

# TEMA 4. Control de errores. Depuración

Desarrollo Web en Entorno Cliente.

Profesor: Juan José Gallego García

# Índice :

- Captura y lanzamiento de errores.
- Depuración.
- Bibliografía.

# Captura y lanzamientos de errores.

Podemos usar el conjunto de palabras claves **try**, **catch**, **throw** y **finally**, para probar un bloque de código en busca de errores y controlarlo.

- **try**, contiene el código a probar.
- **catch**, ejecuta el código en caso de darse alguna excepción o error.
- **throw**, puede generar excepciones o errores personalizados (opcional) .
- **finally**, ejecuta el código independientemente a que se den errores (opcional).

(Ver sintaxis y ejemplos a continuación)

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" >
  <script>
try {
  alert("Hola");
  noExisteFuncion();
  alert ("Adiós");
}
catch(err) {
  alert (err);
}
  </script>
</head>
<body>
</body>
</html>

```

← El siguiente ejemplo devuelve un error al no estar definida la función. El parámetro de error “**err**”, se define como un objeto con dos propiedades: **name** y **message**.

→

El siguiente ejemplo muestra cómo generar un error personalizado con **throw** y el uso de **finally**.

```

<script>
  var n = prompt("Introduzca un número : ");
  try {
    if(n == "") throw "está vacío";
    if(isNaN(n)) throw "no es un número";
    n = Number(n);
    alert ("Se ha introducido el número: "+ n);
  }
  catch(err) {
    alert ("El valor "+ err);
  }
  finally {
    alert ("Siempre se ejecuta");
  }
</script>

```

# Tipos de errores nativos

Tipo de error	Descripción	Ejemplo de lanzado
RangeError	Error de rangos (Ej. valores desorbitados)	<pre>var num = 1; num.toPrecision(500);</pre>
ReferenceError	Error en referencias (Ej. a variables inexistentes)	<pre>var x = 0; x = y + 1;</pre>
SyntaxError	Errores sintácticos (Ej. falta una comilla simple)	<pre>eval("alert('Hola')");</pre>
TypeError	Errores por tipo (Ej. conversiones no válidas)	<pre>var num = 1; num.toUpperCase();</pre>
URIError	Errores en direcciones	<pre>decodeURI("%%%");</pre>

# Depuración.

Como ya sabemos, al codificar un programa en JS podemos obtener errores de difícil detección. Para la búsqueda de estos errores existen varios métodos.

## Método `console.log ( )`

La mayoría de los navegadores admiten el envío de valores a la ventana de depuración del navegador. Esto permite comprobar si esos valores son los esperados. Sintaxis: **`console.log ( mensaje )`**

Ej:

```
var x = 10;  
console.log ( x );
```

Pulsamos F12 en el navegador y nos vamos a la pestaña “**Console**”. Aparece el mensaje enviado...

# A través del depurador del navegador.

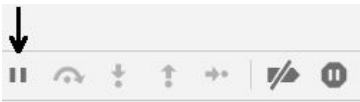
Por lo general los principales navegadores poseen una serie de herramientas para la depuración. También es posible usar el editor de código **Visual Studio Code** para lanzar el programa en modo depuración, por ejemplo, haciendo una llamada a las funcionalidades del Chrome.

En este manual vamos a tomar como referencia las herramientas de depuración del Chrome (PARA LAS ÚLTIMAS VERSIONES DEL CHROME Y M. FIREFOX SOLO FUNCIONA CON PUNTOS DE RUPTURAS ACTIVADOS).

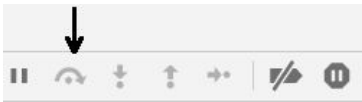
Para depurar un programa en JS con el Chrome podemos seguir los siguientes pasos:

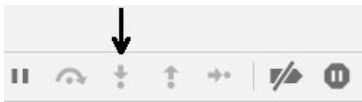
1. Abrimos o ejecutamos el archivo HTML en el navegador.
2. Pulsamos F12 para mostrar las herramientas de desarrollo y depuración.


3. Seleccionamos la pestaña “**Sources/Fuentes**”. Podemos ver el código de nuestro archivo.

4. Pulsamos  o F8 para entrar en modo pausa en la ejecución.


5. Refrescamos con F5 la página para ejecutar de nuevo, esta vez en modo pausa.

6. Avanzamos con  o con F10 para recorrer las instrucciones sin entrar dentro de las funciones.

7. Avanzamos con  o con F11 para recorrer las instrucciones una a una incluso dentro de cada función.

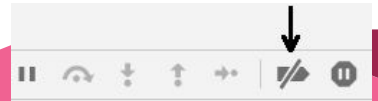
8. Si pulsamos  o Shift + F11 salimos de la actual función y se ejecuta la siguiente línea de código.



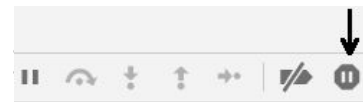
9. Si pulsamos  o F9, al igual que con F11 recorreremos cada instrucción incluso dentro de las funciones pero añadiendo un comportamiento especial en funciones asíncronas (recorre las líneas de código cronológicamente).
10. Para entrar de nuevo en modo normal (sin depuración), pulsaremos F8 estando en una línea de código de forma pausada.

Podemos crear o eliminar **puntos de interrupción** haciendo click en la línea de código correspondiente, o añadiendo la instrucción “**debugger ;**”, al ejecutar el código hará una pausa permitiéndonos analizar los valores de las variables de dos formas, una colocando el puntero del ratón encima de la variable, y otra, añadiendo la variable en el apartado “**watch**” del depurador.

El siguiente botón activa o desactiva los puntos de interrupción.



Para activar una pausa siempre que se de un error, con el botón



The image shows the Chrome DevTools interface with several annotations:

- Salida html**: Points to the top-left corner of the browser window.
- Salida consola**: Points to the console output area at the bottom, showing the number 30.
- Punto de interrupción**: Points to line 12 in the source code, which is highlighted in blue.
- Panel de avance**: Points to the top-right toolbar area containing navigation and execution controls.
- Alcance de las variables**: Points to the 'Scope' panel on the right, which shows the current execution context.
- Variables analizadas**: Points to the 'Watch' panel on the right, which displays the values of variables x (10) and y (20).

The source code in the center is as follows:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p id="demo"></p>
<script>
  var x = 10;
  var y = 20;
  console.log ( x + y );
  function f1() {
    document.getElementById("demo").innerHTML = x;
    document.getElementById("demo").innerHTML = x + 1;
  }
  function f2() {
    let z=0;
    document.getElementById("demo").innerHTML = x + 2;
    document.getElementById("demo").innerHTML = x + 3;
  }
  f1 ( );
  f2 ( );
</script>
</body>
</html>
```

# Bibliografía

- LibrosWeb
- W3Schools
- Developer Mozilla Docs