9.1 CONTROL DE ERRORES

9.1.1 INTRODUCCIÓN AL CONTROL DE ERRORES

Sin duda una de las tareas más importantes, a la hora de programar aplicaciones, es la de solucionar los errores que van apareciendo. Todos los profesionales del mundo del desarrollo han pasado horas y horas, al menos alguna vez, tratando de solventar un error que habían detectado en el programa. Lo peor, es que hay veces que los errores se detectan mucho después porque ocurren circunstancias en el uso de la aplicación que no se habían tenido en cuenta.

La habilidad de encontrar errores se va consiguiendo a través de los años de experiencia, pero hay que conocer los aspectos y las técnicas fundamentales desde el primer momento. Los errores que ocurren en un programa pueden ser:

- **Errores al escribir código por parte del programador.** Son **errores de sintaxis,** errores por cosas como: expresiones incorrectas al escribir el código, cierres de llaves olvidados, palabras clave mal escritas, etc. Son los más fáciles de detectar porque cuando se inter•preta el código, se nos indica el error. En el caso de que el código se esté creando en un entorno de trabajo avanzado, hay errores que aparecen marcados en el mismo instante en el que hemos escrito el código. En el caso de las aplicaciones web, el error aparece también marcado en la consola del panel de depuración del navegador.

  En realidad hay dos tipos:

  - **Errores detectables en tiempo de escritura.** Son fallos de sintaxis evidentes que la mayoría de entornos de desarrollo (incluido **Visual Studio Code)** pueden marcar antes de probar el código. Muchas veces estos entornos lo que hacen es subrayar en rojo el código erróneo.
  - **Errores de ejecución.** Son errores que solo se pueden detectar cuando el código se intenta ejecutar para probar la aplicación. Un ejemplo de error de este tipo es cuando en el código se invoca a una función que aún no ha sido definida. Solamente cuando tratamos de ejecutar el código se puede detectar que esa función no existe.

- Hay errores por mala lógica al desarrollar la aplicación. Son **errores lógicos,** en los que la sintaxis es correcta, pero el programa no funciona como debería. Ninguna herramienta nos avisa automáticamente del error, aunque sí hay herramientas especiales que facilitan su detección, es el desarrollador el que detecta que la aplicación no funciona como debería. Puede ser también, que el error lo detecten los usuarios y se lo comuniquen a los desarrolladores.

- Hay errores por causas externas. Son **errores del sistema,** circunstancias que provocan el error pero que están fuera del control del programador: fallo en la conexión de red, caída de un servicio que estábamos utilizando, etc. No podemos controlar estos fallos la mayoría de las veces, pero al menos sí podemos matizar el daño que causan a nuestra aplicación.

- **Errores de usuario.** Son los provocados por acciones inesperadas que realiza el usuario y que causan un error en tiempo de ejecución. Por ejemplo, pedir al usuario un número y recibir un texto. En realidad, son errores lógicos que ocurren por no prever estas situaciones.

9.1.2 CREAR Y LANZAR ERRORES PROPIOS

Los errores se crean por parte del intérprete de JavaScript cuando ocurren. Pero podemos crear nuestros propios errores:

```
let miError=new Error("Se esperaba un número");
throw miError;
```

Crear un objeto de error no provoca ningún error. Lanzar el error en sí y provocar una excepción, se hace con la palabra clave **throw.**

9.1.3 GESTIONAR LAS EXCEPCIONES. BLOQUE TRY...CATCH

JavaScript aporta una estructura llamada **try..catch.** Esta estructura trabaja con esta sintaxis:

```
try{
   …
   código que puede provocar un error
   …
}catch(objetoError){
   …
   código que se ejecuta si hay error
   …
}
```

En el bloque encabezado por la palabra **try** se coloca el código que puede provocar un error. Si ese error se produce, el flujo del programa pasa al apartado catch, dejando el resto de líneas del try posteriores a la que produce el error, sin ejecutar.

Veamos este sencillo ejemplo:

```
try{
    console.log(e);
    console.log(“aquí”);
}
catch(error){
        let e=1;
        console.log(e);
```

Si el archivo solo tiene este código, se va a intentar escribir por consola el valor de una variable llamada ***e*** que no existe porque no se ha declarado. Con lo cual el código **console.log(e)** produciría un error. Al estar en un bloque **try,** se crea el objeto de error y se envía dicho objeto al apartado **catch.** Las líneas dentro del catch pasan a ejecutarse: se declara la variable **e** con valor **1** y se escribe por pantalla. Veremos simplemente, al ejecutar el código, que se muestra el número 1 por consola.

Todo cambia si ***e*** se declara antes de la instrucción ***try:***

```
let e=12;
try{
   console.log(e);
   console.log("aquí");
}
catch(error){
    let e=1;
    console.log(e);
}
```

Se muestra por pantalla:

```
12
aquí
```

No se muestra el código del catch, no hay error.

Hay posibilidad de añadir un bloque **finally** cuyo contenido se muestra tanto si se produce la excepción, como si no:

```
try{
      consolé.log(e);
      consolé.log("aquí");
}
catch(error){
        let e=l;
        consolé.log(e);
}
finally{
      consolé.log("Yo salgo siempre");
}
```

e no está definido. El código dentro del try genera un error. y ese error pasa a catch

el parámetro error es la variable que recoge el error capturado.
la variable e, la creas en catch y es independiente a error.
Si no lo manejas atrapas el error pero no haces nada.
Para manejarlo en catch(error){ console.error("Ha ocurrido un erro";error.message);

Muestra:

```
1
Yo salgo siempre
```

```
function conversor(array) {
  let numero;

  try {
    numero = parseInt(array);

    if (isNaN(numero)) {
      throw new Error('No es un número');
    }

    if (numero == null || numero == '') {
      throw new Error('La cadena está vacía o es nula');
    }
  } catch (e) {
    console.log(e.message);
  }

  return numero;
}
```