Javascript ES5

https://en.wikipedia.org/wiki/ECMAScript
https://nodejs.org/en/docs/
https://developer.mozilla.org/en-US/docs/Web/JavaScript

Object Available By Default in Node Application

Global

Reference to the main runtime object available to all node modules/files. Just like "window" object in browser
Try console.log(global); to see what is inside global object

https://nodejs.org/api/globals.html#globals_global
In browsers, the top-level scope is the global scope. This means that within the browser var will define a new global variable. In Node.js this is different. The top-level scope is not the global scope; var inside a Node.js module will be **local to that module**.

Module.Exports/Exports

exports Object are used to reference a module's objects and function in another module/file

```
console.log(module);
console.log(module.exports);
console.log(exports);
console.log(this);
```

Very Important NOTE: In a module/file this === exports === module.exports

Console

Used to access standard output

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

Object and this keyword

```
// ES5 Objects
var myObject = {
  var1: 50,
  function1: function () {
     return this.var1 * 2;
  }
};

myObject.var2 = 100;

myObject.function2 = function(myName) {
  console.log("Hi " + myName);
};

console.log(myObject.var1);
myObject.function1();

console.log(myObject.var2);
myObject.function2("Sheraz");
```

```
=========
50
100
Hi Sheraz
=========
```

Function Class and this keyword

```
var MyClass = function(constructorVar1) {
  var privateVar = "My Private Value. " + constructorVar1;

  var privateFunction = function (a, b) {
     return a + b
  };

  this.publicVar = "My Public Values";

  this.publicFunction = function (a, b) {
     return privateFunction(10, 20) + a + b;
  };

  this.getPrivateVar = function () {
     return privateVar
  };
};

var myClassVar = new MyClass("Sheraz");
```

```
console.log(myClassVar.publicVar);
console.log(myClassVar.getPrivateVar());
console.log(myClassVar.publicFunction(30, 40));
```

```
===========
My Public Values
My Private Value. Sheraz
100
===========
```

Inheritance/Extension/Prototype

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/prototype

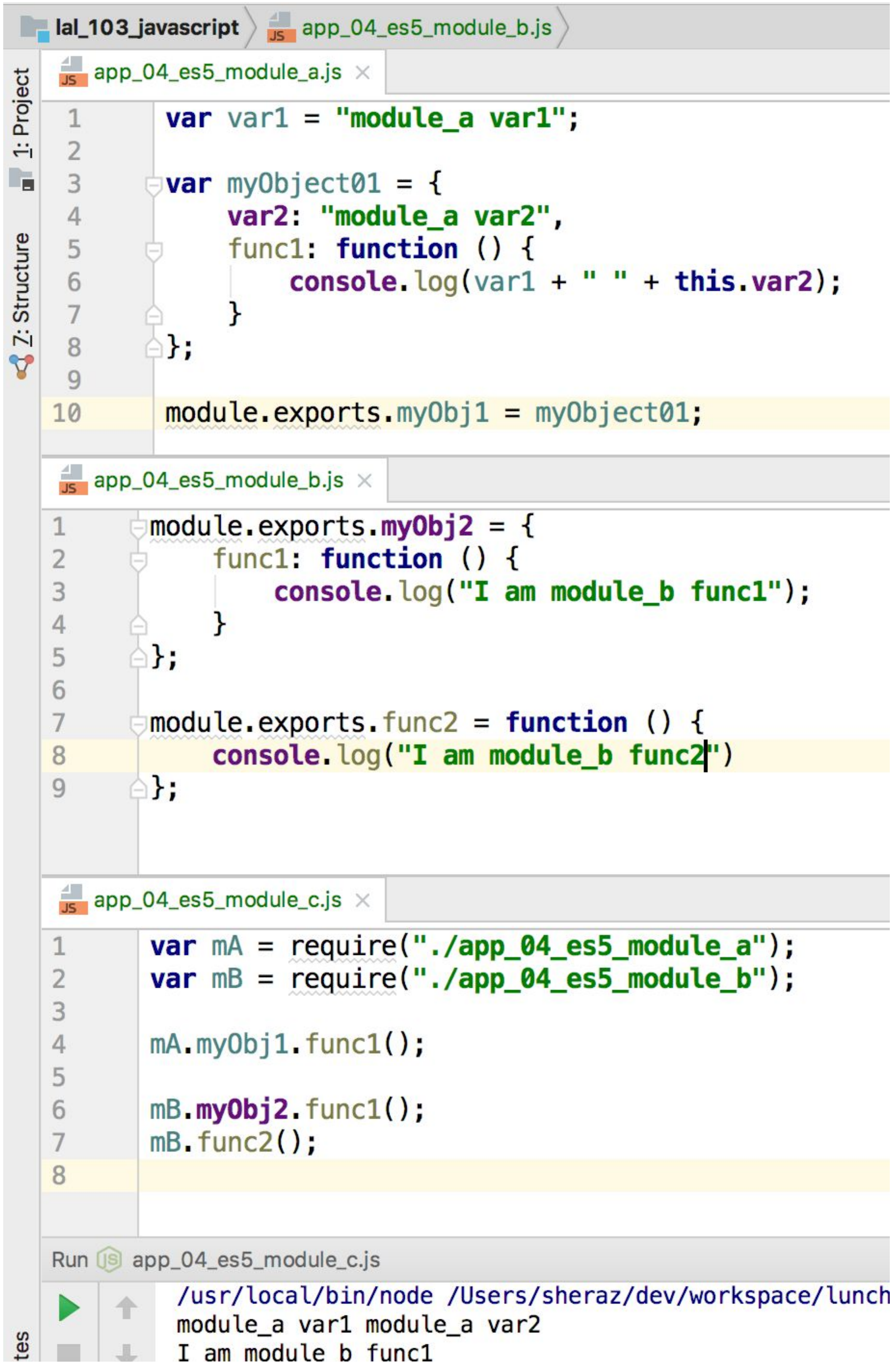
```
var MyMath = function() {
  this.add = function (a, b) {
      return a + b;
  };
};

MyMath.prototype.subtract = function (a, b) {
  return a - b;
};

var myMath = new MyMath();
console.log(myMath.add(20, 5));
console.log(myMath.subtract(20, 5));
```

Modules and require()

Think of module as a javascript file.
Or objects or variable that a javascript exposes.



app_04_es5_module_a.js

```javascript
var var1 = "module_a var1";

var myObject01 = {
    var2: "module_a var2",
    func1: function () {
        console.log(var1 + " " + this.var2);
    }
};

module.exports.myObj1 = myObject01;
```

app_04_es5_module_b.js

```javascript
module.exports.myObj2 = {
    func1: function () {
        console.log("I am module_b func1");
    }
};

module.exports.func2 = function () {
    console.log("I am module_b func2")
};
```

app_04_es5_module_c.js

```javascript
var mA = require("./app_04_es5_module_a");
var mB = require("./app_04_es5_module_b");

mA.myObj1.func1();

mB.myObj2.func1();
mB.func2();
```

Run app_04_es5_module_c.js

```
/usr/local/bin/node /Users/sheraz/dev/workspace/lunch
module_a var1 module_a var2
I am module_b func1
```

Object Factory

Object Factory is a technique used create multiple instances of an Object definition.

```
function myObjectFactory(salary) {
  return {
    name: "PersonA",
    age: 10,
    salary: salary,
    annualSalary: function () {
      return this.salary * 12;
    },
    toString: function() {
      return "name=" + this.name + ", age="
        + this.age + ", salary=" + this.salary
        + ", annualSalary=" + this.annualSalary();
    }
  };
}

var profileA = myObjectFactory(100);
var profileB = myObjectFactory(200);

profileB.name = "Sheraz";

console.log(profileA.toString());
console.log(profileB.toString());
```

```
==============
name=PersonA, age=10, salary=100, annualSalary=1200
name=Sheraz, age=10, salary=200, annualSalary=2400
==============
```

Self Executing Anonymous Function

```
(function() {
  console.log("My Application have successfully started");
})();
```

```
==============
My Application have successfully started
```

==============

Deleting Object Property

```
var myObject = {
  propA: "Prop A val",
  propB: "Prop B val"
};

console.log(myObject);

delete myObject.propB;

console.log(myObject);
```

==============

```
{ propA: 'Prop A val', propB: 'Prop B val' }
{ propA: 'Prop A val' }
```

==============

Function arguments

```
var showArguments = function() {
  // "arguments" is array link object that contains
  // all the arguments passed to a function.
  // It indexes all its values.
  console.log(arguments);

};
showArguments(2,3,4);

var argumentsToArray = function () {
// since "arguments" is not an array but actually an object
// so we have to use different techniques to convert it to an array
// e.g.
  var argumentsArray = [];
  for (var argument in arguments) {
     argumentsArray[argument] = arguments[argument];
  }
  return argumentsArray
};

console.log(argumentsToArray(2,4,5));
```

```
===========
{ '0': 2, '1': 3, '2': 4 }
[ 2, 4, 5 ]
===========
```

Closure

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

Dynamic Lexical Scope

JavaScript function calls have dynamic lexical scope. Which means it's **this** scope changes from where it gets called. To change scope we use these method

- bind()
- call()
- apply()

Bind

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind
Bind function is used to change lexical scope and Override function arguments.

```
this.name = "Sheraz";

function myFunction(greeting) {
   console.log(greeting + " " + this.name);
}

function executeMyFunction(func) {
   this.name = "Muhammad";
   func("AOA");
}

executeMyFunction(myFunction);
executeMyFunction(myFunction.bind(this));
executeMyFunction(myFunction.bind(this, ["Hello"]));
```

```
===========
AOA Muhammad
AOA Sheraz
Hello Sheraz
===========
```

Call

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call
call() function does what bind() do and it also executes the function.
We can also pass **comma separated** function arguments after scope object

```
this.name = "Sheraz";

function myFunction(greeting, lastName) {
   console.log(greeting + " " + this.name + " " + lastName);
}

var myObject = {name: "Tariq"}

myFunction.call();
myFunction.call(myObject);
myFunction.call(this);
myFunction.call(this, ["Hello"]);
myFunction.call(this, "Hello", "Chaudhry");
```

```
===========
undefined undefined undefined
undefined Tariq undefined
undefined Sheraz undefined
Hello Sheraz undefined
Hello Sheraz Chaudhry
===========
```

Apply

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply
apply() function does what bind() do and it also executes the function.
We can also pass **array** of function arguments after scope object

```
this.name = "Sheraz";

function myFunction(greeting, lastName) {
   console.log(greeting + " " + this.name + " " + lastName);
}

var myObject = {name: "Tariq"}

myFunction.apply();
myFunction.apply(myObject);
myFunction.apply(this);
myFunction.apply(this, ["Hello"]);
myFunction.apply(this, ["Hello", "Chaudhry"]);
```

```
===========
undefined undefined undefined
undefined Tariq undefined
undefined Sheraz undefined
Hello Sheraz undefined
Hello Sheraz Chaudhry
===========
```