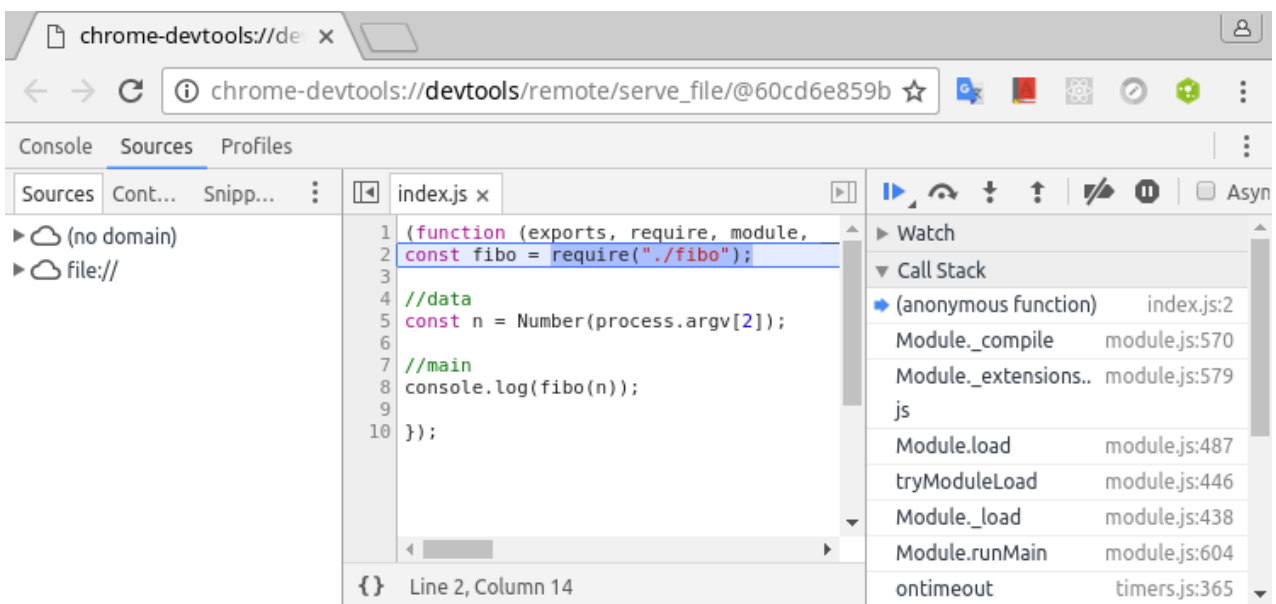
```
$ node --inspect --debug-brk index.js 15
Debugger listening on port 9229.
Warning: This is an experimental feature and could change at any time.
To start debugging, open the following URL in Chrome:
  chrome-
devtools://devtools/remote/serve_file/@60cd6e859b9f557d2312f5bf532f6aec5f284980/inspect
or.html?experiments=true&v8only=true&ws=localhost:9229/ba9ceace-bee2-4c68-a5bf-
74ec4d059161
Observe que node indica en qué puerto está escuchando el inspector, 9229, y el URL a utilizar para acceder al inspector.
```

4. Abrir `Chrome`.
5. Hacer clic en la extensión `Node.js V8 Inspector`:



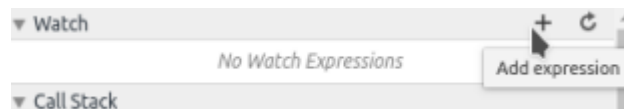
6. Hacer clic en **Launch V8 Inspector**.

La extensión conectará al puerto indicado y abrirá **Chrome DevTools** con el *script* bajo depuración:



Abrir **Chrome DevTools** también es posible mediante el URL **chrome-devtools** que muestra **node**. Pero es más fácil hacerlo mediante esta extensión.

7. Desplegar todos los paneles para ver su contenido: **Watch**, **Call Stack**, **Scope**, **Breakpoints**...
8. Añadir un observador para visualizar el valor de la variable **n**. Para ello, hacer clic en el botón **+** del panel **Watch**:



A continuación, escribir **n** y pulsar **INTRO**:



9. Añadir un punto de interrupción manual en la línea 8 del archivo **index.js**. Para ello, editar el archivo en **DevTools** y hacer clic en el número de línea:

```

index.js x
1 {function (exports, require, module, filename,
2 const fibo = require("../fibo");
3
4 //data
5 const n = Number(process.argv[2]);
6
7 //main
8 console.log(fibo(n));
9
10 });

```

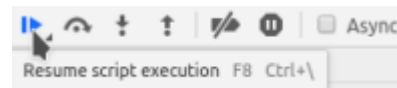
- Ir al panel **Breakpoints** y observar que se lista el punto de interrupción:

```

▼ Breakpoints
☑ index.js:8
  console.log(fibo(n));

```

- Ir al panel de ejecución paso a paso. Sin pulsar ningún comando, mostrar sus respectivas ayudas. Para ello, situar el cursor del ratón encima de cada botón y esperar a que la ayuda aparezca:



Resume script execution F8 Ctrl+\

- Hacer clic en el comando **Resume script execution (F8)** para que ejecute hasta el primer punto de interrupción:

```

index.js x
1 {function (exports, require, module, filename,
2 const fibo = require("../fibo"); fibo = fibo(n)
3
4 //data
5 const n = Number(process.argv[2]); n = 15
6
7 //main
8 console.log(fibo(n));
9
10 });

```

La línea sombreada indica la línea en la que estamos detenidos. Su ejecución *todavía no* se ha llevado a cabo.

- Consultar detenidamente los paneles **Watch**, **Call Stack**, **Scope** y **Breakpoints**.

El observador muestra el valor actual de la variable `n`. Antes era `undefined`, ahora 15.

En **Scope**, encontramos las variables del contexto de ejecución que hay en la línea actual. Por un lado, tenemos el contexto local, bajo **Local**; y el global, bajo **Global**:

```

▼ Scope
► Local
► Global

```

Desplígelos.

El panel **Breakpoints** muestra los puntos de interrupción definidos. El sombreado en amarillo es en el que nos encontramos actualmente.

- Ejecutar el comando de ejecución paso a paso **Step into next function call (F11)**.

Si no recuerda cuál es, sitúese encima de los botones del panel de ejecución paso a paso y espere a que salga la ayuda.

Recordemos que este comando tiene como objeto ejecutar la línea actual y entrar en la primera función invocada dentro de ella, deteniéndose en la primera línea del cuerpo de la función invocada.

Como puede observar, se mete en el archivo `bootstrap_node.js`, un módulo *core* que nada tiene que ver con lo que nosotros deseamos depurar. Está fuera de nuestro ámbito. Vamos a indicárselo al depurador mediante la ocultación o *blackboxing*.

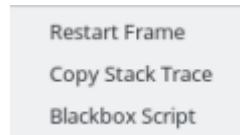
- Ir al panel **Call Stack**.

- Seleccionar el marco de pila activo, o sea, el que se encuentra arriba del todo y se encuentra

señalado por una flecha:

| Call Stack             |                       |
|------------------------|-----------------------|
| get                    | bootstrap_node.js:253 |
| (anonymous function)   | index.js:8            |
| Module._compile        | module.js:570         |
| Module._extensions..js | module.js:579         |
| Module.load            | module.js:487         |
| tryModuleLoad          | module.js:446         |

- Mostrar el menú contextual del marco, haciendo clic en el botón derecho del ratón, y seleccionar **Blackbox Script**:

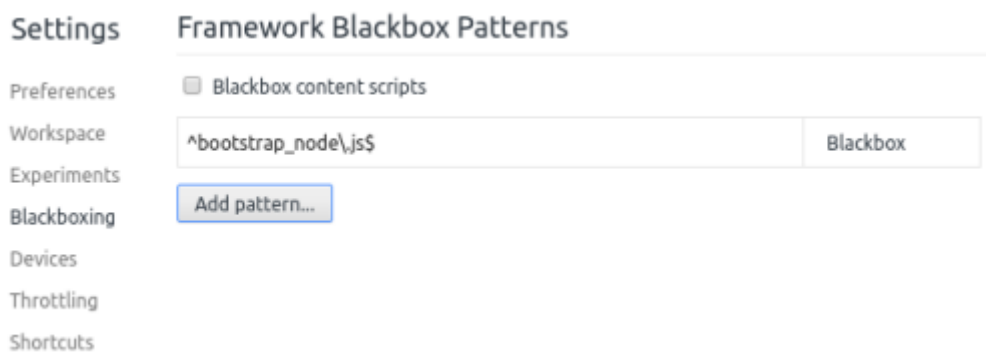


Esto creará un patrón en la lista de ocultación. La cual podemos consultar en cualquier momento. Para ello, pulsar **F1** y seleccionar **Blackboxing**:

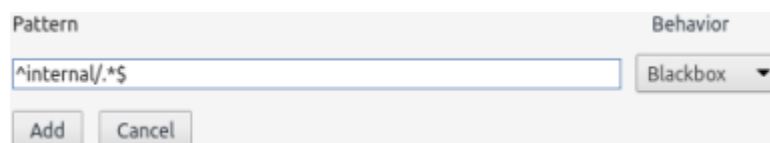
### Settings

- Preferences
- Workspace
- Experiments
- Blackboxing
- Devices
- Throttling
- Shortcuts

Aparecerá la lista de patrones de ocultación:



- Crear un nuevo patrón para los archivos ubicados en la carpeta internal. Para ello, hacer clic en **Add pattern...** e indicar el patrón:

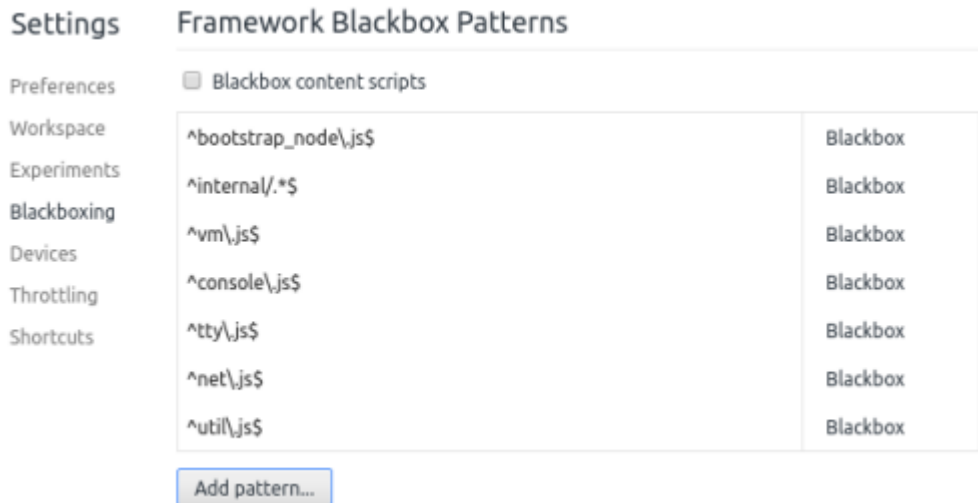


A continuación, hacer clic en **Add** para añadirlo a la lista.

- Añadir el patrón para el archivo vm.js:

20. Añadir los siguientes patrones:

```
^console\\.js$
^tty\\.js$
^net\\.js$
^util\\.js$
```



Si durante la práctica aparece algún otro módulo que no sea `index.js` o `fibonacci.js`, añádalo a la lista.

21. Cerrar **Chrome**.

22. Ir a la consola.

23. Ejecutar de nuevo el *script* en modo depuración:

```
$ node --inspect --debug-brk index.js 15
```

24. Abrir **Chrome** y el inspector mediante la extensión **Node.js V8 Inspector**.

25. Hacer clic en el comando paso a paso **Resume script execution (F8)** para que ejecute hasta el primer punto de interrupción.

26. Ejecutar el comando de ejecución paso a paso **Step into next function call (F11)**.

Observe que el inspector se queda en la misma línea, pero resalta la invocación a la función `fibonacci()`:

```
1 (function (exports, require, module, __filename,
2 const fibonacci = require("../fibonacci"); fibonacci = fibonacci(n)
3
4 //data
5 const n = Number(process.argv[2]); n = 15
6
7 //main
8 console.log(fibonacci());
9
10 });
```

El comando ha entrado en la proposición de la línea interrumpida y se ha quedado en la invocación, pero todavía no la ha realizado.

27. Volver a ejecutar el comando de ejecución paso a paso **Step into next function call (F11)**.

Esta vez, entramos en el módulo `fibonacci.js`, donde tenemos definida la función `fibonacci()` y se queda en su primera línea:

```

1 (function (exports, require, module, __filename,
2   var res; res = undefined
3
4   if (n < 2) res = 1;
5   else res = fibo(n - 1) + fibo(n - 2);
6
7   return res;
8 });
9
10 });

```

28. Consultar el panel **Call Stack** y observar que se ha añadido el marco de pila para la llamada a `fibo()`:

| Call Stack           |               |
|----------------------|---------------|
| fibo                 | fibo.js:4     |
| (anonymous function) | index.js:8    |
| Module_compile       | module.js:570 |

La proposición que invocó a la función se representa mediante su propio marco. Justo el que está debajo del activo. Observe que a la derecha aparece el nombre del archivo y la línea.

29. Ejecutar el comando **Step into next function call (F11)**.

A medida que vaya ejecutando, mantenga desplegados los paneles **Watch**, **Call Stack** y **Scope** para ver cómo muestran la situación de cada momento. **Watch** mostrará que el valor de la variable `n` va reduciéndose tras cada invocación a `fibo()`. El panel **Call Stack** mostrará los marcos de pila para cada invocación. Recuerde que la función `fibo()` es recursiva y se debe ejecutar varias veces. Cada invocación se representará mediante su propio marco. Siendo el que está arriba del todo el activo.

Repítalo unas 10 veces. Sea paciente y observador.

30. Ejecutar el comando **Step over next function call (F10)** unas 10 veces.

Observe los paneles.

31. Ejecutar el comando **Step out of current function (Shift+F11)**.

Con este comando, se sale del marco de pila activo. Continúe hasta que la pila deje de tener marcos de pila para la función `fibo()`. Esto es, hasta que vuelva al archivo `index.js`, donde se invocó la función recursiva `fibo()`:

```

1 (function (exports, require, module, __filename, __dirname)
2   const fibo = require("../fibo"); fibo = function fibo(n)
3
4   //data
5   const n = Number(process.argv[2]); n = 15
6
7   //main
8   console.log(fibo(n));
9
10  });

```

32. Ejecutar el comando **Resume script execution (F8)**.

Observe que se alcanza el final del *script*. El observador de la variable `n` se encuentra a `undefined` y la pila de llamadas se encuentra vacía.

33. Cerrar **Chrome**.

Al cerrar **Chrome**, se cerrará la sesión de depuración.

34. Ir a la consola.

35. Observar el resultado impreso por el *script*.

987

## Definición de puntos de interrupción híbridos

El objetivo ahora es mostrar cómo definir puntos de interrupción híbridos. Recordemos, aquellos que se

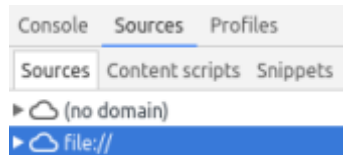
asocian a una determinada línea e incorporan una condición.

1. Ejecutar el *script* a depurar:

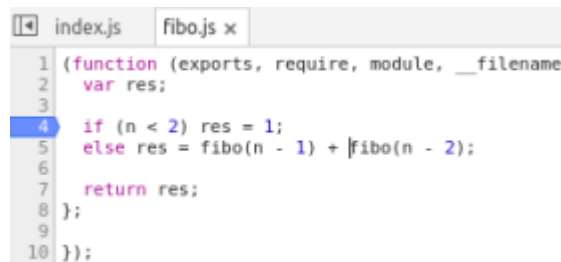
```
$ node --inspect --debug-brk index.js 15
```

2. Abrir **Chrome** y, a continuación, el inspector mediante la extensión **Node.js V8 Inspector**.
3. Ejecutar el comando paso a paso **Resume script execution (F8)**.
4. Ir al archivo `fibonacci.js`.

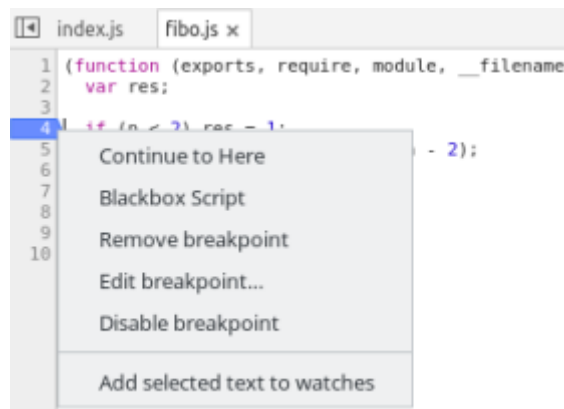
Para ello, vaya al panel **Sources**, donde se encuentran los archivos de código fuente abiertos por el inspector hasta ese momento, y búsquelo bajo el nodo `file://`:



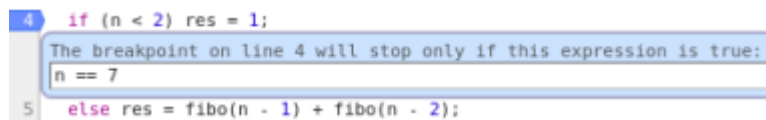
5. Añadir un punto de interrupción en la línea 4:



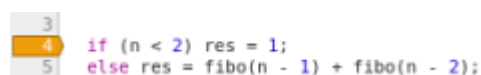
6. Mostrar el menú contextual del punto de interrupción y seleccionar **Edit breakpoint...**:



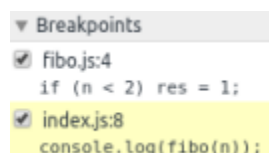
7. Añadir la expresión de interrupción siguiente:



Esto hará que el depurador se detenga en esa línea sólo cuando el valor del parámetro `n` sea 7. Observe que el marcador del punto de interrupción ha cambiado de color:



8. Ir al panel **Breakpoints** para ver los puntos de interrupción definidos:





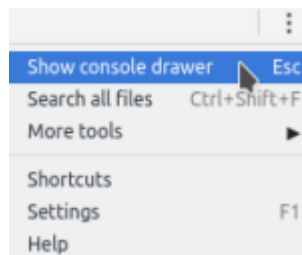
9. Volver al archivo `index.js`.
10. Ejecutar el comando paso a paso `Resume script execution (F8)` para que el inspector ejecute hasta el siguiente punto de interrupción. En nuestro caso, ubicado en el archivo `fibonacci.js`.  
Observe los paneles `Watch` y `Call Stack`. Si no hubiese sido un punto de interrupción híbrido, se habría parado en la primera invocación de la función. Pero lo ha hecho en aquella que tiene el valor de `n` igual a 7, según establece su condición adjunta.
11. Volver a ejecutar el comando paso a paso `Resume script execution (F8)` para que el inspector ejecute hasta el siguiente punto de interrupción. Repetirá el mismo. Ejecútelo tantas veces como sea necesario hasta terminar. Observe que en todas las ejecuciones `n` vale 7, de ahí que se pare. Pero cuando deja de valer 7, ya no se detiene.
12. Cerrar `Chrome`.

## Consola de consulta

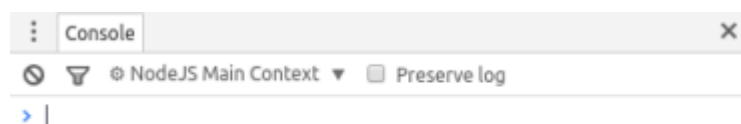
A continuación, vamos a ver cómo usar la consola del inspector como otro medio para consultar el contexto de ejecución en el que se encuentra detenida la ejecución:

1. Ejecutar el `script` a depurar:  
`$ node --inspect --debug-brk index.js 15`
2. Abrir `Chrome` y el inspector mediante la extensión `Node.js V8 Inspector`.
3. Mostrar la consola de consulta.

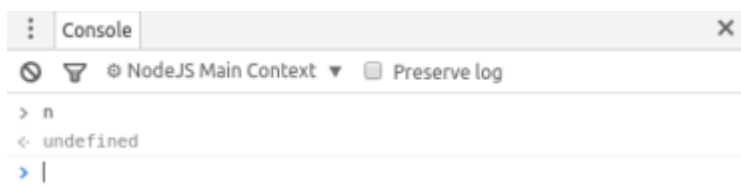
Para ello, se puede pulsar la tecla `Esc` o bien seleccionar `Show console drawer` del menú del depurador:



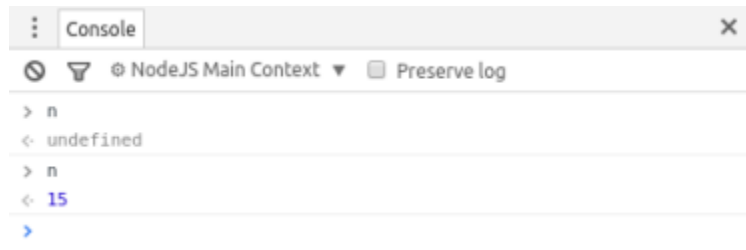
Esto mostrará una consola similar al del intérprete interactivo de `node`:



4. Consultar el valor de la variable `n`:



5. Ejecutar el comando paso a paso `Resume script execution (F8)`.
6. Ir a la consola.
7. Consultar el valor de la variable `n`:



8. Cerrar **Chrome**.