Front-end development

Lesson 3 / JS basics

Eimantas@adapt.dk

Functional programming

```
function createPerson(name) {
  const obj = {};

  obj.name = name;
  obj.greeting = function () {
    alert('Hi! I am ' + this.name + '.');
  }
  return obj;
}
let newPerson = createPerson('Jeff');
```

THEY CAN BE ASSIGNED TO VARIABLES

```
const f = (m) => console.log(m)
f('Test')
```

Since a function is assignable to a variable, they can be added to objects:

```
const obj = {
  f(m) {
    console.log(m)
  }
}
obj.f('Test')
```

as well as to arrays:

```
const a = [
  m => console.log(m)
]
a[0]('Test')
```

THEY CAN BE USED AS AN ARGUMENT TO OTHER FUNCTIONS

```
const f = (m) => () => console.log(m)
const f2 = (f3) => f3()
f2(f('Test'))
```

THEY CAN BE RETURNED BY FUNCTIONS

```
const createF = () => {
  return (m) => console.log(m)
}
const f = createF()
f('Test')
```

HIGHER ORDER FUNCTIONS

Functions that accept functions as arguments or return functions are called **Higher Order Functions**.

Examples in the JavaScript standard library include `Array.map()`, `Array.filter()` and `Array.reduce()`, which we'll see in a bit.

```
const highpass = cutoff => n => n >= cutoff;

const gt3 = highpass(3);

[1, 2, 3, 4].filter(gt3); // [3, 4];
```

OOP programming

```
var person1 = new Object({
  name: 'Chris',
  age: 38,
  greeting: function() {
    alert('Hi! I\'m ' + this.name + '.');
  }
});
```

2. You can now create a new person by calling this function — try the following lines in your browser's JavaScript console:

```
var salva = createNewPerson('Salva');
salva.name;
salva.greeting();
```

This works well enough, but it is a bit long-winded; if we know we want to create an object, why do we need to explicitly create a new empty object and return it? Fortunately, JavaScript provides us with a handy shortcut, in the form of constructor functions — let's make one now!

3. Replace your previous function with the following:

```
function Person(name) {
  this.name = name;
  this.greeting = function() {
    alert('Hi! I\'m ' + this.name + '.');
  };
}
```

Scope From Functions

```
function foo(a) {
        var b = 2;

        // some code

        function bar() {
                // ...
        }

        // more code

        var c = 3;
}
```

Anonymous vs. Named

```
setTimeout( function(){
        console.log("I waited 1 second!");
}, 1000 );
```

Invoking Function Expressions Immediately

```
var a = 2;

(function foo(){

    var a = 3;
    console.log( a ); // 3

})();


console.log( a ); // 2
```

Closure

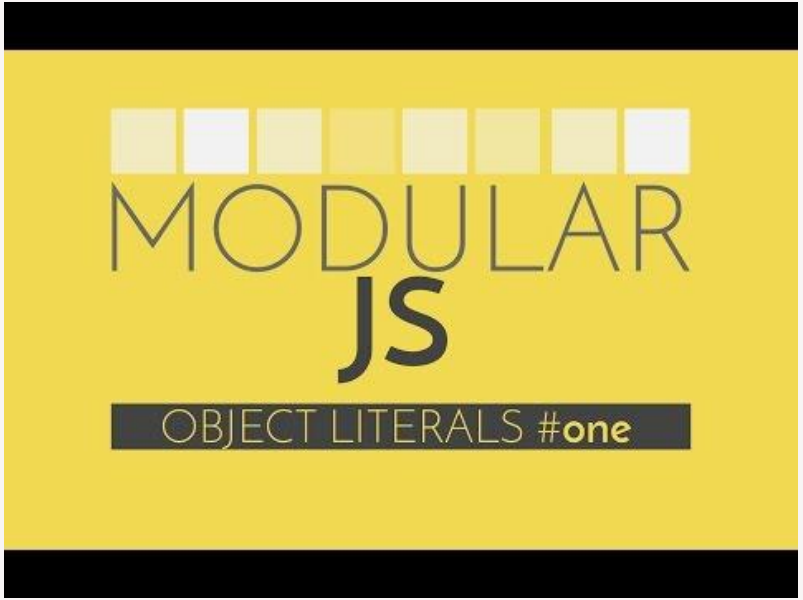
A *closure* is the combination of a function and the lexical environment within which that function was declared. This environment consists of any local variables that were in-scope at the time the closure was created.

```
function makeAdder(x) {
  return function(y) {
    return x + y;
  };
}

var add5 = makeAdder(5);
var add10 = makeAdder(10);

console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

Function as a way to structure your code into smaller pieces

Building a module

```
var jpm = {
    animated: true,
    openMenu: function( ) {
        ...
        this.setMenuStyle( );
    },
    closeMenu: function( ) {
        ...
        this.setMenuStyle( );
    },
    setMenuStyle: function( ) {
... }
};
```



```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

Data types in JS

String

var string = “Home”

```
✗  http://eslint.org/docs/rules/quotes  Strings must use singlequote
src\router\index.js:11:13
      name: "Hello",
            ^
```

var string string = ‘Home’

const dynamicSting = `Hello ${var}!`

To check variable type: typeof
Check with equality with type: ===

var status = 1;
console.log(status === ’1’);

True or False?

Number

const int = 10;

int = 10,1

JavaScript Demo: Standard built-in objects - parseInt()

```
1  function roughScale(x, base) {
2    var parsed = parseInt(x, base);
3    if (isNaN(parsed)) { return 0 }
4    return parsed * 100;
5  }
6
7  console.log(roughScale(' 0xF', 16));
8  // expected output: 1500
9
10 console.log(roughScale('321', 2));
11 // expected output: 0
12
```

Run ›

Reset

```
> 1500
> 0
```

https://jsfiddle.net/8svkenuf/

Access (index into) an Array item

```
var first = fruits[0];
// Apple

var last = fruits[fruits.length - 1];
// Banana
```

Loop over an Array

```
fruits.forEach(function(item, index, array) {
  console.log(item, index);
});
// Apple 0
// Banana 1
```

Add to the end of an Array

```
var newLength = fruits.push('Orange');
// ["Apple", "Banana", "Orange"]
```

Remove from the end of an Array

```
var last = fruits.pop(); // remove Orange (from the end)
// ["Apple", "Banana"];
```

Array

```
let apiJSON = [];

apiJSON.forEach(row => {
  console.log(row);
})
undefined
```

const key = 2;
var arr = [1,2,3];

//console.log last array element

arr = [];
arr.map(function(el){
  console.log(el);
})

Homogeneous Arrays

As the name may suggest a homogeneous array is an array that stores a single data type(string, int or Boolean values).

```
var array = ["Matthew", "Simon", "Luke"];
var array = [27, 24, 30];
var array = [true, false, true];
```

Heterogeneous Arrays

A heterogeneous array is the opposite to a homogeneous array. Meaning it can store mixed data types.

```
var array = ["Matthew", 27, true];
```

Multidimensional Arrays

Also known as an array of arrays, multidimensional arrays allow you to store arrays within arrays, a kind of “array-ception”.

```
var array = [["Matthew", "27"], ["Simon", "24"], ["Luke", "30"]];
```

Jagged Arrays

Jagged arrays are similar to multidimensional array with the exception being that a jagged array does not require a uniform set of data.

Array types

```
var array = [
    ["Matthew", "27", "Developer"],
    ["Simon", "24"],
    ["Luke"]
];
```

https://github.com/adaptdk/Adapt-Academy-Frontend/blob/master/lesson3/README.md

Objects

```
var myCar = new Object();
myCar.make = 'Ford';
myCar.model = 'Mustang';
myCar.year = 1969;
```

```
const object1 = {
  a: 'somestring',
  b: 42,
  c: false
};

console.log(Object.values(object1));
// expected output: Array ["somestring", 42, false]
```

```
Object.assign()
Object.create()
Object.defineProperties()
Object.defineProperty()
Object.entries()
Object.freeze()
Object.fromEntries()
Object.getNotifier()
Object.getOwnPropertyDescriptor()
Object.getOwnPropertyDescriptors()
Object.getOwnPropertyNames()
Object.getOwnPropertySymbols()
Object.getPrototypeOf()
Object.is()
Object.isExtensible()
Object.isFrozen()
Object.isSealed()
Object.keys()
```

Loops

For loops

```
var str = "";
for (var i = 0; i < 9; i++) {
  str = str + i;
}
console.log(str);
// expected output: "012345678"
```

do .. while

```
var i = 0;
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

while

```
var n = 0;
var x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

labeled statement

```
markLoop:
while (theMark == true) {
   doSomething();
}
```

for...of statement

```
var arr = [3, 5, 7];
arr.foo = 'hello';

for (var i in arr) {
   console.log(i); // logs "0", "1", "2", "foo"
}

for (var i of arr) {
   console.log(i); // logs 3, 5, 7
}
```

https://hackernoon.com/rethinking-javascript-death-of-the-for-loop-c431564c84a8

If statements

```
if (x > 5) {
 /* do the right thing */
} else if (x > 50) {
 /* do the right thing */
} else {
 /* do the right thing */
}
```

```
x === 1 ? false : true;
```

Switch statements

```
switch (fruittype) {
  case 'Oranges':
    console.log('Oranges are $0.59 a pound.');
    break;
  case 'Cherries':
  case 'Blueberies':
    console.log('Cherries are $3.00 a pound.');
    break;
  default:
   console.log('Sorry, we are out of ' + fruittype + '.');
}
```

Data types and data structures

Dynamic typing

JavaScript is a *loosely typed* or a *dynamic* language. Variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned (and re-assigned) values of all types:

```
var foo = 42;    // foo is now a number
foo     = 'bar'; // foo is now a string
foo     = true;  // foo is now a boolean
```

Data types

The latest ECMAScript standard defines seven data types:

- Six data types that are primitives:
  - Boolean
  - Null
  - Undefined
  - Number
  - String
  - Symbol (new in ECMAScript 6)
- and Object

```
console.log(null); // null

var name = null;
console.log(name); // null
```

While they may seem similar, it's important to understand the difference between `null` and `undefined`. In basic terms, `undefined` means that a variable has been `declared` but has not yet been assigned a value. Moreover, `null` and `undefined` are different `types`: `null` is actually an object whereas `undefined` is a type unto itself:

```
console.log(typeof(null)); // object
console.log(typeof(undefined)); // undefined
```

We can also compare the similarity and differences of `undefined` and `null` by checking them using equality (`==`) and identity (`===`) operators:

```
console.log(null == null); // true
console.log(null === null); // true

console.log(undefined == undefined); // true
console.log(undefined === undefined); // true

// Check equality.
console.log(null == undefined); // true
// Check identity.
console.log(null === undefined); // false
```

Assignments

Everything in JavaScript acts like an object, with the only two exceptions being `null` and `undefined`.

```
false.toString(); // 'false'

[1, 2, 3].toString(); // '1,2,3'

function Foo(){}

Foo.bar = 1;

Foo.bar; // 1
```

Accessing Properties

The properties of an object can be accessed in two ways, via either the dot notation or the square bracket notation.

```
var foo = {name: 'kitten'}

foo.name; // kitten

foo['name']; // kitten

var get = 'name';

foo[get]; // kitten

foo.1234; // SyntaxError

foo['1234']; // works
```

What this one will print?

Array(16).join("lol" - 2)

Array(16).join("lol" - 2) + " Batman!";

```
Array(16).join("lol" - 2) + " Batman!";
"NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!"
```

Horoscope task

https://github.com/adaptdk/Adapt-Academy-Frontend/blob/master/lesson3/README.md

promises

```
/_ ES5 _/

...


// call our promise
var askMom = function () {
    willIGetNewPhone
        .then(function (fulfilled) {
            // yay, you got a new phone
            console.log(fulfilled);
         // output: { brand: 'Samsung', color: 'black' }
        })
        .catch(function (error) {
            // oops, mom don't buy it
            console.log(error.message);
         // output: 'mom is not happy'
        });
};


askMom();
```

https://jsbin.com/nifocu/1/edit?js,console

promises

How do I access the data in a promise? I use `.then()`:

```
function getFirstUser() {
    return getUsers().then(function(users) {
        return users[0].name;
    });
}
```

How do I catch the errors from a promise chain? I use `.catch()`

```
function getFirstUser() {
    return getUsers().then(function(users) {
        return users[0].name;
    }).catch(function(err) {
        return {
          name: 'default user'
        };
    });
}
```

```
async function getFirstUser() {
    let users = await getUsers();
    return users[0].name;
}
```

```
async function getFirstUser() {
    try {
        let users = await getUsers();
        return users[0].name;
    } catch (err) {
        return {
            name: 'default user'
        };
    }
}
```

```
let user = await getFirstUser();
```

Functions for dom manipulation

```
document.getElementById('textbox_id').value
```

```
    document.getElementById("demo").innerHTML = "Paragraph changed!";
```

```
<select id="mySelect" onchange="myFunction()">
  <option value="Audi">Audi
  <option value="BMW">BMW
  <option value="Mercedes">Mercedes
  <option value="Volvo">Volvo
</select>


<script>
function myFunction() {
    var x = document.getElementById("mySelect").value;
    document.getElementById("demo").innerHTML = "You selected: " + x;
}
</script>

</body>
```

Ačiū

https://www.codecademy.com/learn/introduction-to-javascript