

# Don't Be Afraid of the Javascript Stack Trace

...

Jennifer Bland

# Jennifer Bland

Sr. Software Engineer | Google  
Developers Expert

jenniferbland.com  
codeprep.io



Web Technologies  
GDE

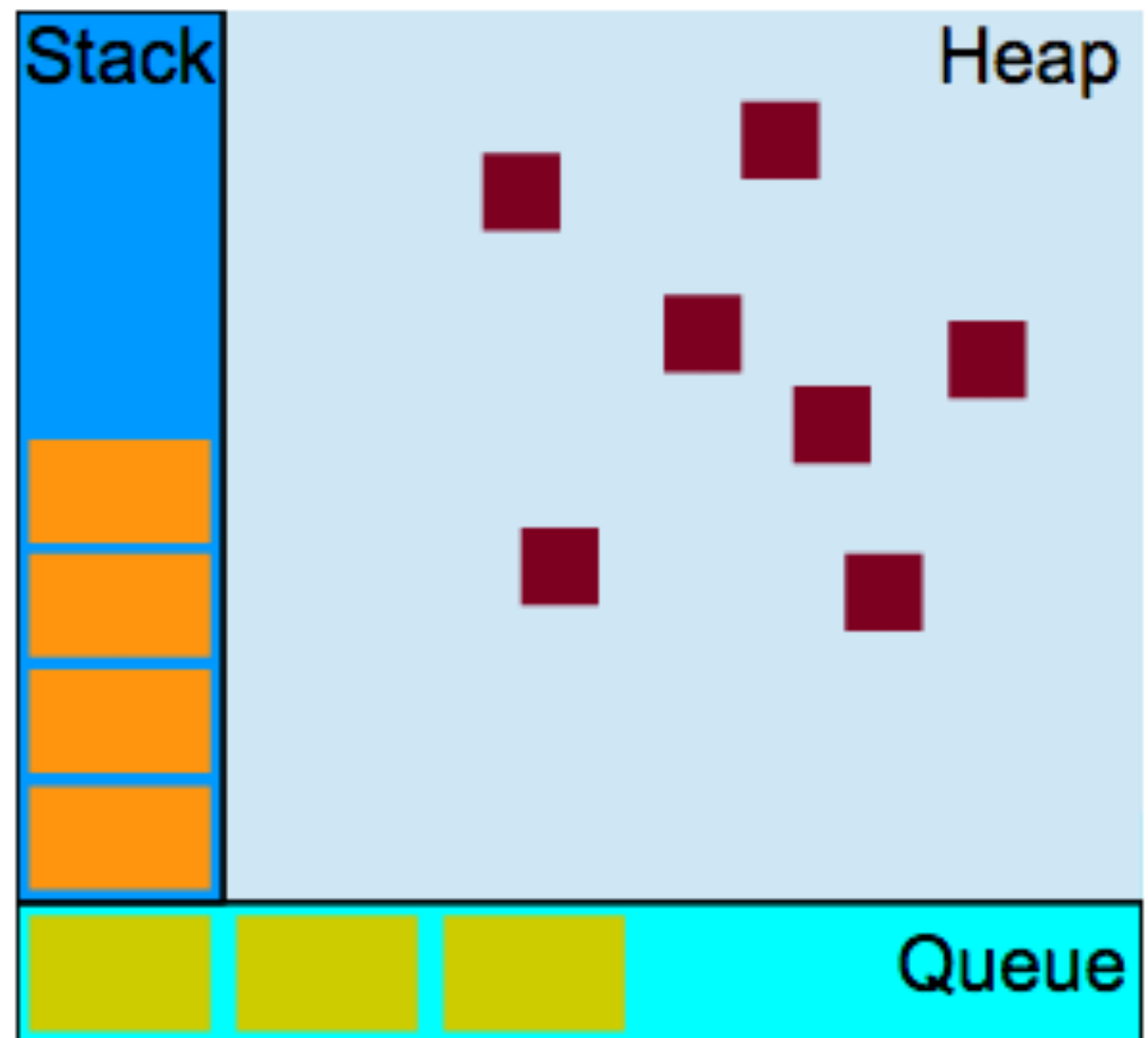


# 3 Truths 1 Lie

- I saw the only Gold Medal won by Great Britain
- I saw the only medal won by Mongolia
- I was an Olympic Torch Runner
- I attended the Opening Ceremony

# Background Terminology

- Call Stack
- Heap
- Queue
- Concurrency Model



# Call Stack

```
1  function foo(b) {  
2    var a = 5;  
3    return a * b + 10;  
4  }  
5  
6  function bar(x) {  
7    var y = 3;  
8    return foo(x * y);  
9  }  
10  
11 console.log(bar(6));
```



## Call Stack

console.log(bar(6))

main()

Output:-

# Heap

- mostly unstructured region of memory
- memory allocation to variables and objects happen here
- finite in size

# Queue

- list of messages to be processed and the associated callback functions to execute
- messages executed by call stack
- message processing ends with stack is empty

# What is the Stack Trace?

- what functions were called
- in what order
- from which line and file
- with what arguments



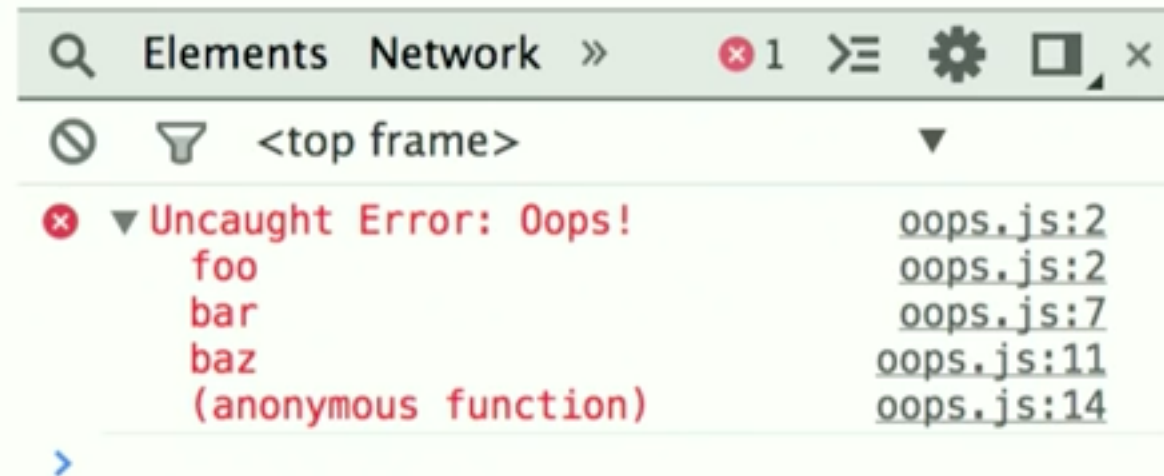
# Stack Trace Demo

```
function foo() {  
  throw new Error('Oops!');  
}
```

```
function bar() {  
  foo();  
}
```

```
function baz() {  
  bar();  
}
```

```
baz();
```



The screenshot shows a web browser's developer console with the 'Elements' and 'Network' tabs selected. A red 'x' icon indicates an error. The error message is 'Uncaught Error: Oops!'. The stack trace shows the following frames:


Frame	File	Line
foo	oops.js	2
bar	oops.js	7
baz	oops.js	11
(anonymous function)	oops.js	14

# 5 Levels of Logging

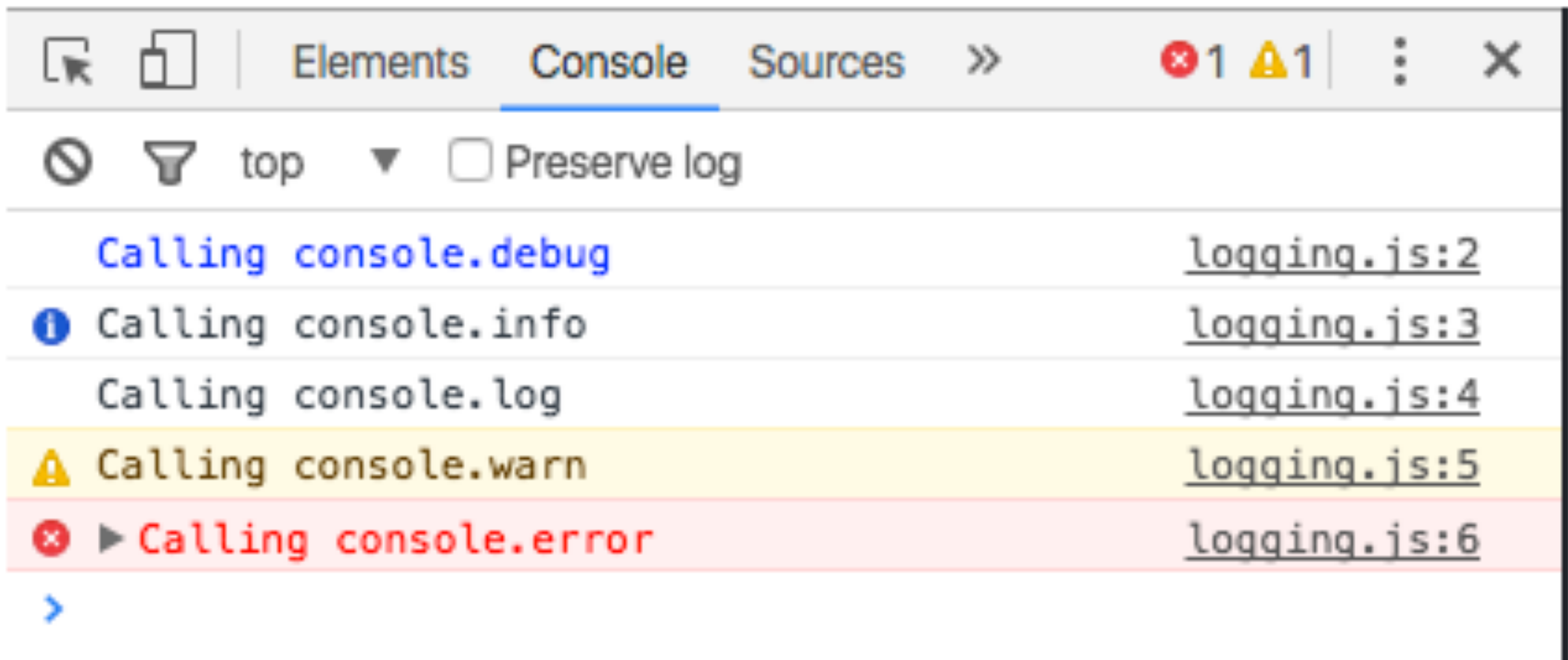
- debug
- info
- log
- warn
- error

```
function log_them() {  
    console.debug("Calling console.debug");  
    console.info("Calling console.info");  
    console.log("Calling console.log");  
    console.warn("Calling console.warn");  
    console.error("Calling console.error");  
}  
  
log_them();
```

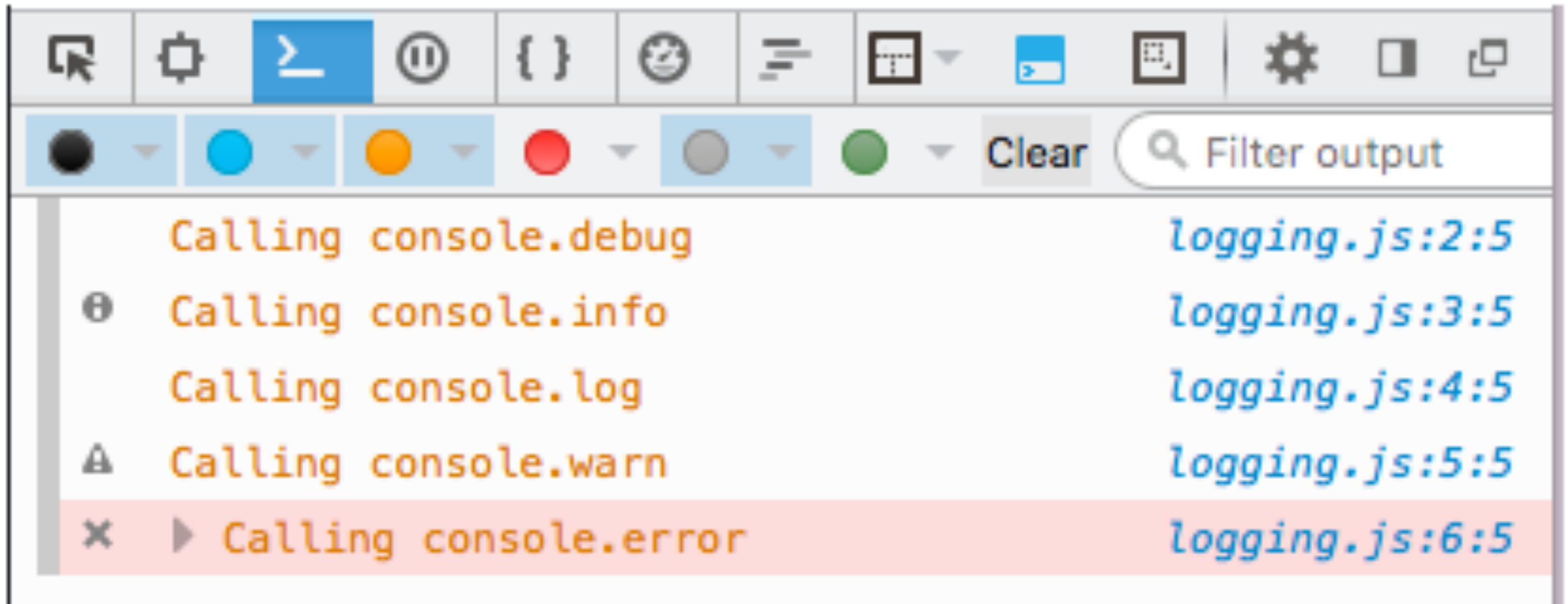
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <script src="logging.js"></script>
</head>
<body>
  <h2>Open the JavaScript console.</h2>
</body>
</html>
```



# Chrome



# Firefox



# console.table()

- display objects as a table
- click on headers to sort
- works with objects of objects

# console.trace

```
function add(x, y) {  
    console.trace('add called with ', x, 'and', y);  
    return x+y;  
}  
  
function calc() {  
    return add(8, 11) + add(9, 14);  
}  
  
function main() {  
    var x = add(2, 3);  
    var y = calc();  
}  
  
main();
```



▼add called with 2 and 3 logging\_trace.js:2

add @ logging\_trace.js:2

main @ logging\_trace.js:11

(anonymous) @ logging\_trace.js:15

---

▼add called with 8 and 11 logging\_trace.js:2

add @ logging\_trace.js:2

calc @ logging\_trace.js:7

main @ logging\_trace.js:12

(anonymous) @ logging\_trace.js:15

---

▼add called with 9 and 14 logging\_trace.js:2

add @ logging\_trace.js:2

calc @ logging\_trace.js:7

main @ logging\_trace.js:12

(anonymous) @ logging\_trace.js:15

---

# Why use `console.trace()`

- easier than writing a series of `console.log`
- remember to remove only one line for production
- shows file where it was called
- shows line number where it was called

# Debug

- find & locate the code for a particular function

# Why Do Error Occur?

- EvalError
- InternalError
- RangeError
- ReferenceError
- SyntaxError
- TypeError
- URIError
- MyCustomError

# window.onerror()

```
window.onerror = function(message, file, line, col, error) {  
    // send crash information back to your servers  
    console.log(error.stack);  
}
```

# error event

```
window.addEventListener("error", function(error) {  
  // send crash information back to your servers  
  console.log(error.stack);  
})
```



# 3 Truths 1 Lie

- I saw the only Gold Medal won by Great Britain
- I saw the only medal won by Mongolia
- I was an Olympic Torch Runner
- I attended the Opening Ceremony