ECMAScript Quick Start

Day 01

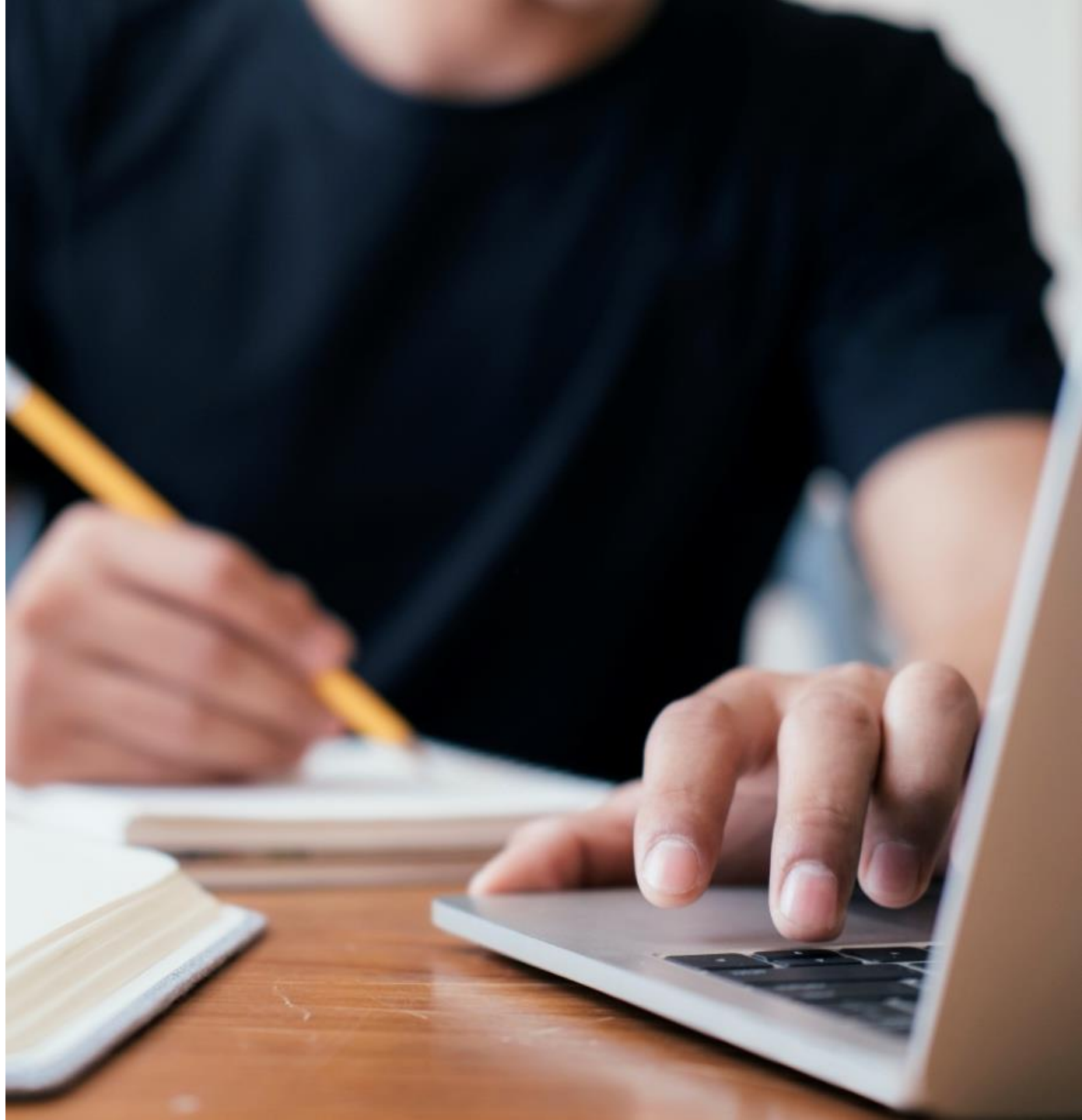
Axle Barr

Bootcamp

AGENDA

Day 1

- Review of functions
- CommonJS Modules
- Example using imports and exports
- Arrow (lambda) functions
- Var, let, and const



Environment Setup

Linux + NodeJS + VS Code

At a Linux terminal run the command, at any **folder** of your choice:

```
npm init
```

Accept all the defaults

In the same folder, `touch` a js file called index.js, so:

```
touch index.js
```

Open index.js in an editor or in the default text editor that comes with Linux.

Add this line just to make sure it works:

```
console.log("Hello");
```

The index.js file will become the main file for code for the rest of the bootcamp.

Quick Review of Functions

Functions Review

```
function x(){
  return "Hi I am function X ";
}
function y(){
  return "Hi I am function Y";
}
//
console.log(y);
//console.log(y());
```

Quick Review of Functions

Functions Review

```
function x(y){
  return "Hi I am function X ";
}
function y(){
  return "Hi I am function Y";
}
//
console.log(x);
```

Quick Review of Functions

Functions Review

```
function x(y){
  return "Hi I am function X  working with Y\n” + y();
}
function y(){
  return "Hi I am function Y";
}
//
console.log(x(y));
```

```
axle@pc0484:~/Documents/ECMAScript/FunctionPassing$ node index
Hi I am function X working with Y Hi I am function Y
axle@pc0484:~/Documents/ECMAScript/FunctionPassing$ node index
Hi I am function X working with Y
Hi I am function Y
```

Create this class in a file called employee.js

```
class Employee {
        name = "";
        department = "";
        constructor(name, department) {
         this.name = name;
         this.department = department;
        }
        get name() {
           return this.name;
        }
        employeeDetails() {
           return this.name + " works in " + this.department;
  };
}
```

CommonJS Modules

Export the class

```
export class Employee {
        name = "";
        department = "";
        constructor(name, department) {
         this.name = name;
         this.department = department;
        }
        get name() {
           return this.name;
        }
        employeeDetails() {
           return this.name + " works in " + this.department;
  };
}
```

CommonJS Modules

Import the employee class into index.js

```
import { Employee } from "./employee.js";

const employee = new Employee("Axle", "IT");

console.log(employee.employeeDetails());
```

CommonJS Modules

Export the class, alternative

```
class Employee {
        name = "";
        department = "";
        constructor(name, department) {
         this.name = name;
         this.department = department;
        }
        get name() {
           return this.name;
        }
        employeeDetails() {
           return this.name + " works in " + this.department;
  };
};
export { Employee };
```

CommonJS Modules

Export an object

```
class Employee {
        name = "";
        department = "";
        constructor(name, department) {
         this.name = name;
         this.department = department;
        }
        get name() {
           return this.name;
        }
        employeeDetails() {
           return this.name + " works in " + this.department;
  };
};
const employee = new Employee("Axle", "Software");
export default employee;
```

CommonJS Modules

Export an object, then import it in index.js

employee.js

```
class Employee {
        name = "";
        other code here …
  };
};

const employee = new Employee("Axle", "Software");

export default employee;
```

index.js

```
import employee from "./employee.js";
console.log(employee.employeeDetails());
```

CommonJS Modules

Multiple exports

```
class Employee {
        name = "";
        department = "";
        constructor(name, department) {
         this.name = name;
         this.department = department;
        }
        get name() {
           return this.name;
        }
        employeeDetails() {
           return this.name + " works in " + this.department;
  };
};
export const colors = ["Orange", "Yellow", "Green"];
export default Employee;
```

CommonJS Modules

Importing from multiple exports

```
import Employee, {colors} from "./employee.js";

const e = new Employee("Axle", "Programming");

console.log(e.employeeDetails() + "," + colors[2]);
```

CommonJS Modules

Real-world Example

At your terminal window, run the following command to install *ExpresJS*:
npm install express

Delete the exployee.js file from the folder but keep it somewhere, we won't be using it for this example.

After running the command above, you will get a new folder called node_modules, this folder will contain the **ExpressJS** module that we will import and use

Your package.json file will also change to reflect the installation we just did, but that is not important for this bootcamp

Real-world Example

First import the package

```
import express from 'express’;
```

Then create a variable and point it to the express constructor

```
const app = express();
```

Now that we have an object, we can use the methods like get()

```
app.get('/', function(req, res){
  res.send('Hello from Skillsoft!’);
});
```

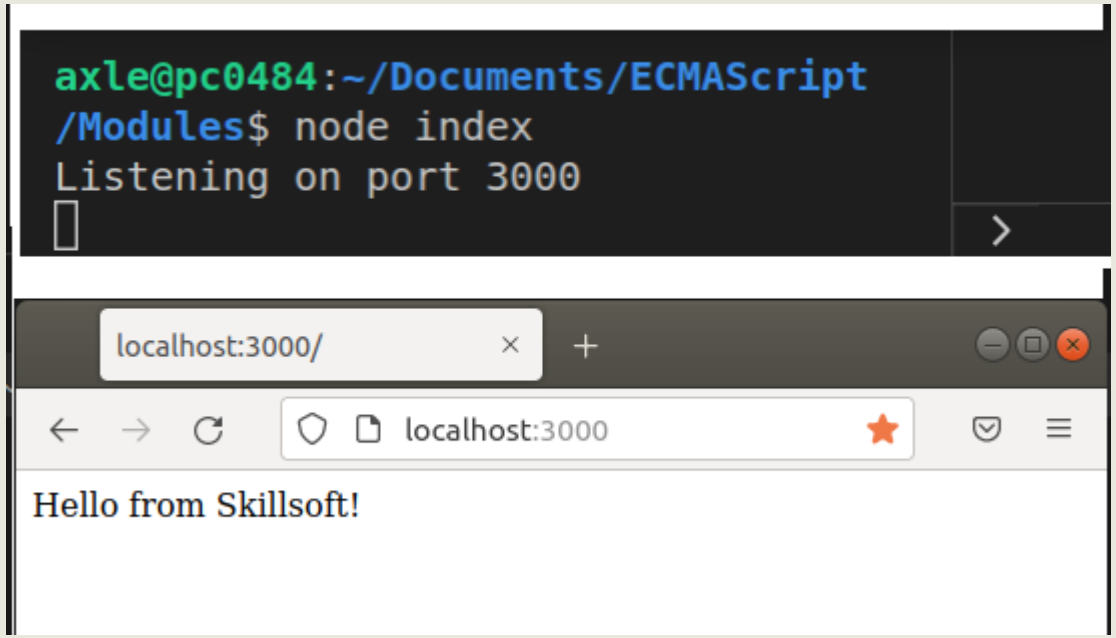
There is also a listen() method we can use

```
app.listen(3000, function(){
  console.log("Listening on port 3000");
});
```

Real-world Example

At the terminal window, run the index file like you did before

Once you get the response “Listening on port 3000” open a browser and navigate to that port



Function Expression

Traditional Functions

```
const x = function (num1, num2){
  return num1 + num2;
}
//
console.log( x(3,5) );
```

Receive two values via the parameters

Supply two values to the function called x

Function Expression

```
const x = function (num1, num2){
  return num1 + num2;
}
//
console.log( x (3,5) );
```

Throw (pass) the Values to the Function

Arrow Functions (Lambda Expressions)

```
const x = (num1, num2) => {return num1 + num2};
//
console.log( x(3,5) );
```

We can use just one line

Arrow Functions (Lambda Expressions)

No need for return keyword or curly braces

```
const x = (num1, num2) => num1 + num2;
//
console.log( x(3,5) );
```

Arrow Functions (Lambda Expressions)

Parameters were passed and => instead of function

```
const x = (num1, num2) => num1 + num2;
//
console.log( x(3,5) );
```

function

The same parameters were passed into the function, just we don't call it function we call it =>

Arrow Functions (Lambda Expressions)

Only one parameter, no need for parenthesis

```
const x = num1 => num1 + num1;
//
console.log( x(3,5) );
```

Also if there is just one parameter, then we don't need the parenthesis for the parameters

Arrow Functions (Lambda Expressions)

If no parameters, we do need parenthesis

```
const x = () => 5 + 3;
//
console.log( x(3,5) );
```

However if there are NO parameters you must indicate this with empty parenthesis

Real-world Example

Delete everything from the folder and paste in the two files from the previous real-world example

Run the command **npm install** to restore the node_modules folder and other necessary supporting files

Run the *index* file via node to make sure it works, so node index

If you see the response “listening on port 3000” then the code works like it did before

On the next slide we will transform all the current functions to use arrows instead

Real-world Example

Go from this:

```
app.get('/', function(req, res){
   res.send('Hello from Skillsoft!’);
});
```

To this:

```
app.get('/', (req, res) => {
   res.send('Hello from Skillsoft!’);
});
```

Finally:

```
app.get('/', (req, res) =>  res.send('Hello from Skillsoft!’) );
```

Real-world Example

Go from this:

```
app.listen(3000, function(){
   console.log("Listening on port 3000");
});
```

To this:

```
app.listen(3000, () => console.log("Listening on port 3000"));
```

Variables and Declarations

```
const num = 5;
var num2 = 6;
let num3 = 7;
function x(){
  return "Global variable sum = " + (num + num2 + num3);
}
//
console.log(x())
```

Three ways to declare and use variables

Variables and Declarations

Using *let* in different scope:

```
const num = 5;
var num2 = 6;
let num3 = 7;
function x(){
  let num3 = 8;
  return "Global variable sum = " + (num + num2 + num3);
}
//
console.log(x())//prints 19
console.log(num3)//prints 7
```

The *let* keyword will allow re-assignment, but variable is constrained to { }

Variables and Declarations

Three ways to declare and use variables:

```
const num = 5;
var num2 = 6;
let num3 = 7;
function x(){
  num = 8;//!!error
  return "Global variable sum = " + (num + num2 + num3);
}
//
console.log(x())
```

The *const* keyword will not allow re-assignment