

Debug o depuración

DigitalHouse >
Coding School



**Certified Tech
Developer**
The Ultimate Degree

Índice

1. [¿Qué es “debug”?](#)
2. [Breakpoints](#)
3. [Debug en Chrome](#)
4. [Debug en Visual Studio Code](#)

1 | ¿Qué es “debug”?

“

Llamamos **debuggear o depurar** al proceso de encontrar, analizar y remover las causas de fallos en el software.

Se realiza la ejecución paso a paso de cada instrucción del programa para analizar las variables y sus valores.

”



2 | Breakpoints

“ Un **breakpoint** es un punto de interrupción en nuestro código para detener la ejecución del programa en líneas específicas y analizar la situación del mismo, revisando por ejemplo el estado de las variables o de la pila de llamadas en ese momento. ”



Debug

Se puede realizar el debug de una aplicación utilizando:

- Las herramientas del desarrollador desde la consola del navegador, ej: Chrome Dev Tools, Firefox Dev Tools
- La opción Debug dentro del framework o IDE utilizado para el desarrollo, ej: Visual Code, Visual Studio.

3 | Debug en Chrome

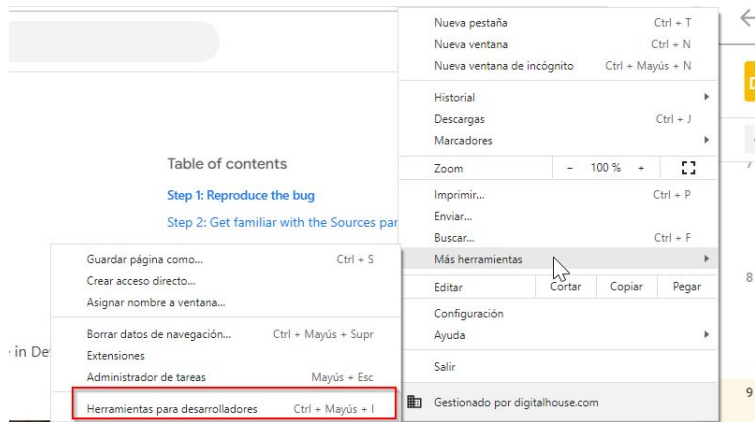
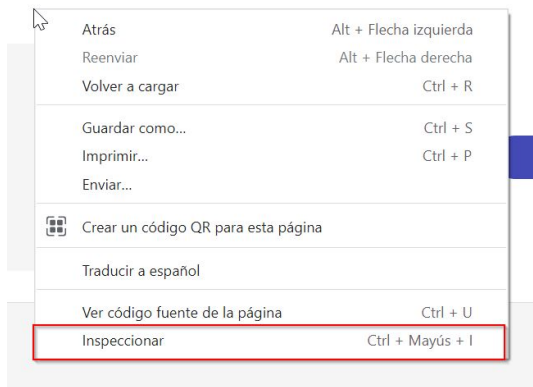
Debug desde la consola de Chrome

Se puede depurar código JavaScript directamente desde la consola de Chrome, siguiendo los pasos a continuación:

1

Acceder a la consola de Chrome:

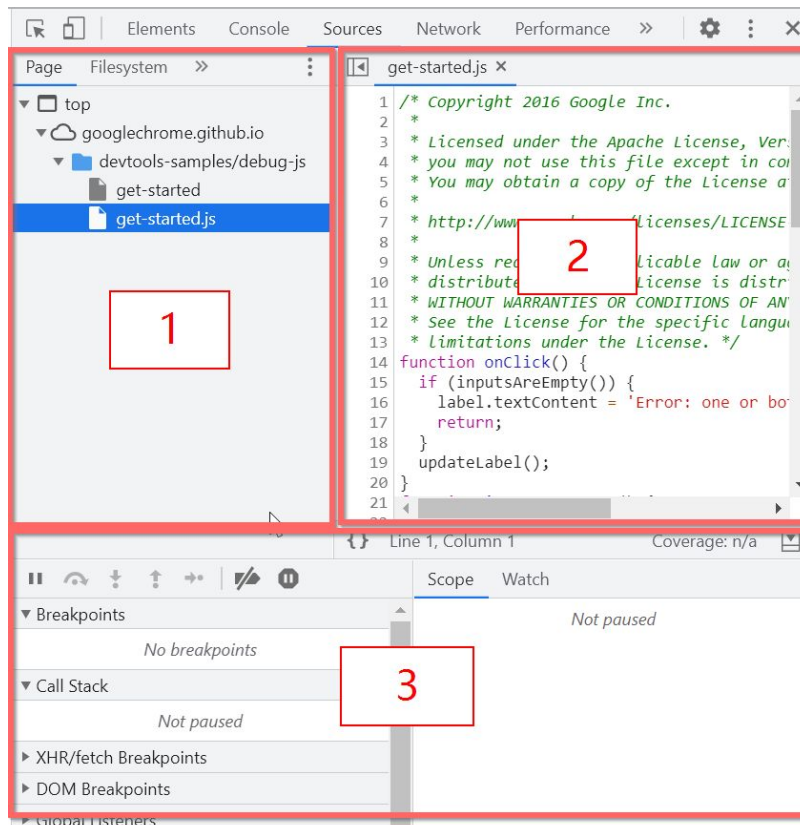
- haciendo clic derecho y seleccionando “Inspeccionar” o “Ctrl + Mayús + I”
- presionando la tecla F12(en algunas pcs)
- haciendo clic el menú lateral derecho, ir a “Más herramientas” y seleccionar “Herramientas para desarrolladores”



Debug desde la consola de Chrome(cont.)

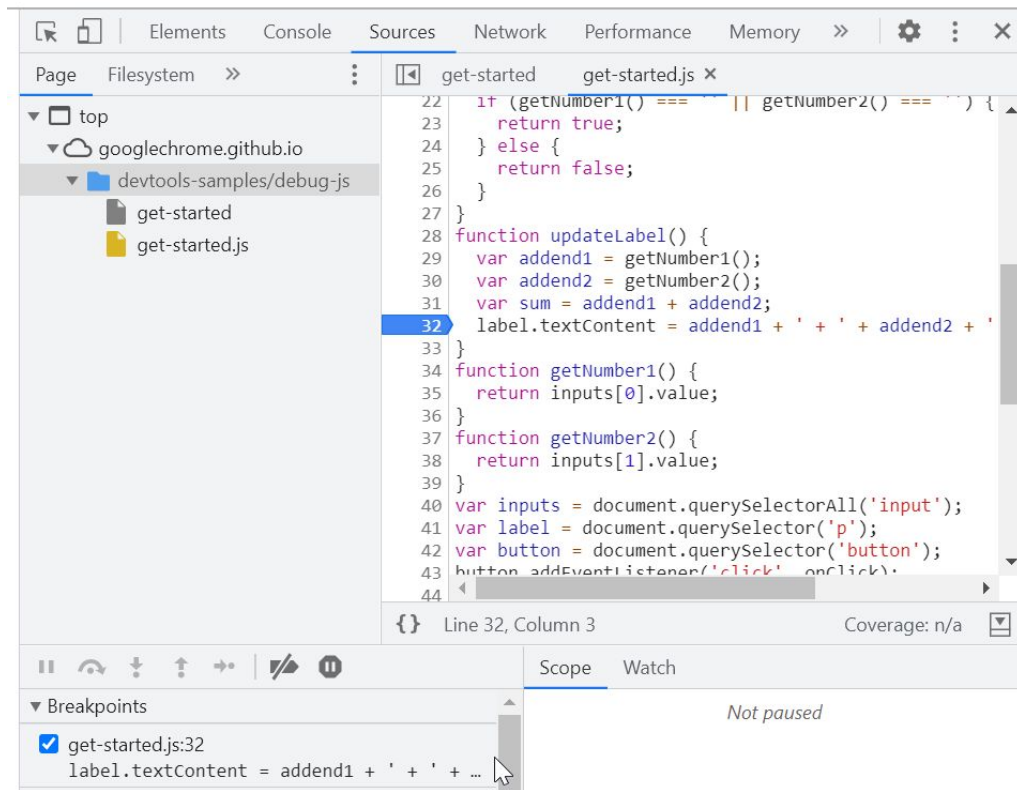
2 Ir a la pestaña "Sources", que cuenta con 3 partes:

1. Navegador de archivos
2. Editor de código
3. El depurador de Javascript



Debug desde la consola de Chrome(cont.)

- 3 Marcar el breakpoint en la línea de código correspondiente, haciendo clic en el número de la misma

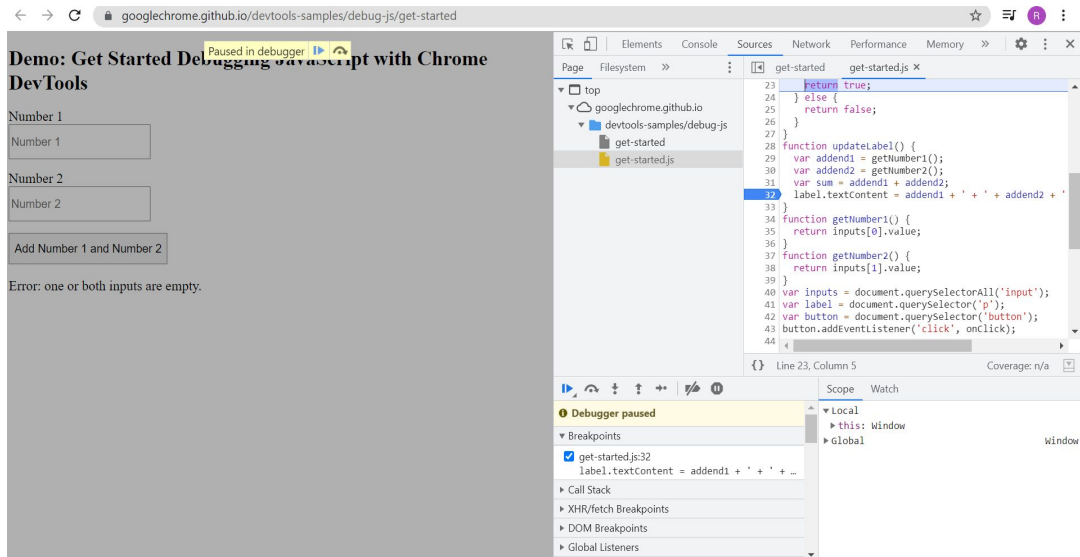


Debug desde la consola de Chrome(cont.)

4

Comenzando con alguna función se inicializa el modo debug, por ejemplo un click en un botón, luego se puede presionar F11 para recorrer línea por línea y F8 para recorrer de un breakpoint a otro.

También, se pueden utilizar los comandos de la propia interfaz que se verán a continuación:



Interfaz de debug

Cuando abrimos el menú de debug tanto en Chrome como en el Visual Studio nos despliega lo siguiente:

- **Continue/Resume**
- **Step over**
- **Step into**
- **Step out**
- **Restart**
- **Stop**

Chrome









Visual Studio



Veamos cada parte en más detalle...

Controles

Icono	Función
	Continue: Continúa con la ejecución del código hasta el siguiente breakpoint.
	Step over: Ejecuta la siguiente línea de código. Si es una función, la ejecuta y retorna el resultado.
	Step into: Ejecuta la siguiente línea de código, pero si es una función “entra” a ejecutarla línea por línea.
	Step out: Devuelve el debugger a la línea donde se llamó a la función.
	Restart: Reinicia el debugger.
	Stop: Frena el debug.

Ejemplo

The screenshot shows a web browser window with the address bar displaying `googlechrome.github.io/devtools-samples/debug-js/get-started`. The page content includes a title **Demo: Get Started Debugging JavaScript with Chrome DevTools**, two input fields labeled "Number 1" and "Number 2" both containing the value "1", a button labeled "Add Number 1 and Number 2", and a result display showing `1 + 1 = 11`.

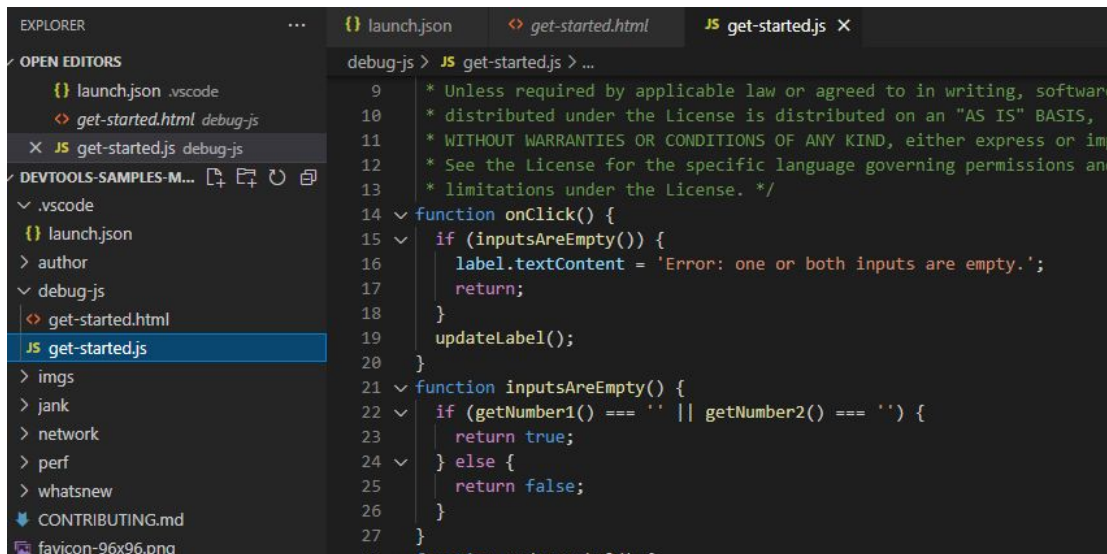
Chrome DevTools is open on the right side of the browser window. The top toolbar shows tabs for Elements, Console, Sources, Network, Performance, and Memory. The "Elements" tab is selected, showing a tree view of the document structure. Below the tree view, there are tabs for Styles, Computed, Layout, Event Listeners, DOM Breakpoints, Properties, and Accessibility. The "Styles" tab is selected, showing a filter bar with the text "Filter" and a dropdown menu with the value ":hov .cls".

4 | Debug en Visual Studio

Debug en Visual Studio

1

Dentro de la aplicación, Ir a la funcionalidad que se requiere depurar, desde el explorador de soluciones



Debug en Visual Studio(cont.)

- 2 Colocar un breakpoint en la línea de código requerida.

Breakpoint

Al hacer clic a la izquierda del número de línea, establecemos un breakpoint.

Contenido

El contenido de nuestro código.

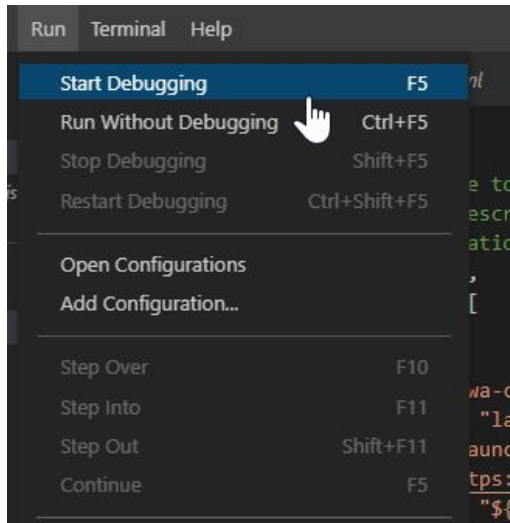


Número de línea

Nos indica el número de línea en el que está el código.

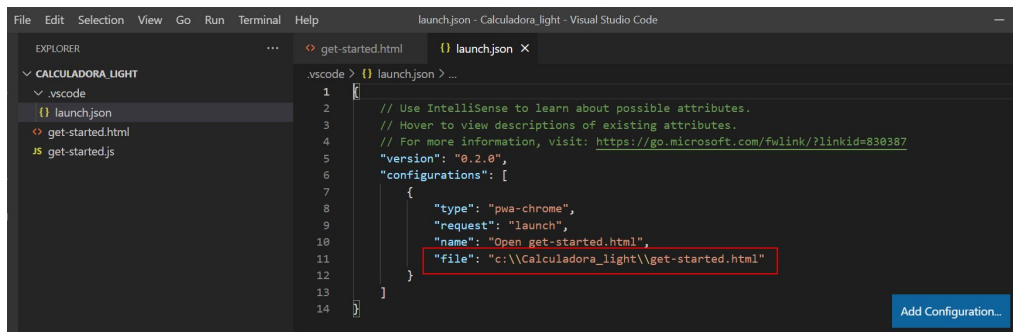
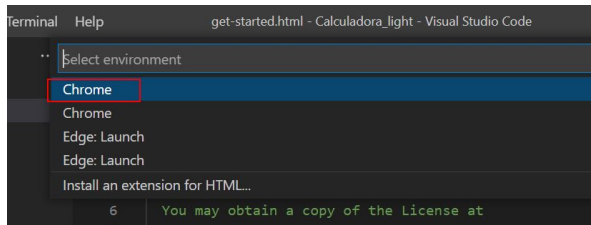
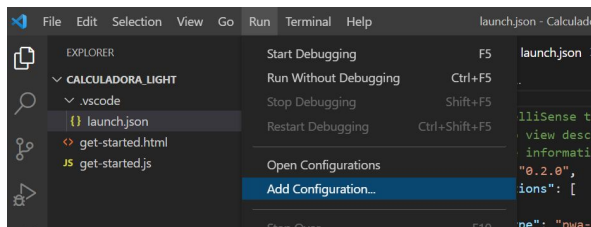
Debug en Visual Studio(cont.)

- 3 Iniciar el Debug, haciendo clic en F5 o ingresando en Ejecutar(Run) -> Start Debugging



Debug en Visual Studio(cont.)

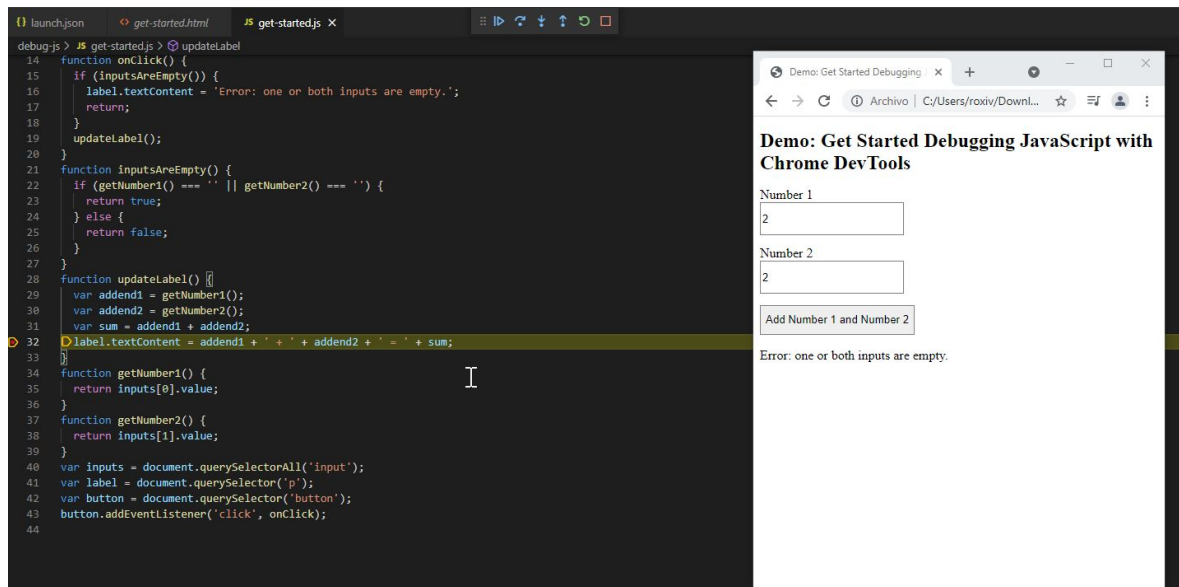
Si es la primera vez que se va a realizar debugging sobre la aplicación, se debe ingresar al menú **“Add configuration...”** y agregar la configuración de **Chrome** para ejecutar la aplicación. Esto crea el archivo **launch.json** en el cual se debe configurar en **“file”** la dirección del archivo .html que inicia la aplicación:



Debug en Visual Studio(cont.)

4

Debido a que se utiliza Chrome para realizar el debugging, es necesario realizar una acción en la aplicación



Control de variables

```
debug-js > JS get-started.js > updateLabel
14 function onClick() {
15   if (inputsAreEmpty()) {
16     label.textContent = 'Error: one or both inputs are empty.';
17     return;
18   }
19   updateLabel();
20 }
21 function inputsAreEmpty() {
22   if (getNumber1() === '' || getNumber2() === '') {
23     return true;
24   } else {
25     return false;
26   }
27 }
28 function updateLabel() {
29   var addend1 = getNumber1();
30   var addend2 = getNumber2();
31   var sum = addend1 + addend2;
32   label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;
33 }
34 function getNumber1() {
35   return inputs[0].value;
36 }
```

VARIABLES

- Local: updateLabel
 - addend1: '2'
 - addend2: '2'
 - sum: '22'
 - > this: Window
 - Return value: undefined
- Global

WATCH

Como podemos ver, en el breakpoint nuestra variable vale "2" y eso está reflejado en el menú de debug a la izquierda.

DigitalHouse>
Coding School