JAVASCRIPT BASICS

CI435: Introduction to Web Development

Semester 2

Robin Heath (thanks to Marcus Winter)

Session overview

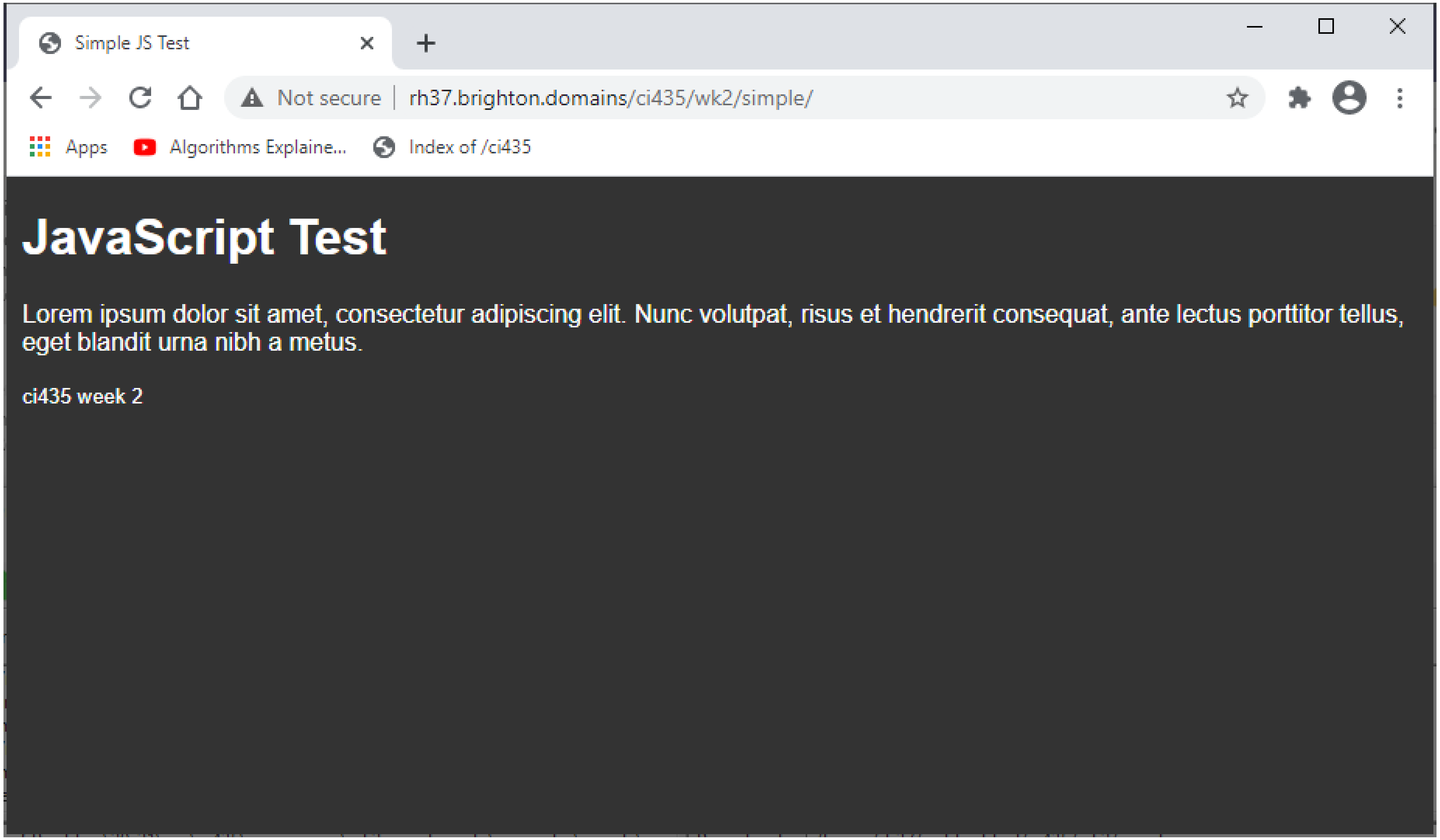
- Last week we looked at the module structure, the history of JavaScript and the development tools and examples we’ll use in this semester

- This week we will look at:
  - how to integrate JS with HTML
  - statements and comments
  - variables and their scope
  - data types and operators

- Next week we'll look at control structures

INTEGRATING JAVASCRIPT AND HTML

Example 1



JavaScript Test

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc volutpat, risus et hendrerit consequat, ante lectus porttitor tellus, eget blandit urna nibh a metus.

ci435 week 2

HTML <script> element

JavaScript *can* be embedded directly into a HTML page:

```
<script>
    console.log("Hello World");
</script>
```

This is considered bad practice as it mixes content (HTML) and behaviour (JS)

HTML <script> element

index.html

```
<html>
  <head>
    <script src="index.js"></script>
  </head>
  <body>
    ....
  </body>
</html>
```

Typically in the head section

Leave blank

Path to script file

index.js

```
console.log('Hello World');
```

It is best practice to keep JavaScript code in a separate file and link to it via the <script> element:

HTML <script> element

index.html

```
<html>
  <head>
  </head>
  <body>
    ....
  </body>
  <script src="index.js"></script>
</html>
```

index.js

```
console.log('Hello World');
```

More recently, it has become fashionable to include scripts at the bottom end of an HTML page.

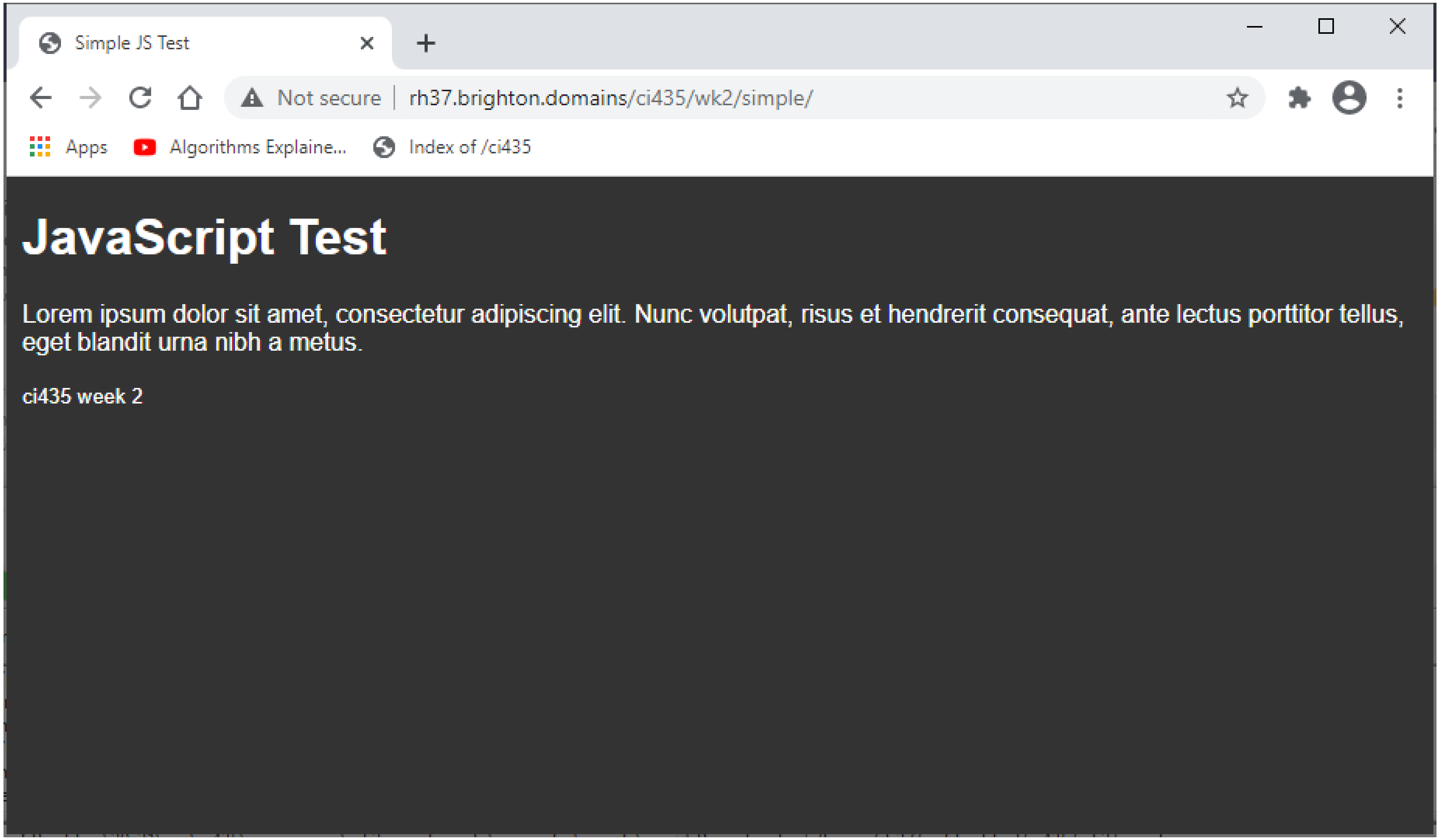
Why?

What are the implications?

» In our examples we'll stick with traditional practice and include scripts in the head section

STATEMENTS AND COMMENTS

Example 1



JavaScript Test

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc volutpat, risus et hendrerit consequat, ante lectus porttitor tellus, eget blandit urna nibh a metus.

ci435 week 2

Statements

- A program in JavaScript consists of a set of instructions called statements

- For example:

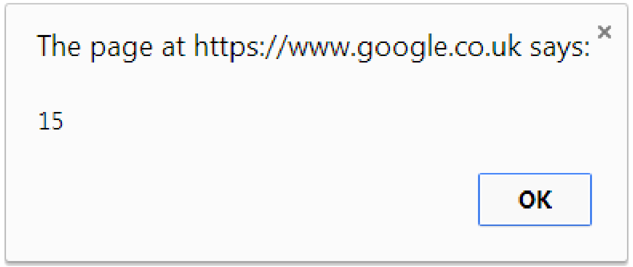
Code:

```
var a = 5;
var b = 10;
alert(a+b);
```

Statement

Semicolon (;)

Result:



The page at https://www.google.co.uk says:

15

Automatic semicolon insertion

Statements end with a semicolon;

- The semicolon at the end of a statement is *optional*
- If you forget it, the interpreter automatically inserts one, effectively guessing where your statement should end
- It does so silently, without warning or notice
- This is considered by many a bad feature of the language
- Automatic semicolon insertion can easily mask other problems in your code

- **Always use semicolons!**

Comments

Often you want to annotate your code to help you (or another developer) understand it, e.g. when looking at it a year later.

In order to tell the browser that these notes are NOT code, you mark them as comments.

Two forward slashes (//) indicates a single line comment:

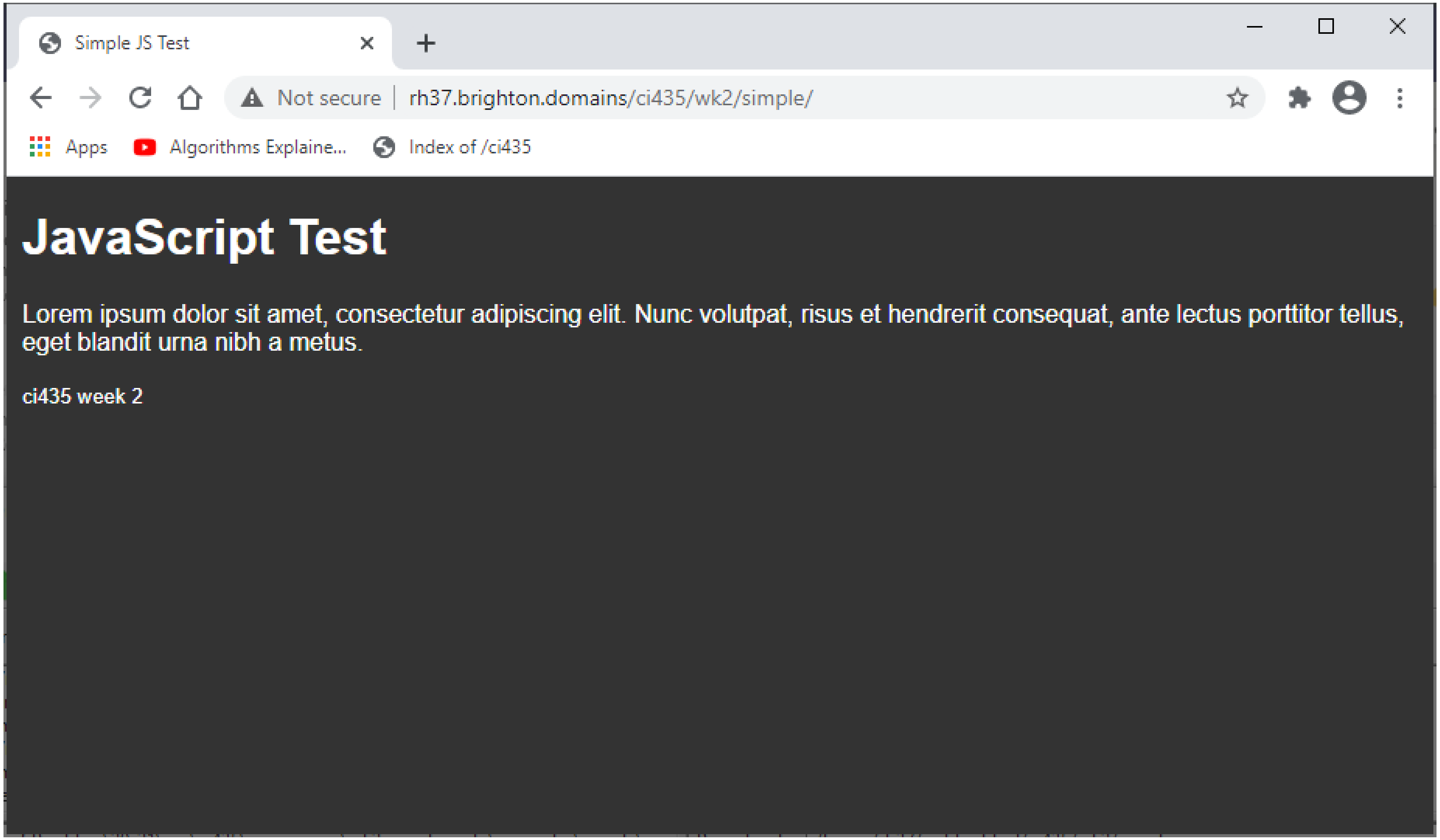
```
// This is a single-line comment
```

Multi lined comments are enclosed in `/* */`, e.g.

```
/* This comment
   is three
   lines long */
```

VARIABLES

Example 1

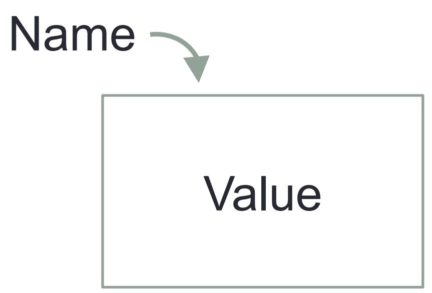


JavaScript Test

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc volutpat, risus et hendrerit consequat, ante lectus porttitor tellus, eget blandit urna nibh a metus.

ci435 week 2

Variables



- Variables store values
- The value in a variable can be changed
- Always declare (create) variables before using them

- Classic JavaScript uses the keyword `var` to declare a variable:

```
var color;
var quantity;
```

- Giving a variable a value is called **assignment**:

```
var color;
color = "green";  // assign the value "green"
var quantity = 25;   // declare and assign value 25
```

- Multiple variables can be **declared** and **initialized** together:

```
var color = "green",
    quantity = 25;
```

Variable scope

Classic JavaScript has no constants and no block scope variables, i.e. the value in a variable declared with **`var`** can always change and it is visible either everywhere (global scope) of inside a function (local scope).

Newer versions of JavaScript (ES6+) introduce **`let`** and **`const`** as alternative ways to declare variables:

- **`let:`** variable has block scope and can be reassigned

```
let color = "green";
color = "yellow"; // fine
```

- **`const:`** variable has block scope and cannot be reassigned

```
const color = "green";
color = "yellow"; // error
```

» We'll mainly use the classic **`var`** in our examples for this semester

Variable names

Variable names can contain letters, numbers, the dollar sign (`$`) and underscores - **but no spaces or hyphens**:

- These statements will produce a syntax error:

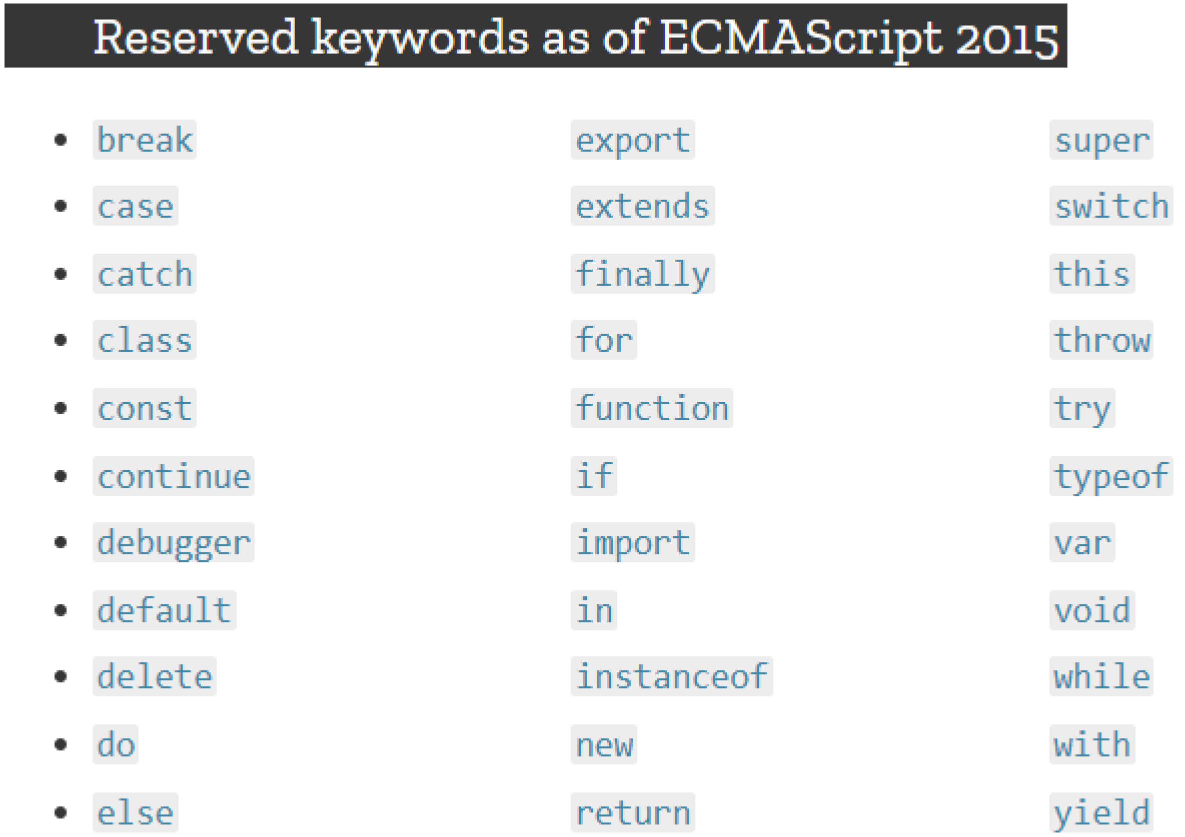
```
var box color = "green";// parse error
var box-color = "green";// parse error
```

- Use CamelCase to make variable names readable:

```
var boxColor = "green"; // ok: use CamelCase
```

Variable names

Reserved words in JavaScript **cannot be used as variable names**:



Reserved keywords as of ECMAScript 2015

- `break`
- `case`
- `catch`
- `class`
- `const`
- `continue`
- `debugger`
- `default`
- `delete`
- `do`
- `else`
- `export`
- `extends`
- `finally`
- `for`
- `function`
- `if`
- `import`
- `in`
- `instanceof`
- `new`
- `return`
- `super`
- `switch`
- `this`
- `throw`
- `try`
- `typeof`
- `var`
- `void`
- `while`
- `with`
- `yield`

Full list https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar

DATATYPES

Data types

```
var color_str = "green", quantity = 25;
```

- The above variables hold two different types of data (string and number) but JavaScript makes no distinction in how they are declared or assigned
- JavaScript is **weakly typed** allowing variables to hold different types of data at any point
- The following would produce an error in strongly typed languages (e.g. Java) but is perfectly fine in JavaScript:

```
  var age_str= "twenty";
  age_str = 20;
```

Data type: String

- Strings are enclosed in single quotes or double quotes:

```
Mood_str = "happy";
mood_str = 'happy';
```

- If the string contains single or double quotes as part of its content, they must be **escaped** using a backslash (\) character:

```
var mood_str = 'don't ask';// parse error
var mood = 'don\'t ask';    // this works

var mood = "don't ask";     // this also works
var mood = "don\'t ask";    // this also works
```

String concatenation

Strings can be **concatenated** with other strings (and any other data type) with the **+** sign:

```
var greeting = "Hello";
var name = "Claire";
var num = 1000000;
var text = greeting + " " + name
         + ", you have " + num
         + " followers (you wish)";

console.log(text);  // output to console
```

» We'll learn more about strings in a later session...

Data Type: Number

- JavaScript numbers are by default floating-point numbers (i.e. they can have decimal places) with double precision

```
var age = 21;
var age = 21.25;
var age = 21.251498629;
```

- Negative numbers can be used with a minus sign in front of the number

```
var temperature = -10.44;
```

Data Type: Number

The number type has three symbolic values:

- **`+Infinity`**

```
console.log(42 / 0);              // Infinity
```

- **`-Infinity`**

```
console.log(-42 / 0);             // -Infinity
```

- **`NaN`** `// Not a Number`

```
console.log(42 / "hello");         // NaN
```

Arithmetic Operators

```
var a = 17;
var b = 5;

console.log(a + b);     // 23    (addition)
console.log(a - b);     // 12    (subtraction)
console.log(a * b);     // 85    (multiplication)
console.log(a / b);     // 3.4   (division)
console.log(a % b);     // 2     (modulo)
```

Modulo results in the remainder of a division. This is sometimes needed in programming, e.g. to test if a number is even or odd:
if (number % 2) results in 1, then the number is odd.

Increment (++) and decrement (−−)

The increment (++) and decrement (−−) operators increase or decrease the value of a number by 1

```
year++;       is equivalent to:    year = year + 1;
year--;       is equivalent to:    year = year - 1;
```

Incrementing/decrementing can be done before or after the variable is read:

```
var a = 1;
var b = a++;// b=1, a=2
var c = ++b;// c=2, b=2
```

Increment and decrement are often used in loops (we'll cover that later)

Operator precedence

Arithmetic operations are executed in the usual way (**BODMAS**):

- **B**rackets first
- **O**rders (powers, roots, increments, etc.)
- **D**ivision and **M**ultiplication (left-to-right)
- **A**ddition and **S**ubtraction (left-to-right)

Use brackets to specify the order of operations, e.g.

```
var farenheit = 95;
var celisius = (farenheit - 32) / 1.8;
```

Converting strings to numbers

Sometimes we get numeric data as a string (e.g. form field).

If we use a string value in arithmetic operations, JavaScript **automatically** converts it to a number. This happens silently and can mask other errors.

```
var fahrenheit_str = "95";
var celsius = (fahrenheit_str - 32) / 1.8;  // bad practice
```

It is **best practice to explicitly convert strings** to numbers before using them in calculations:

```
var fahrenheit_str = "95";
var fahrenheit = parseInt(fahrenheit_str);
var celsius = (fahrenheit - 32) / 1.8;   // good practice
```

Number conversion examples

`Built-in JavaScript functions:`

`parseInt()` and `parseFloat()`

```
parseInt('10')    // 10
parseInt('10 tons') // 10
parseInt('0xFF', 16);  // 255     (hexadecimal)*
parseInt('1111', 2);   // 15      (binary)*
parseInt('oops')        // NaN     (Not a Number)

parseFloat('1.0')       // 1.0
parseFloat('-1.0')   // -1.0
parseFloat('69.5%') // 69.5
parseFloat('  1.0') // 1.0
parseFloat('oops')   // NaN     (Not a Number)
```

* parseInt() can take an optional **radix** parameter for non-decimal integer numbers

Data Type: Boolean

- Boolean values are either **true** or **false**
- Suppose we wanted a variable to represent whether a lecturer was late to a session or not:

```
var late_bool = true;       // boolean true
```

Note: Putting quotes around the value makes it a string, not a boolean:

```
var late_bool = "true"; // string "true" (!)
```

Data Type: Array

- Arrays are list-like objects containing multiple values
- Values in an array are accessed by an index
- Note that array indices start at 0 not 1
- Arrays can be declared with [square brackets]

- Suppose we wanted to represent a list of friends:

```
var friends_ary = ['Tom', 'Zoe', 'Bob', 'Mary'];

console.log(friends_ary[1]);          // Zoe
```

→ We'll learn more about arrays in a later session...

Data Type: Object

- Objects have a set of properties and/or methods
- JavaScript objects can be declared with {curly brackets}
- Suppose we wanted to represent a person:

```
var person_obj = { name: 'Zoe',
                   age: 21,
                   vegetarian: false};
```

- Newer versions of JavaScript (ES6+) also support `class` definitions and instantiation via the keyword `new`

→ We'll learn more about objects in a later session...

Determining types

`typeof` can be used to find the type of a variable:

```
console.log(typeof 42);         // number
console.log(typeof "42");       // string
console.log(typeof true);       // boolean
console.log(typeof [1,2,3]);    // object
console.log(typeof {name: "Zoe", age: 21});
                  // object
```

Two special types:

`null`	represents the intentional absence of any object value
`undefined`	represents a declared variable that has no value

Recommended reading

HTML <script> element

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script

Mozilla's Developer Network: JavaScript grammar and types

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types

KEEP
CALM
AND
KEEP
CODING