Bienvenides a fundamentos 6!

Manejo de errores

Hoy vamos a dar:

- Que es un error
- Errores famosos
- Como disparar manualmente un error
- Como interceptar y manejar errores (Para evitar que nuestro programa explote)

Primero, que es un error?

Son situaciones anomalas que pueden suceder a lo largo de nuestro codigo, como acceder a una propiedad de un objeto que no existe, escribir mal una sintaxis, o usar variables que jamas declaramos.

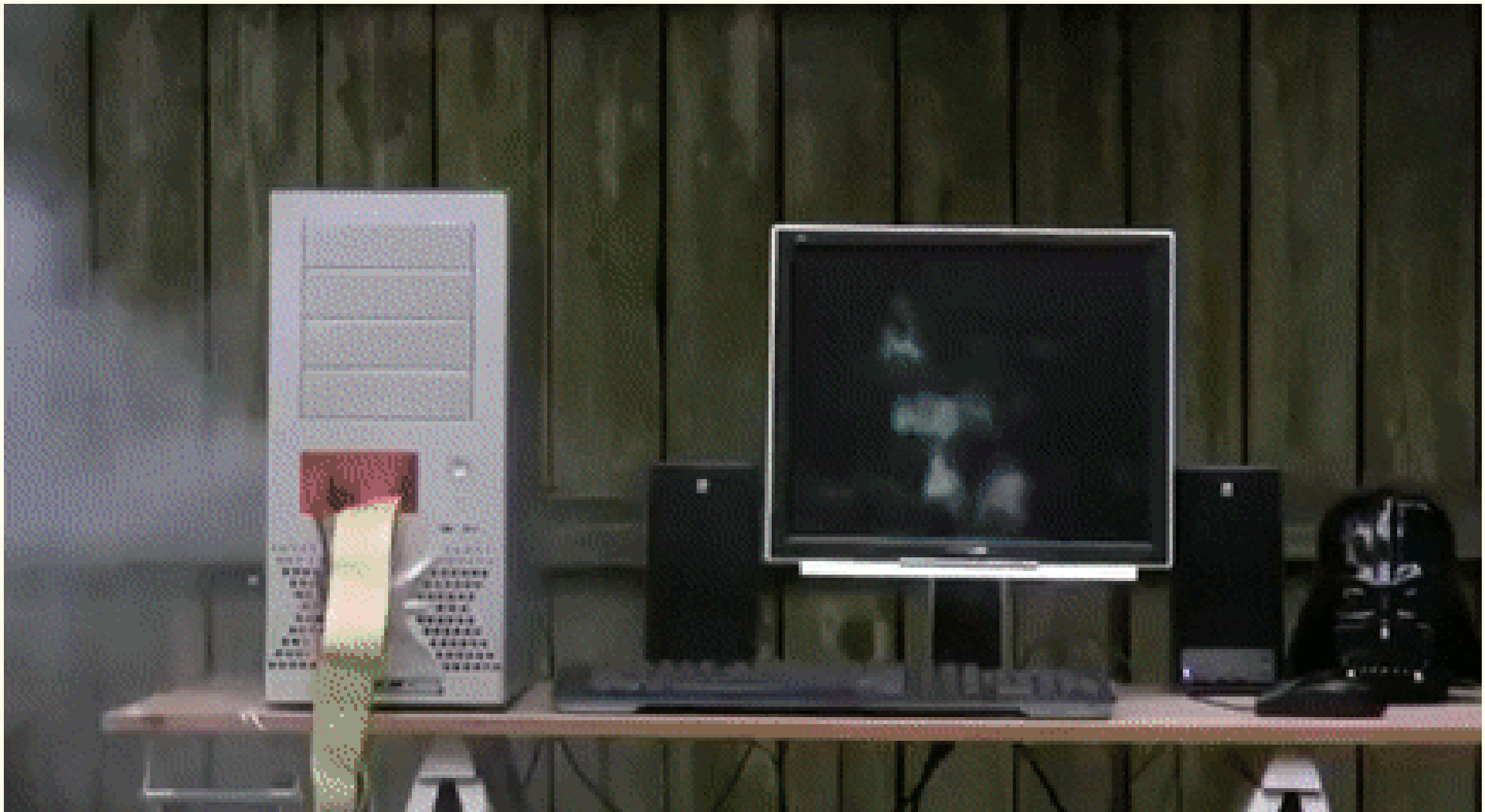
Barbaro, pero para que nos sirve esto?

Como programadores debemos tener en cuenta que algunas cosas pueden ir mal, y prepararnos para esas situaciones.

Como? eso lo vemos mas adelante

Antes de preparanos, como se ven los errores?

Veamos 3 errores famosos



TypeError

El mas comun de todos, ocurre cuando un valor no es del tipo esperado. Suele pasar cuando llamamos a una funcion que en realidad no es una funcion.

```
const variableA = 1;
const resultado = variableA();
```

SyntaxError

Sucede cuando el motor de JavaScript encuentra un error de sintaxis Si un archivo contiene un SyntaxError, no se va a ejecutar nada del codigo del archivo.

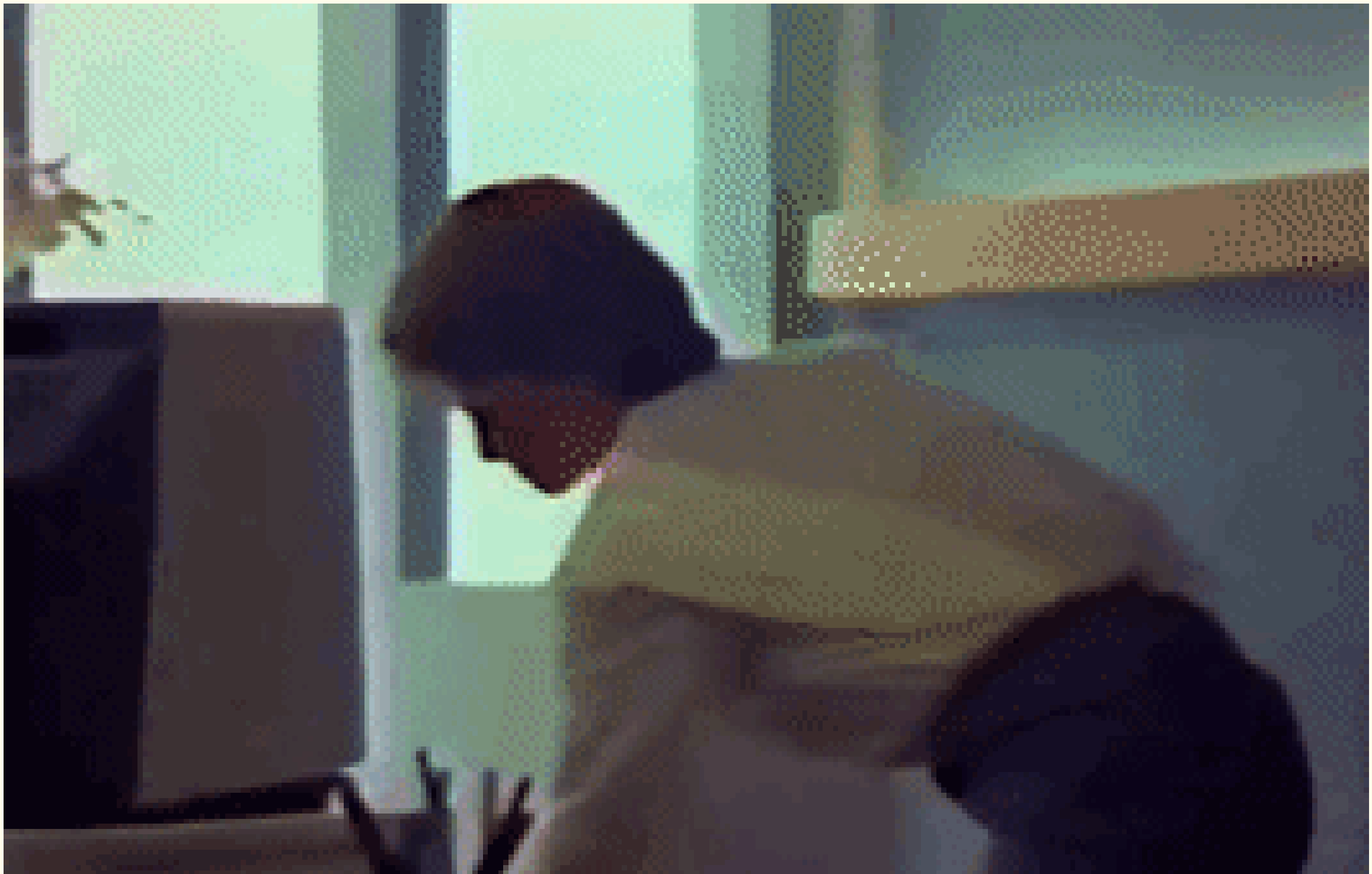
```
console.log('hello')
funcionMalCerrada(
```

ReferenceError

Sucede cuando nos referimos a variables que no existen.

```
console.log(variableInventadaQueNoExiste)
```

Ahora veamos como matar nuestra computadora



Error definido por el usuario

Podemos disparar errores de manera manual usando un objeto Error, que dispara un error con un mensaje personalizado

```
throw new Error('Este es mi error personalizado')
```

Manejar errores y no explotar en el intento

Como dijimos antes, como programadores tenemos que saber como evitar que los errores ocurran y rompan el programa. Para eso vamos a usar unas nuevas herramientas:

- try `define un ambiente de codigo seguro contra instrucciones que pueden disparar errores`
- catch (error) `define un ambiente de codigo que se ejecutara si se atrapo un error en el ambiente try`
- finally `define un ambiente de codigo que se ejecutara siempre`

try - catch - finally

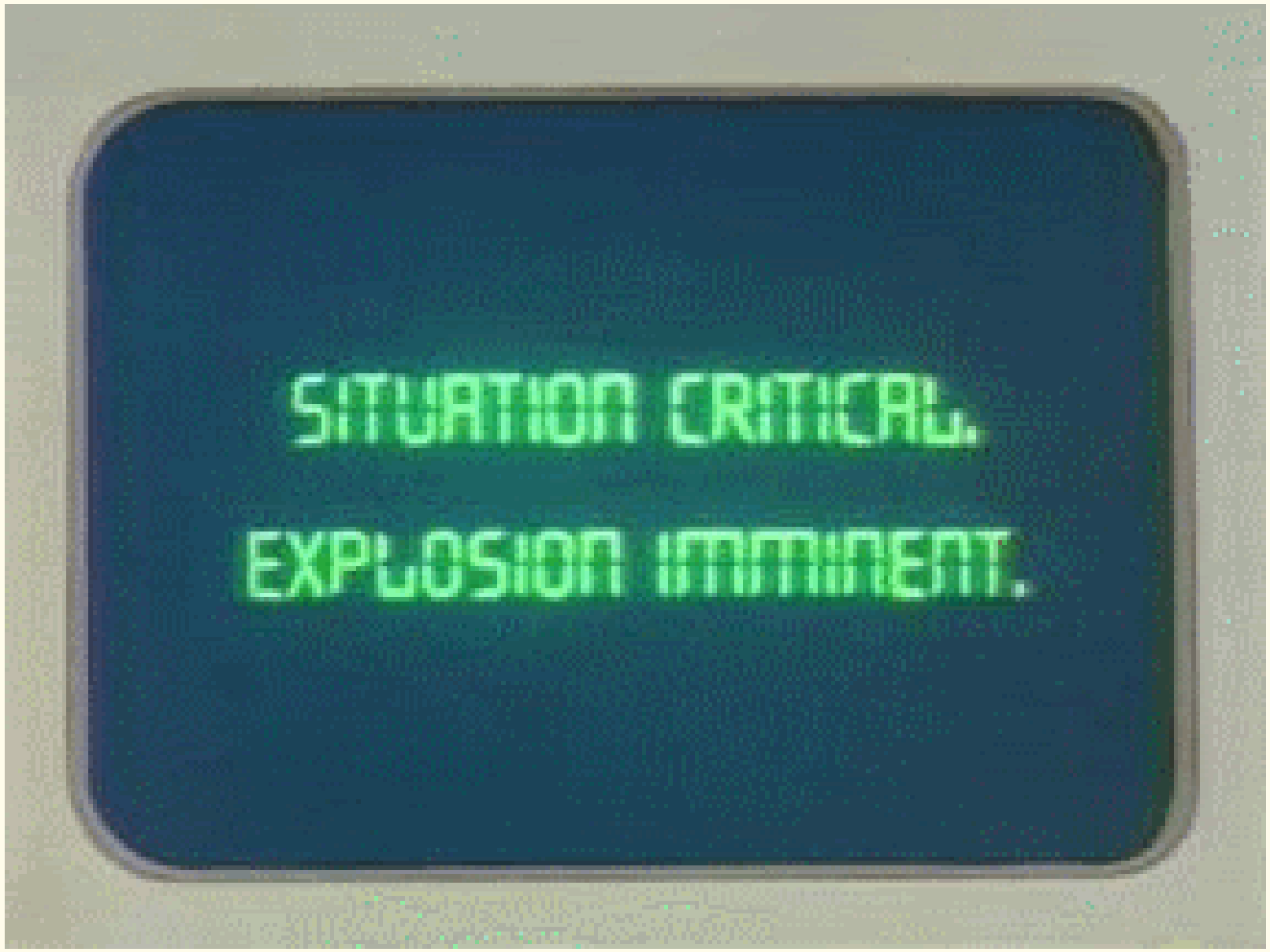
```
try {
    // zona segura
    throw new Error("Error personalizado")
} catch (error) {
    // Se ejecutara si se disparo un error en el try
    console.log(error);
} finally {
    console.log("Esto siempre se va a imprimir, no importa si hubo un error o no")
}
// Se imprime en consola:
// Error: Error personalizado
//  at Object.<anonymous>
//  at Module._compile....
// Esto siempre se va a imprimir, no importa si hubo un error o no
```

Ahora que ya sabemos como capturar errores y como dispararlos

Ejercitemos

- Crear una funcion 'randomError' que tenga 50% de posibilidades de disparar un error con mensaje "Error catastrofico"
- En caso de no disparar un error, la funcion debe devolver un mensaje "Todo ok"
- Capturar el error e informar 'Error capturado'

Primero: creemos la funcion 'randomError'



Resultado

```
function randomError() {
    if (Math.random() < 0.5) {
        // Esto se ejecutara el 50% de las veces
        throw new Error("Error catastrofico");
    }
    return "Todo ok"
}

try {
    // ambiente seguro
    let resultado = randomError();
    console.log(resultado)
} catch (error) {
    console.log(error.message);
}
```

Ejercitacion

1) Funcion de errores

Crear una funcion que recibe tres parametros `chance`, `mensaje` y `resultado`.

- `chance`: es un numero que indica el porcentaje que tiene esa funcion de arrojar un error. si `chance` = 20 entonces la funcion tiene 20% de posibilidades de disparar un error
- `mensaje`: el mensaje que tiene que tener el error que puede disparar la funcion
- `resultado`: es cualquier cosa, un numero, un texto, un objeto, etc. es lo que retorna la funcion si no arrojo un error

2) Simulador de descarga de informacion

Cuando queremos descargar informacion de internet, varias cosas pueden ir mal: desconexion de internet, corte de luz, perdida de informacion por culpa de seres desconocidos. Crear una funcion llamada `pedirInformacion` que no recibe parametros y que devuelve el siguiente objeto que corresponde a los datos de un usuario: `let persona = { id: 19310, nombre: "Bautista", apellido: "Di Santo" }`

La funcion `pedirInformacion` tiene:

- 25% de posibilidades de disparar un error por desconexion de internet, el nombre del error es 'NetworkError'
- 18% de posibilidades de disparar un error por error del servidor, el nombre del error es 'InternalError'
- 10% de posibilidades de disparar un error por una anomalia, el nombre del error es 'AlienError'

Cuando se llame a la funcion, informar:

- Si hubo un error, informar segun el mensaje del error:
  - NetworkError => informar "Hubo un problema en la conexion de internet"
  - InternalError => informar "Hubo un error interno en el el servidor"
  - AlienError => informar "Una anomalia intercepto la informacion"
- Si no hubo un error, informar el objeto `persona`

3) Descarga de informacion prohibida

Crear una funcion llamada `pedirUsuario` que retorna un usuario al azar de la siguiente lista:

```
let lista = [{
    id: 19310,
    nombre: "Bautista",
},{
    id: 90010,
    nombre: "Ema",
},{
    id: 00519,
    nombre: "Lucas",
},{
    id: 00000,
    nombre: "Meison",
}
]
```

La funcion `pedirUsuario` disparar un error llamado 'ForbiddenInformation' si el usuario elegido es el del `id` = 00000

Llamar a la funcion `pedirUsuario` dentro de la funcion `pedirInformacion` para cambiar el retorno de la funcion `pedirInformacion`.

Considerar que la funcion `pedirInformacion` ahora tiene que capturar un error, cuando la capture debe disparar un error 'NetworkError'.

Ejemplo:

```
function pedirUsuario(){
    // puede disparar un error, devuelve un usuario
}

function pedirInformacion(){
    let usuario = pedirUsuario(); // considerar 'pedirUsuario' p
    // el resto de la funcion
    return usuario;
}
```