POWERING POTENTIAL

Error Handling and Debugging

JavaScript Fundamentals

INTRODUCTION

Why you must debug

Understanding the Error object

- The Inbuilt Error types

Creating resilient code using try/catch statements

Throwing Errors

In Browser Debugging

- Examples with Developer Tools for Chrome

Console Debugging

- Logging to the console
- Breakpoints

When things go wrong….

Every modern desktop browser comes with the ability to debug

- As do many IDEs
- There are also testing frameworks

We will investigate these later in the course

The error object

If an exception occurs, an object representing the error is created

- If this error object is not caught, the program fails

The Error type is used to represent generic exceptions

- It has two properties

name – specifics the type of the exception

message – detailed exception information

```
let error = new Error("My error message");
```

- The above code declares an Error object where:

name is error

message is "My error message"

Common errors

There are a series of common errors built in, including:

- Range Error

```
let pi = 3.14159;
pi.toFixed(100000); // RangeError
```

- Reference Error

```
function foo() {
    bar++;             // ReferenceError
}
```

- Syntax Error

```
if (foo) { // SyntaxError – the closing curly brace is missing
```

- Type Error

```
const foo = {};
foo.bar(); // TypeError
```

Handling errors – try, catch and finally (1)C

- An unhandled error can cause a program to fail
  - With error handling, we can cause the program to degrade gracefully
- JavaScript supports a **`try ... catch ... finally`** block
  - Watch for exceptions thrown within the **`try`** block
  - If an errors occurs, the **`catch`** block runs
  - The **`finally`** block always runs
- You then **`throw`** an error object to the **`catch`** block
  - Setting the error’s **`message`** and **`name`**

Handling errors – try, catch and finally (2)C

```
try {
    let x = parseInt(prompt("Enter a number number",
""));
    if (isNaN(x)) {
        let e = new Error();
        e.message = "That wasn't a number";
        throw e;
    }
}
catch (e) {
    alert(`Something went wrong: ${e.message}`);
}
finally {
...
}
```

Try to execute the code

Create a new error object

Check the input

Throw the error

Catch the thrown error

Handle the error

Finally executes on success or failure

Throwing Exceptions

- When things go wrong meaningful messages help
  - We have seen there are inbuilt **`Error`** objects
- JavaScript allows programmers to throw their own exceptions
  - The **`throw`** keyword deliberately causes an error
  - Very useful when the function cannot solve the error itself
  - Any type can be thrown but the inbuilt error types are more useful

```
if (devisor === 0){
   throw new RangeError("Attempted division by zero!");
}
```

Things to remember

The “try…catch…finally” statement is used to handle exceptions

The “try” clause identifies code that could generate exceptions

The “catch” clause is only executed when an exception occurs

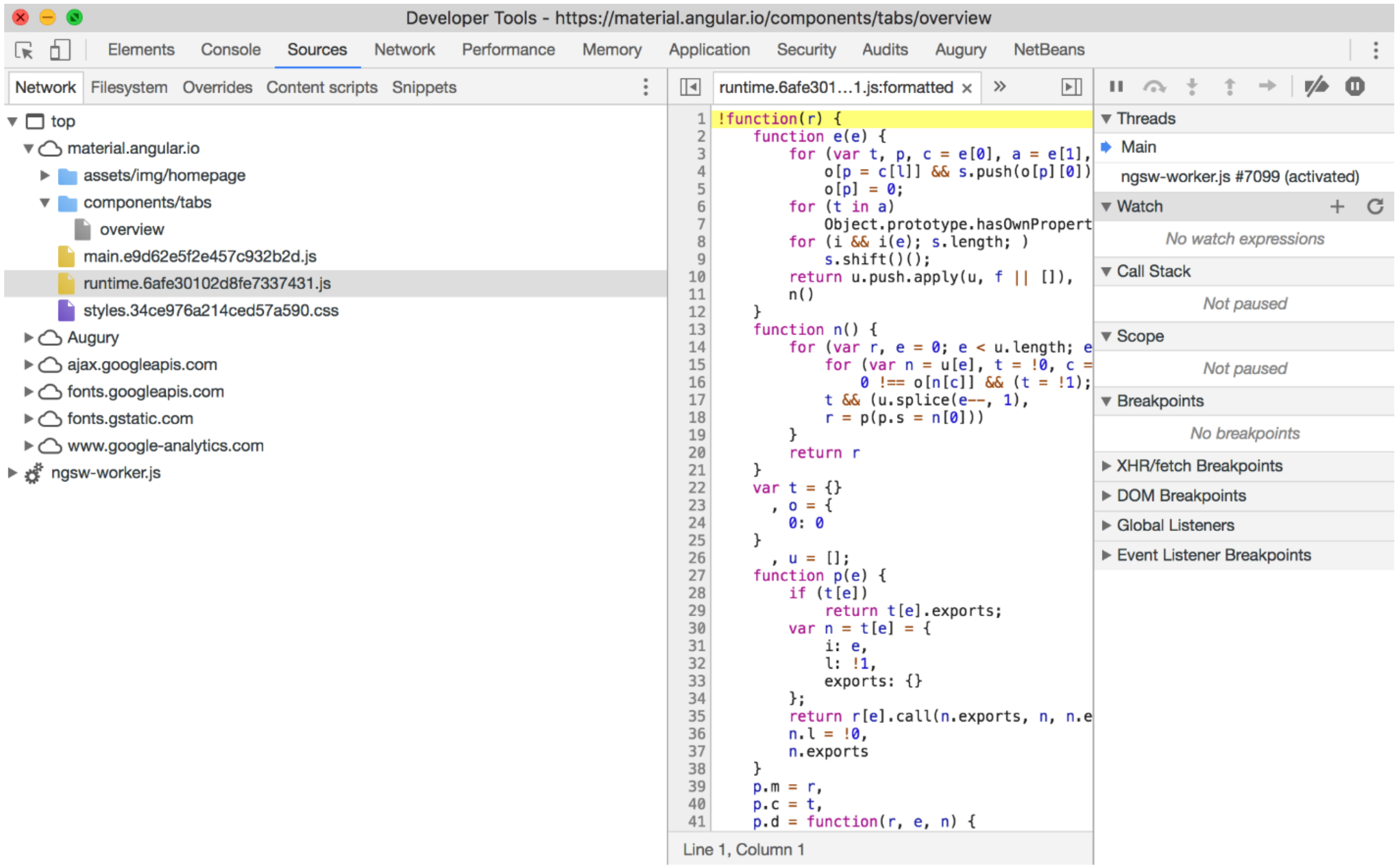
The “finally” clause is always executed, no matter what

The “throw” statement is used to generate exceptions

Debugging demonstration

Contains a JavaScript Console

- DOM explorer
- Console
- CSS style browser
- Can debug
  - Command line interface
  - Not trivial



```
!function(r) {
    function e(e) {
        for (var t, p, c = e[0], a = e[1],
            o[p = c[l]] && s.push(o[p][0])
            o[p] = 0;
        for (t in a)
            Object.prototype.hasOwnPropert
        for (i && i(e); s.length; )
            s.shift()();
        return u.push.apply(u, f || []),
        n()
    }
    function n() {
        for (var r, e = 0; e < u.length; e
            for (var n = u[e], t = !0, c =
                0 !== o[n[c]] && (t = !1);
            t && (u.splice(e--, 1),
            r = p(p.s = n[0]))
        }
        return r
    }
    var t = {}
      , o = {
        0: 0
    }
      , u = [];
    function p(e) {
        if (t[e])
            return t[e].exports;
        var n = t[e] = {
            i: e,
            l: !1,
            exports: {}
        };
        return r[e].call(n.exports, n, n.e
        n.l = !0,
        n.exports
    }
    p.m = r,
    p.c = t,
    p.d = function(r, e, n) {
```

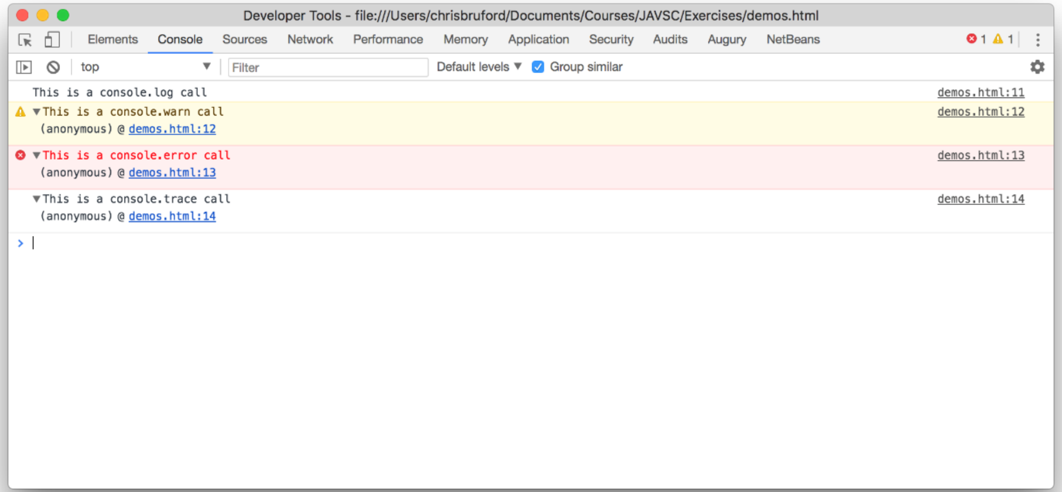
Debugging – Console debugging

- Learning to debug with a console is essential
  - Console API partially supported in most browsers
  - Full implementation in Chrome, Firebug and Safari
  - IE9 has good support, 8 some 7 little, 6 none
  - Opera supports some but went its own way
    - Different commands same concepts
- Never use alert function calls to debug your script
  - They are intrusive modal commands
  - That freeze the UI but not the runtime
  - Timers and AJAX calls are still executing

Debugging – Console logging

The console API has a number of useful functions

Function	Description
console.log()	Writes a message to the console.
console.warn()	As above with visual "warning" icon
console.error()	As above with visual "error" icon
console.trace()	As above with a call trace

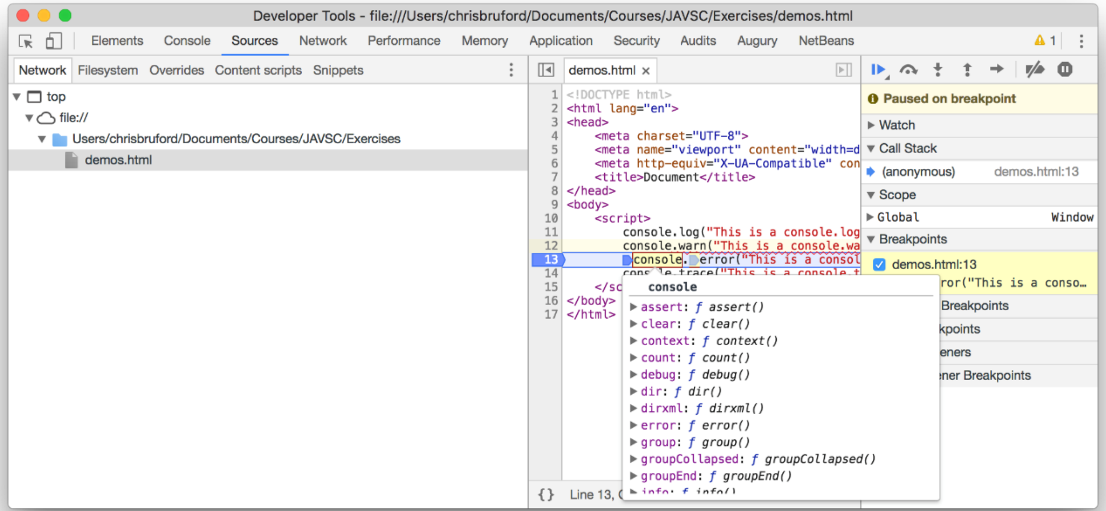


```
  This is a console.log call                                   demos.html:11
⚠ ▼This is a console.warn call                                 demos.html:12
    (anonymous) @ demos.html:12
⊗ ▼This is a console.error call                                demos.html:13
    (anonymous) @ demos.html:13
  ▼This is a console.trace call                                demos.html:14
    (anonymous) @ demos.html:14
>
```

Debugging – breakpoints

Breakpoints pauses the code allowing examination and debugging

- From the developer tools select the sources panel and select from a JavaScript source file
- Set breakpoint by clicking in the gutter of the line you want to pause execution at
- Hover over the source code to inspect variables and functions
- Delete the breakpoint by clicking the blue tag breakpoint indicator



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=d
    <meta http-equiv="X-UA-Compatible" con
    <title>Document</title>
</head>
<body>
    <script>
        console.log("This is a console.log
        console.warn("This is a console.wa
        console.error("This is a consol
        console.trace("This is a console.t
    </sc
</body>
</html>
```

Paused on breakpoint

(anonymous) demos.html:13

Global Window

✓ demos.html:13
ror("This is a conso…

Breakpoints
kpoints
eners
ener Breakpoints

```
console
▶ assert: ƒ assert()
▶ clear: ƒ clear()
▶ context: ƒ context()
▶ count: ƒ count()
▶ debug: ƒ debug()
▶ dir: ƒ dir()
▶ dirxml: ƒ dirxml()
▶ error: ƒ error()
▶ group: ƒ group()
▶ groupCollapsed: ƒ groupCollapsed()
▶ groupEnd: ƒ groupEnd()
▶ info: ƒ info()
```

REVIEW

Why you must debug

Understanding the error object

- The inbuilt error types

Creating resilient code using try/catch statements

Throwing errors

In browser debugging

- Examples with developer tools for chrome

Console debugging

- Logging to the console
- Breakpoints