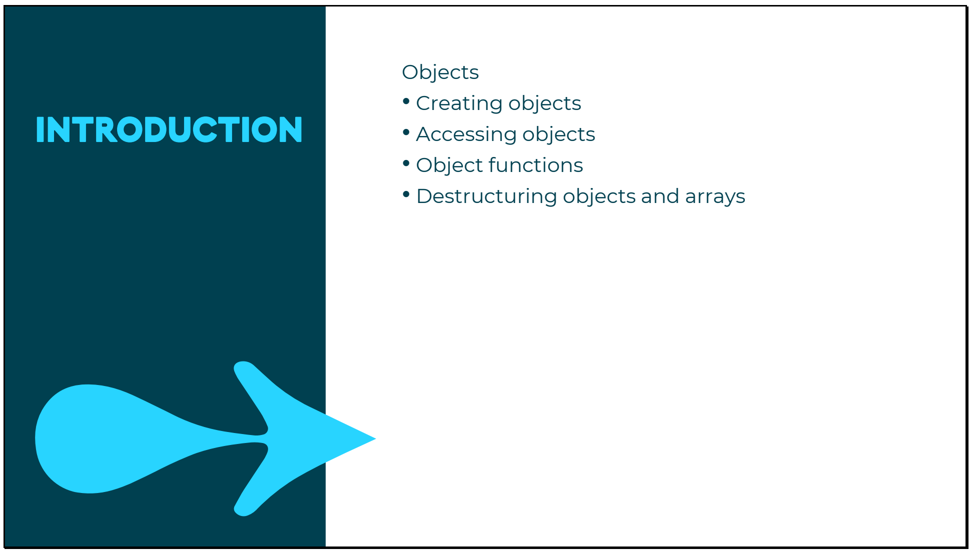
Slide 1

OBJECTS

JavaScript Fundamentals

Slide 2

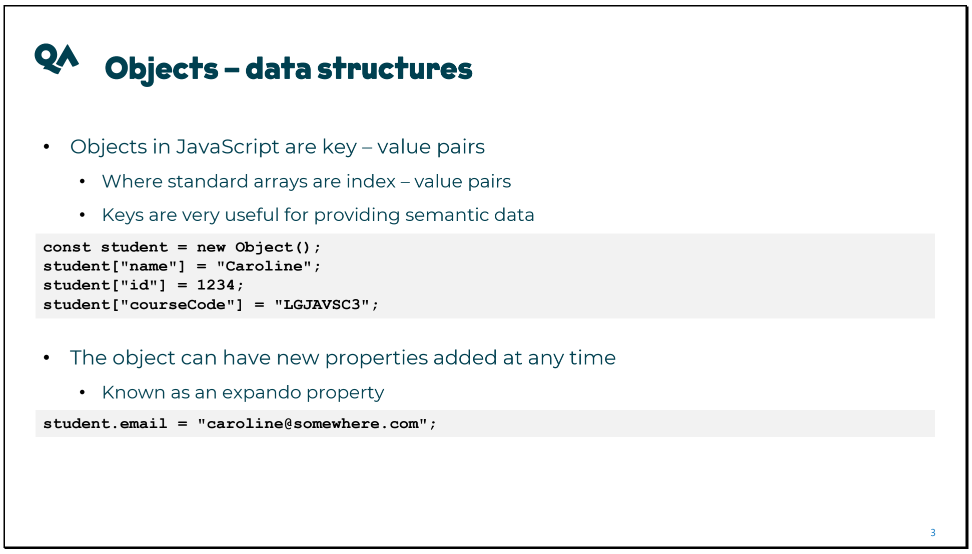


INTRODUCTION

Objects

- Creating objects
- Accessing objects
- Object functions
- Destructuring objects and arrays

Slide 3



Objects – data structures

- Objects in JavaScript are key – value pairs
  - Where standard arrays are index – value pairs
  - Keys are very useful for providing semantic data

```
const student = new Object();
student["name"] = "Caroline";
student["id"] = 1234;
student["courseCode"] = "LGJAVSC3";
```

- The object can have new properties added at any time
  - Known as an expando property

```
student.email = "caroline@somewhere.com";
```

3

Many programming paradigms can be used within JavaScript, one of which is object oriented programming. It provides a series of input objects we will shortly be examining that include window and document and their numerous offspring, which are very important – but they are defined by the browser, not by the programmer. We can define our own objects.
Objects are principal objects in JavaScript. If the variable does not refer to a primitive type, it is an object of some kind. In principal, they are very similar to the concepts of arrays but, instead of an indexed identifier, a string-based key is used (in fact – Arrays in JavaScript are nothing more than special objects).

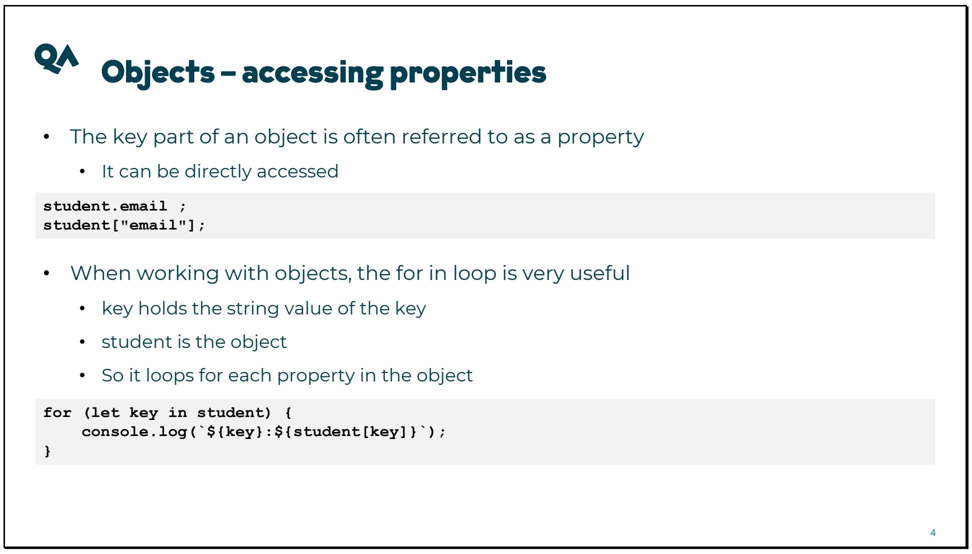
the property…

```
student["name"]
```

can also be read or written by calling:

```
student.name
```

Slide 4



Objects – accessing properties

- The key part of an object is often referred to as a property
  - It can be directly accessed

```
student.email ;
student["email"];
```

- When working with objects, the for in loop is very useful
  - key holds the string value of the key
  - student is the object
  - So it loops for each property in the object

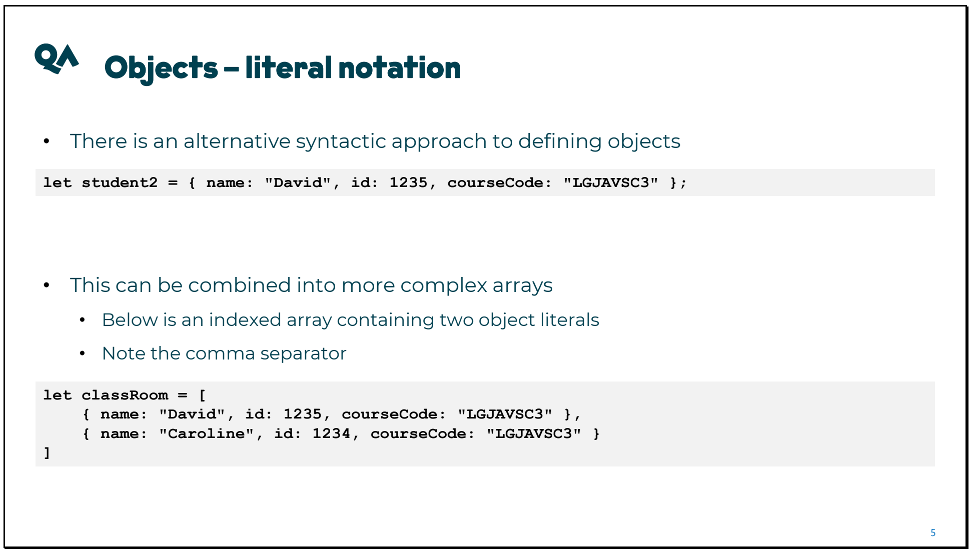
```
for (let key in student) {
    console.log(`${key}:${student[key]}`);
}
```

4

Objects are collections of properties and every property gets its own standard set of internal properties. (We can think of these as abstract properties – they are used by the JavaScript engine but aren't directly accessible to the user. ECMAScript uses the [[*property*]] format to denote internal properties).

One of these properties is [[Enumerable]]. The for-in statement will iterate over every property for which the value of [[Enumerable]] is true. This includes enumerable properties inherited via the prototype chain. Properties with an [[Enumerable]] value of false, as well as *shadowed* properties – i.e. properties that are overridden by same-name properties of descendant objects – will not be iterated.

Slide 5



Objects – literal notation

- There is an alternative syntactic approach to defining objects

```
let student2 = { name: "David", id: 1235, courseCode: "LGJAVSC3" };
```

- This can be combined into more complex arrays
    - Below is an indexed array containing two object literals
    - Note the comma separator

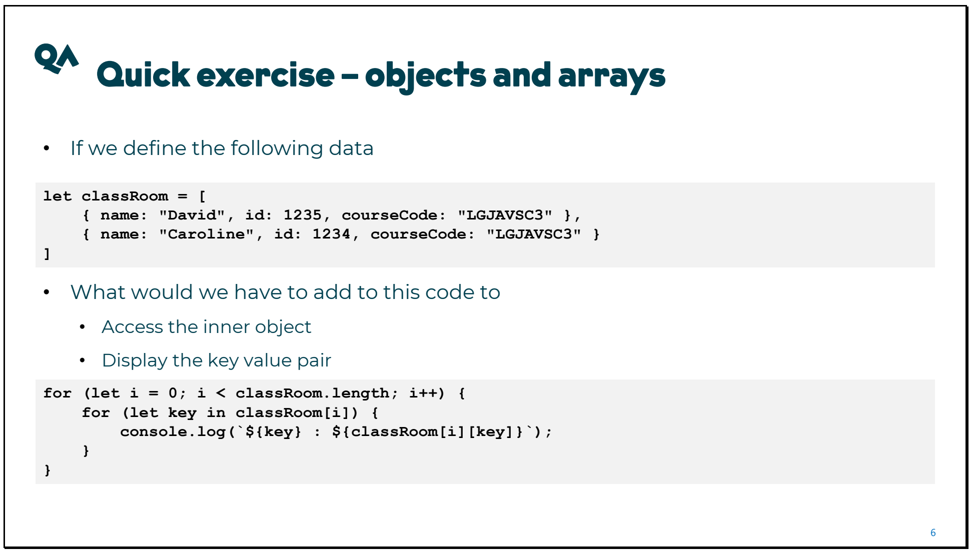
```
let classRoom = [
    { name: "David", id: 1235, courseCode: "LGJAVSC3" },
    { name: "Caroline", id: 1234, courseCode: "LGJAVSC3" }
]
```

5

The above code is a quick implementation for JavaScript object. It initialises the object and sets three properties.

The second example creates an indexed array of object literals

Slide 6



Quick exercise – objects and arrays

- If we define the following data

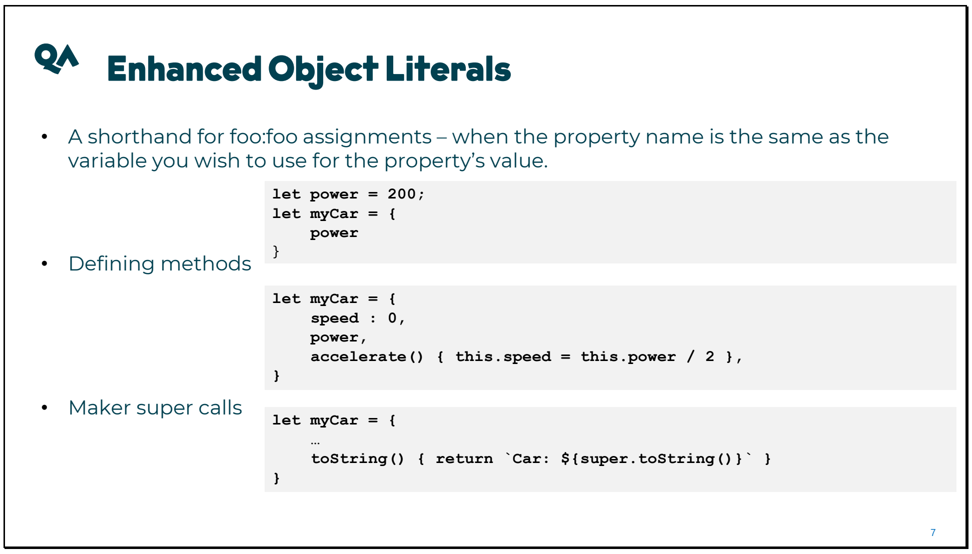
```
let classRoom = [
    { name: "David", id: 1235, courseCode: "LGJAVSC3" },
    { name: "Caroline", id: 1234, courseCode: "LGJAVSC3" }
]
```

- What would we have to add to this code to
  - Access the inner object
  - Display the key value pair

```
for (let i = 0; i < classRoom.length; i++) {
    for (let key in classRoom[i]) {
        console.log(`${key} : ${classRoom[i][key]}`);
    }
}
```

6

Slide 7



Enhanced Object Literals

- A shorthand for foo:foo assignments – when the property name is the same as the variable you wish to use for the property's value.

```
let power = 200;
let myCar = {
    power
}
```

- Defining methods

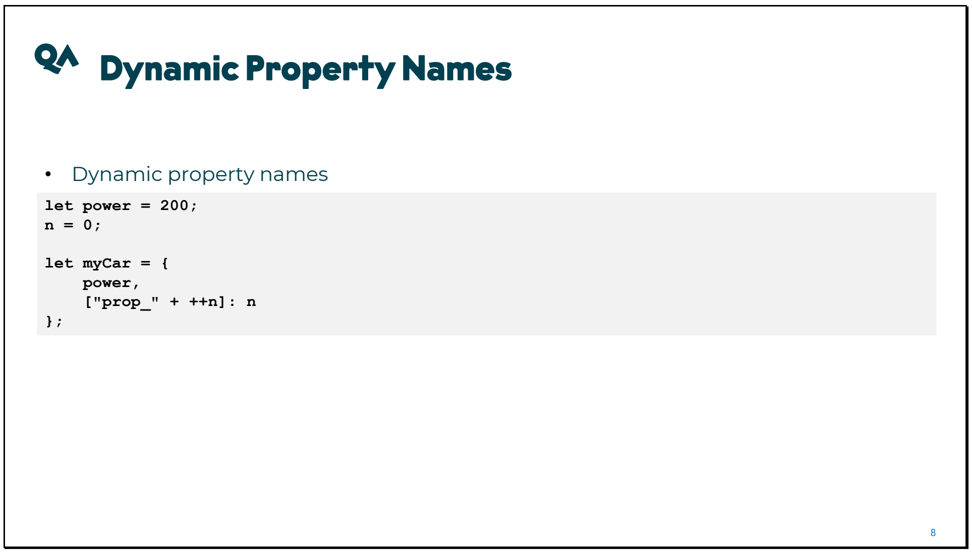
```
let myCar = {
    speed : 0,
    power,
    accelerate() { this.speed = this.power / 2 },
}
```

- Maker super calls

```
let myCar = {
    …
    toString() { return `Car: ${super.toString()}` }
}
```

7

Slide 8

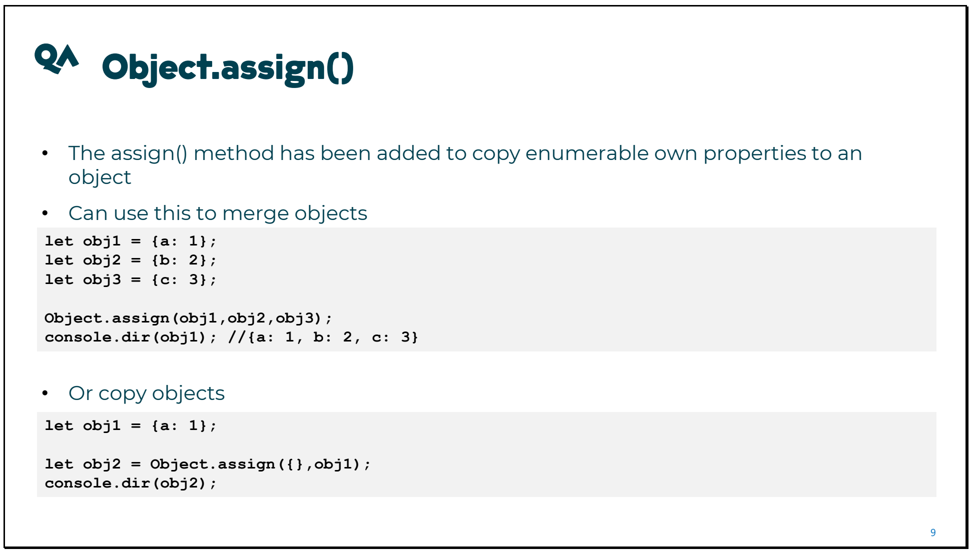


Dynamic Property Names

- Dynamic property names

```
let power = 200;
n = 0;

let myCar = {
    power,
    ["prop_" + ++n]: n
};
```

Slide 9



Object.assign()

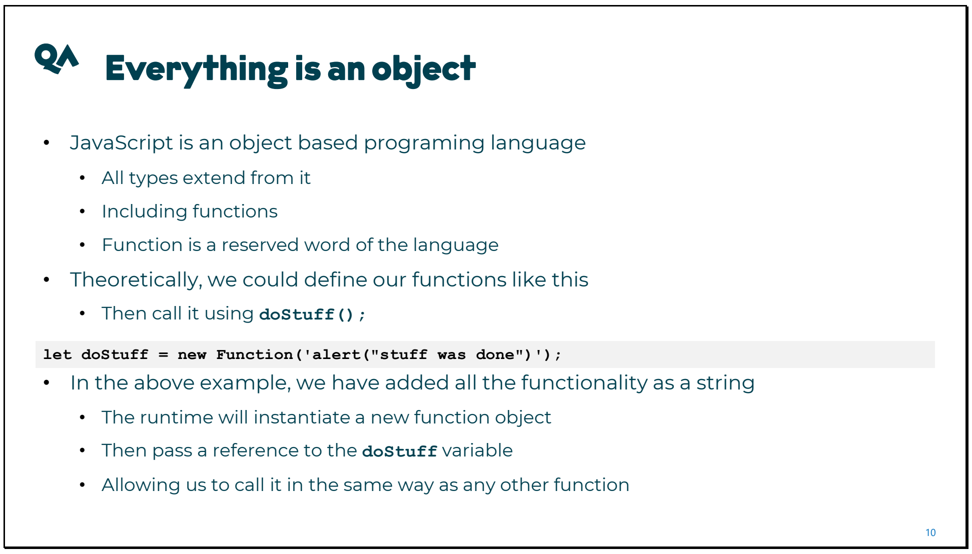
- The assign() method has been added to copy enumerable own properties to an object
- Can use this to merge objects

```
let obj1 = {a: 1};
let obj2 = {b: 2};
let obj3 = {c: 3};

Object.assign(obj1,obj2,obj3);
console.dir(obj1); //{a: 1, b: 2, c: 3}
```

- Or copy objects

```
let obj1 = {a: 1};

let obj2 = Object.assign({},obj1);
console.dir(obj2);
```

Slide 10



Everything is an object

- JavaScript is an object based programing language
  - All types extend from it
  - Including functions
  - Function is a reserved word of the language
- Theoretically, we could define our functions like this
  - Then call it using `doStuff();`

```
let doStuff = new Function('alert("stuff was done")');
```

- In the above example, we have added all the functionality as a string
  - The runtime will instantiate a new function object
  - Then pass a reference to the `doStuff` variable
  - Allowing us to call it in the same way as any other function

10

Objects are the building blocks of the JavaScript language. In fact, it is defined as an *object based programming language*. Absolutely everything we work with in JavaScript has an object at its core. Consider the following code:

```
let x = 5;
const y = new Number(5);
```

Both code implementations create an object of type number and the variable then receives a memory reference to that object. It is important to remember that JavaScript variables are simply pointers to this object.

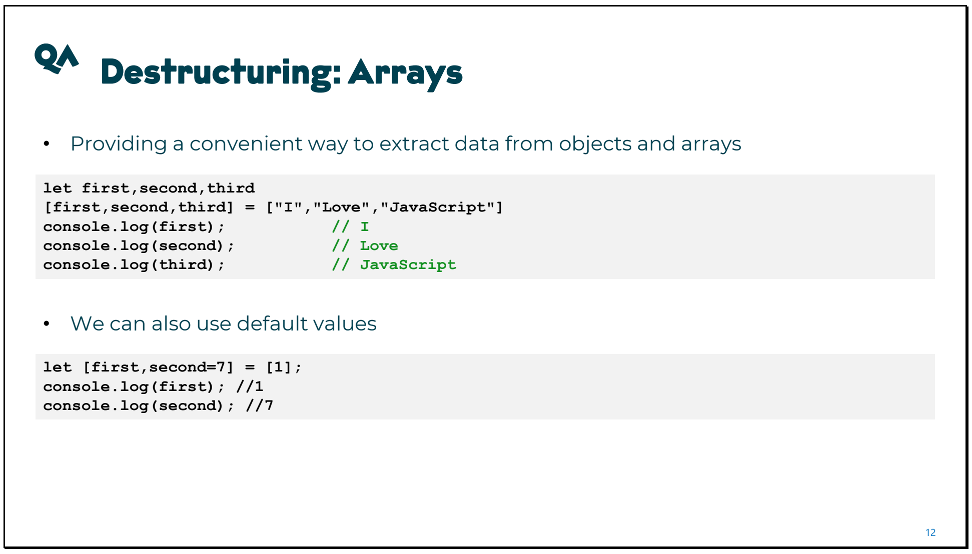
This concept extends to functions. In the code block above, we see another way of creating a function. The **new** keyword instantiates a function, with the logic passed in as a string. **doStuff** receives a reference to the function. As long as the variable **doStuff** remains in scope, the function remains available.

Slide 11

DESTRUCTURING

JavaScript Fundamentals

Slide 12



Destructuring: Arrays

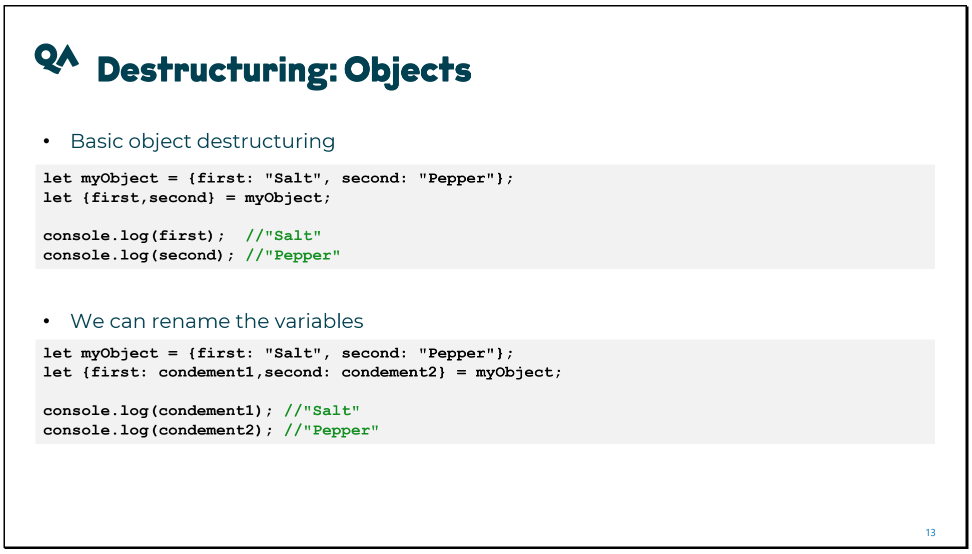
- Providing a convenient way to extract data from objects and arrays

```
let first,second,third
[first,second,third] = ["I","Love","JavaScript"]
console.log(first);           // I
console.log(second);          // Love
console.log(third);           // JavaScript
```

- We can also use default values

```
let [first,second=7] = [1];
console.log(first); //1
console.log(second); //7
```

Slide 13



Destructuring: Objects

- Basic object destructuring

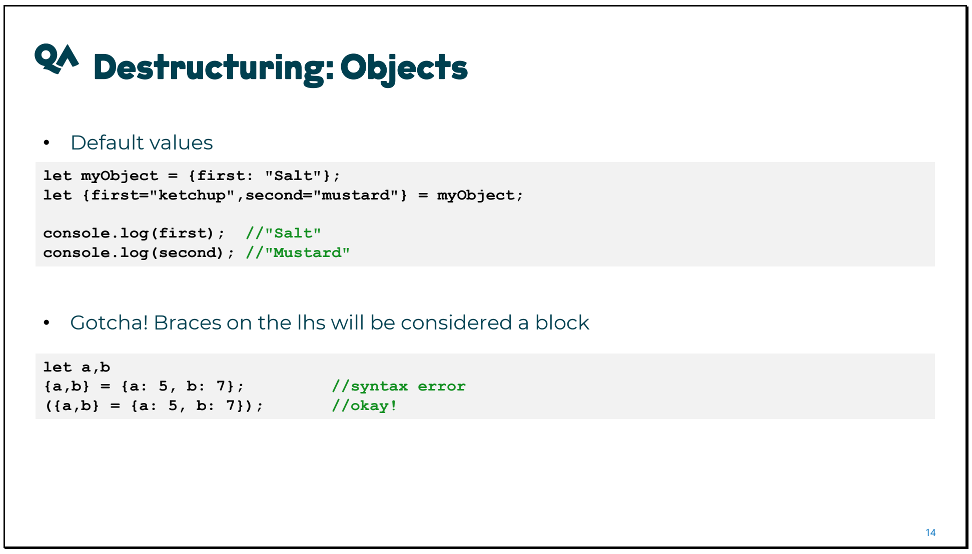
```
let myObject = {first: "Salt", second: "Pepper"};
let {first,second} = myObject;

console.log(first);  //"Salt"
console.log(second); //"Pepper"
```

- We can rename the variables

```
let myObject = {first: "Salt", second: "Pepper"};
let {first: condement1,second: condement2} = myObject;

console.log(condement1); //"Salt"
console.log(condement2); //"Pepper"
```

13

Slide 14



Destructuring: Objects

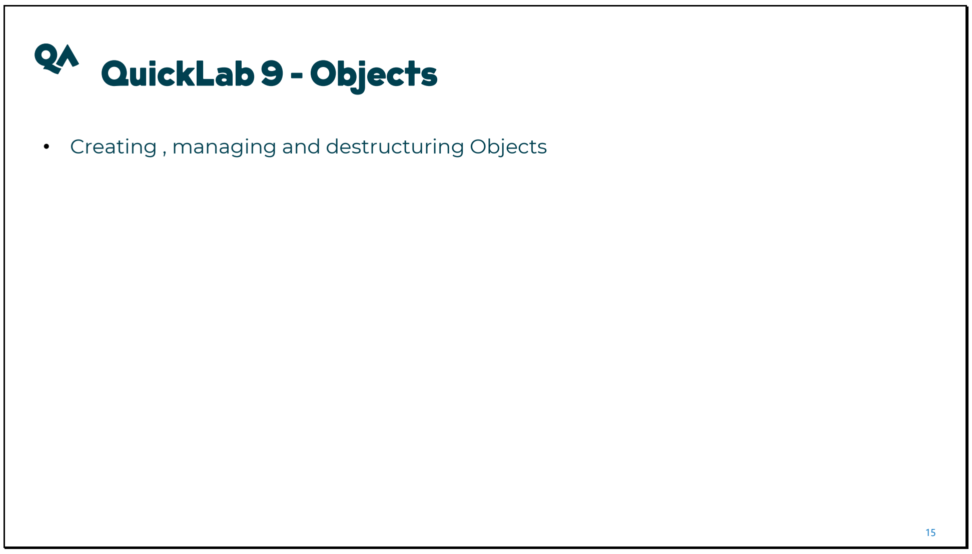
- Default values

```
let myObject = {first: "Salt"};
let {first="ketchup",second="mustard"} = myObject;

console.log(first);  //"Salt"
console.log(second); //"Mustard"
```

- Gotcha! Braces on the lhs will be considered a block

```
let a,b
{a,b} = {a: 5, b: 7};         //syntax error
({a,b} = {a: 5, b: 7});       //okay!
```

Slide 15

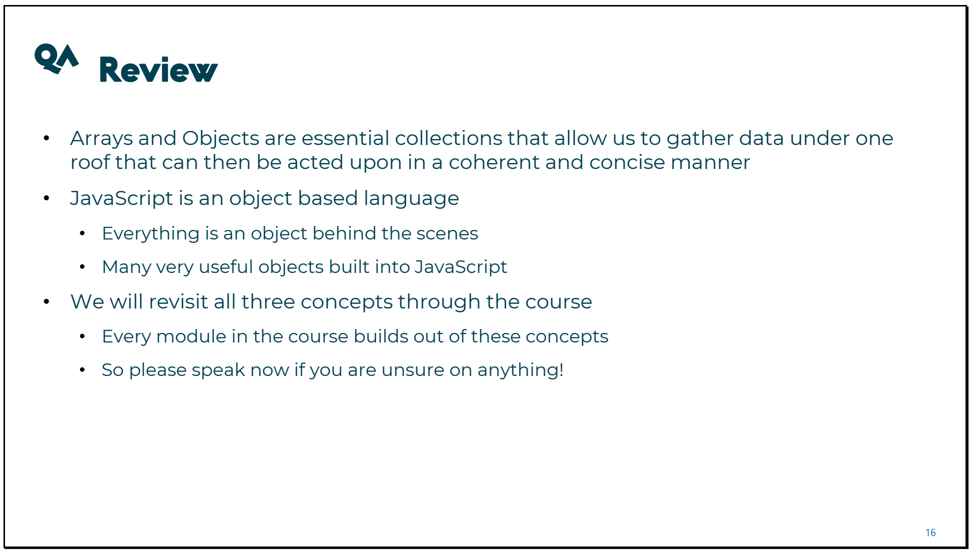


QuickLab 9 - Objects

- Creating , managing and destructuring Objects

15

Slide 16



Review

- Arrays and Objects are essential collections that allow us to gather data under one roof that can then be acted upon in a coherent and concise manner
- JavaScript is an object based language
  - Everything is an object behind the scenes
  - Many very useful objects built into JavaScript
- We will revisit all three concepts through the course
  - Every module in the course builds out of these concepts
  - So please speak now if you are unsure on anything!