

# JavaScript desde cero

Módulo 1

# Errores

# Tipos de errores

En las próximas diapositivas, explicaremos los siguientes tipos de errores:

- Errores de sintaxis.
- Errores en tiempo de ejecución (*runtime*).
- Errores lógicos.

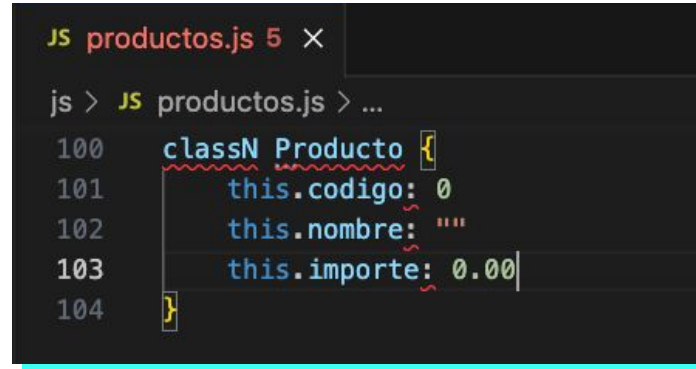


## Errores de sintaxis

Es normal que se cometan **errores de sintaxis**. Pueden ser de diferentes tipos. Lo importante es aprender a detectarlos y corregirlos en consecuencia. Los errores de sintaxis se comparan con un **error “ortográfico”**, propio del lenguaje natural de las personas.

*¿Qué consecuencias traen?*

**En lenguajes compilados: un error sintáctico no permitirá que el programa se compile.** Se debe corregir para compilar el programa y poder ejecutarlo.



```
JS productos.js 5 X
js > JS productos.js > ...
100  classN Producto {
101      this.codigo: 0
102      this.nombre: ""
103      this.importe: 0.00|
104  }
```

**Por ejemplo:** los editores de código actuales, pueden detectar errores en la sintaxis e indicarnos los mismos en el momento en el cual escribimos la lógica del programa.

## Errores en tiempo de ejecución

Un programa puede estar perfectamente **bien escrito**, libre de errores de sintaxis, pero puede generar **errores durante su ejecución**.

Estos errores suceden en **“tiempo de ejecución”** (*runtime*: intervalo de tiempo que va desde que el programa inicia hasta que finaliza).



Son producidos por **acciones / operaciones imposibles de realizar** (incompatibilidad entre tipos de dato y operadores u operaciones sin solución).

### Por ejemplo:

- Una división por cero.
- Cálculo de la raíz cuadrada de un número negativo.
- Bucles infinitos.
- Que el programa intente hacer un cálculo aritmético con valores de tipo `string`.

## Errores lógicos

Son aquellos relacionados con **acciones inesperadas que comete el programa**:

Por ejemplo:

- Que el programa realice una suma aritmética cuando en realidad debió haber *restado*.
- Que el programa borre datos de la aplicación en lugar de agregar datos nuevos.
- Que el programa no arroje mensajes en pantalla cuando debió haberlo hecho.

**Estos errores son los peores** con los que nos podemos encontrar ya que **requerirá una revisión en la lógica de alguna funcionalidad en particular o bien una revisión de toda la aplicación**.

Muchas veces, están relacionados con el **pseudo-código o los diagramas de flujo** que se armaron **previamente al código real** del programa.



# Características de los errores

Los errores tienen **dos formas de manejarse**:

## El error técnico

- Sirve para poder **analizar qué sucede**, y luego buscar una **solución**.

## Error de cara al usuario

- Debe ser un **error genérico, coloquial**, y que le **transmita de la forma más clara posible** que la aplicación no se comporta de la forma esperada.
- Si este último mensaje puede guiar al usuario a buscar una solución, mejor. Si no, deberá **indicarle cómo proceder** para que el mismo no ocurra nuevamente.

Como programadores, tenemos que probar nuestra aplicación y entender que **nunca** **lograremos tener un control total de posibles errores.**

Por más que la probemos decenas, cientos o miles de veces, debemos **siempre buscar la forma de manejar cualquier posible error.**



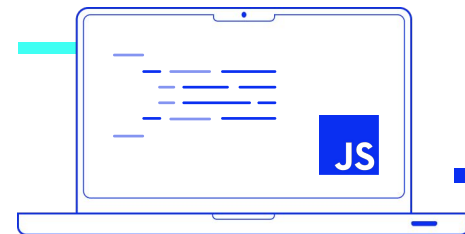


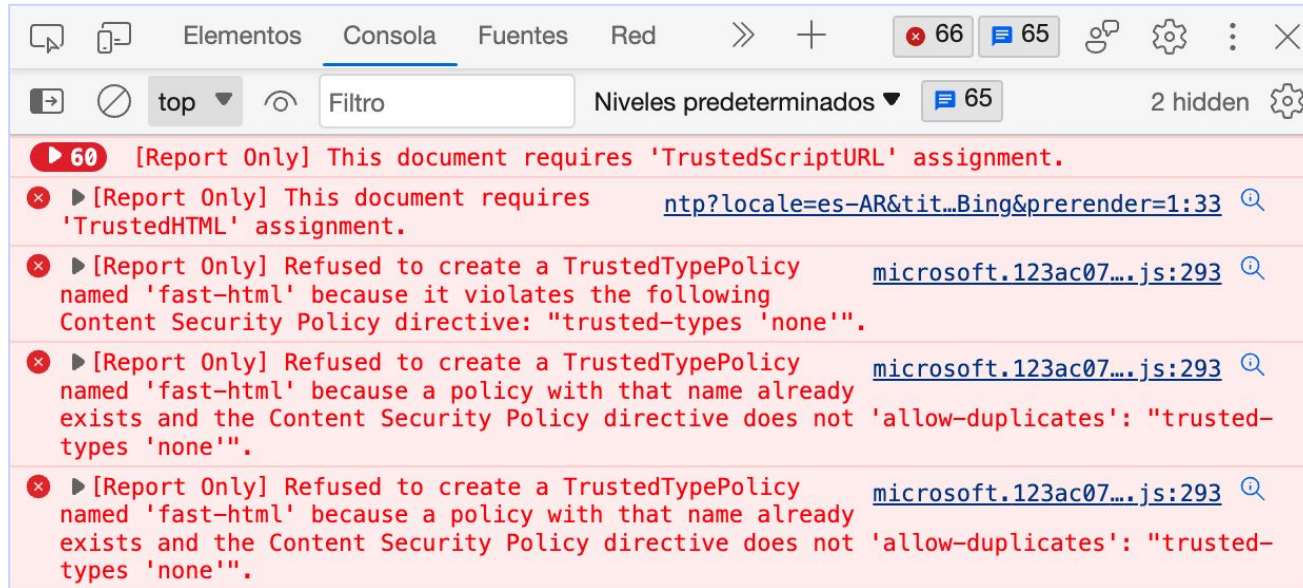
# La consola JS

En el desarrollo de aplicaciones web, cuando una aplicación construida con JS falla de manera inesperada, **los errores que acontecen se representan en la Consola JS.**

Todos ellos suelen estar descriptos con **un mensaje, donde se explica el error en particular.**

Además, casi siempre encontramos una **referencia del archivo JS** donde se produce el error, y también el **número de línea** donde está la sentencia de código que lo provoca.





The image shows a browser's developer console with the 'Consola' tab selected. The console displays several security warnings related to Content Security Policy (CSP) directives. The first warning is a report-only message about 'TrustedScriptURL' assignment. The following three warnings are error messages (indicated by a red 'x') about 'TrustedHTML' assignment and 'TrustedTypePolicy' creation failures. The failures are due to the 'trusted-types' directive being set to 'none', which prevents the creation of policies with names like 'fast-html'.

Elementos Consola Fuentes Red >> + 66 65

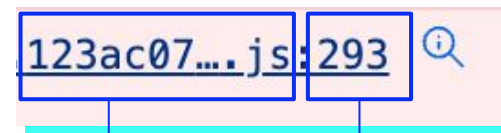
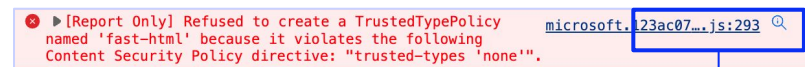
top Filtro Niveles predeterminados 65 2 hidden

- ▶ 60 [Report Only] This document requires 'TrustedScriptURL' assignment.
- ✘ ▶ [Report Only] This document requires 'TrustedHTML' assignment. [ntp?locale=es-AR&tit...Bing&prerender=1:33](#)
- ✘ ▶ [Report Only] Refused to create a TrustedTypePolicy named 'fast-html' because it violates the following Content Security Policy directive: "trusted-types 'none'". [microsoft.123ac07...js:293](#)
- ✘ ▶ [Report Only] Refused to create a TrustedTypePolicy named 'fast-html' because a policy with that name already exists and the Content Security Policy directive does not 'allow-duplicates': "trusted-types 'none'". [microsoft.123ac07...js:293](#)
- ✘ ▶ [Report Only] Refused to create a TrustedTypePolicy named 'fast-html' because a policy with that name already exists and the Content Security Policy directive does not 'allow-duplicates': "trusted-types 'none'". [microsoft.123ac07...js:293](#)

## Ver y analizar errores en JS

Esta guía es el **punto de partida, para ir a ese archivo y línea**, y así poder analizar qué problema ocurre.

Luego de su análisis, podremos decidir cuál es la mejor forma de resolver el error y controlarlo cuando ocurra o anticiparnos para que no suceda.



**Nombre del archivo**  
JavaScript

**Línea** donde se produce el error

# Controlar un error: bloque *try-catch*

JavaScript brinda muchas formas de controlar errores. La más apropiada es mediante el uso de un bloque conocido como **try-catch**.

Esta estructura se utiliza para **manejar excepciones o errores** que pueden ocurrir durante la ejecución de un bloque de código.

Permite proteger ciertas partes del código que pueden generar errores y **capturar esas excepciones para realizar una acción específica en caso de que ocurran**.

## Funcionamiento

Cuando se ejecuta un bloque de código dentro del **try**, si ocurre una excepción, en lugar de detener todo el programa, se captura el error y se ejecuta el bloque **catch**.

Dentro del bloque **catch**, se puede acceder al objeto error, que contiene información sobre la excepción que se produjo, como el tipo de error y el mensaje asociado.



## Sintaxis

La sintaxis básica del bloque **try-catch**, es la siguiente:

```
try {  
    // Código que puede generar una excepción  
} catch (error) {  
    // Acciones a realizar en caso de que ocurra una  
    excepción  
}
```



## Ejemplo

Veamos un ejemplo práctico de cómo implementamos el bloque try-catch dentro de nuestro código JavaScript.

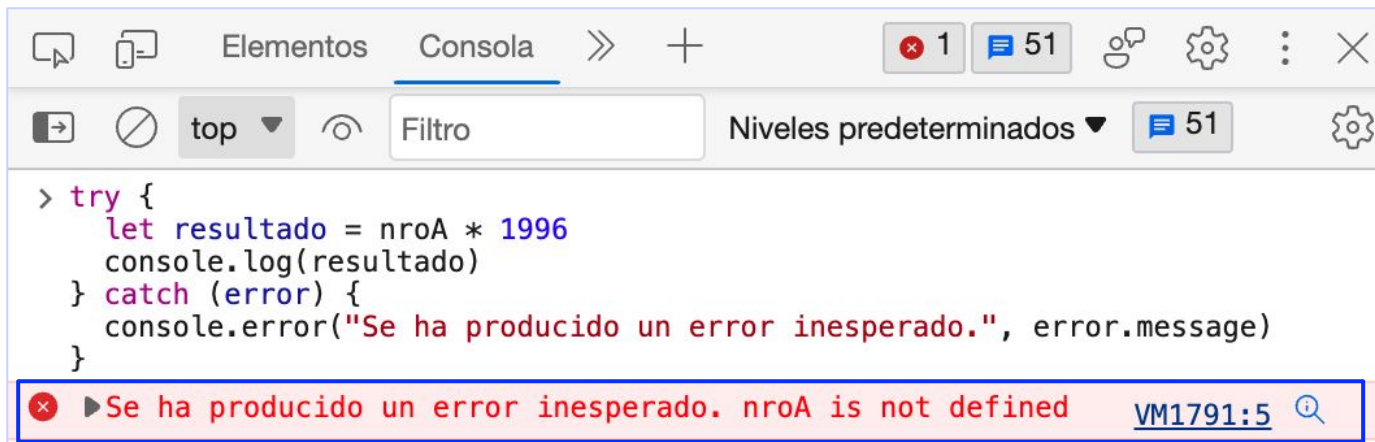
Dentro del bloque **try** intentamos multiplicar la variable **nroA** por **1996**.

La variable en cuestión nunca fue declarada dentro de nuestra aplicación. Como la misma no existe, **la aplicación arrojará un error. Este error será capturado por el bloque catch y mostrado a continuación en la consola JS.**

```
try {
  let resultado = nroA * 1996
  console.log(resultado)
} catch (error) {
  console.error("Se produjo un error inesperado.", error.message)
}
```

Al probar el bloque de código en la Consola JS, se verifica que, efectivamente, causa un error en la aplicación **por no haber declarado previamente la variable nroA**.

El bloque **try-catch** controla y maneja este error de forma efectiva, como se muestra en la imagen siguiente:



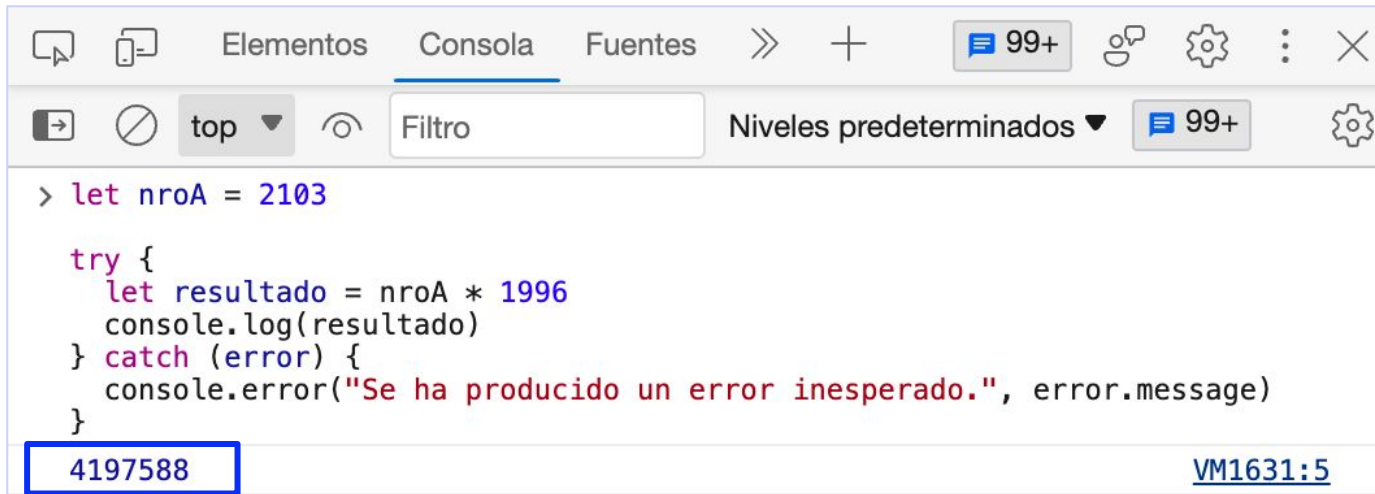
The image shows a browser's developer console with the 'Consola' tab selected. The console displays a JavaScript code snippet that uses a try-catch block to handle an error. The code is as follows:

```
> try {  
  let resultado = nroA * 1996  
  console.log(resultado)  
} catch (error) {  
  console.error("Se ha producido un error inesperado.", error.message)  
}
```

Below the code, the console shows an error message: "Se ha producido un error inesperado. nroA is not defined". The error message is highlighted with a red box. The error details include a red 'x' icon, a play button, the message text, and the location "VM1791:5" with a magnifying glass icon.

Si **corregimos nuestro algoritmo** asignándole un valor numérico, al ejecutar nuestro código veremos que éste no arrojará ningún error en la consola JS.

Se ejecutará solamente el bloque `try`, y se **visualizará el resultado de la operación aritmética**:



```
> let nroA = 2103

  try {
    let resultado = nroA * 1996
    console.log(resultado)
  } catch (error) {
    console.error("Se ha producido un error inesperado.", error.message)
  }

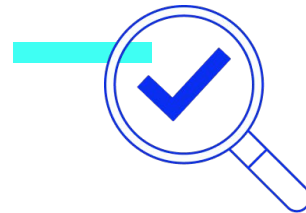
4197588
```

VM1631:5



# Revisión

- Repasar los conceptos básicos de un **error en JavaScript**.
- Investigar sobre los diferentes **tipos de errores**.
- Generar un error voluntario para indagar desde la consola.
- Implementar los bloques **try-catch** en nuestro código.
- Aplicar todas las propiedades en el **Proyecto integrador**.
- Realiza las preguntas necesarias antes de continuar.



**¡Sigamos  
trabajando!**