JavaScript, Just-In-Time compilation (JIT) and WebAssembly (Wasm)

Adam Boult (www.bou.lt)

March 23, 2024

Contents

Part I

Javascript basics

Chapter 1

Numerical variables

1.1 Introduction

1.1.1 Variable assignment and data types

```
x = 1;
```

undefined

null

boolean

”number”, other sort of number?

1.1.2 ”var” keyword and strict

Following will run, but the typo no noticed.

```
myVariable = 1;
myVaraible = myVariable + 1;
```

We can enforce declaration of variables to help prevent this.

```
"use strict"
var x = 1;
x = 1;
```

1.1.3 ”let” keyword and block scope

This works, even though we may not expect it to.

```
{
    var x = 1;
}
```

```
var y = x;
```

Block requires "let"

```
{
    let x = 1;
}
var y = x;
```

"let" is a newer keyword than "var". Broadly "use strict" and "let" should be used throughout.

1.1.4 "const" keyword

Like let (uses block scope) but can't reassign. This increases safety and can make compiler work better.

1.1.5 Dynamic typing

Dynamically typed. eg can add different types of numerical stuff?

1.1.6 Relations

```
a==b
a!=b
```

returns boolean

if different types, converts before comparing

```
"1" == 1 // evaluates to true
```

If use 3 equals signs, no type conversion.

```
"1" === 1 evaluates to false
```

Chapter 2

Control flow

2.1 Introduction

2.1.1 Introduction

```
let x = 0;
for (let i = 0; i < 10; i++) {
    x = x + i;
}
```

2.1.2 Iterating over arrays

```
let x = 0;
for (const e of arr) {
    x = x + e;
}
```

Chapter 3

Functions

3.1 Introduction

3.1.1 Function declaration

Variables created without a declaration keyword (var, let, or const) are always global, even if they are created inside a function.

```
function myFunction(a, b) {
  return a * b;
}
myFunction(10, 2);
```

3.1.2 Function scope

We might expect the following to fail, but it doesn't.

```
function myFunction() {
    x = 5;
}
myFunction();
var y = x;
```

The following does fail, as x has not been declared.

```
"use strict";
function myFunction() {
    x = 5;
}
myFunction();
var y = x;
```

As does the following, because declaring the variable keeps it in the scope of the function.

```
"use strict";
function myFunction() {
    var x = 5;
}
myFunction();
var y = x;
```

3.1.3 Function expressions

The function here is an anonymous function. We just happen to be naming it.

```
const myFunction = function(a, b) {
    return a * b
};
```

Function expressions are not necessarily in the global score, and so can be preferred.

3.1.4 Arrow function expressions

Another way of writing function expressions.

```
const myFunction = (a, b) => a * b;
```

Again this is an anonymous function, that has happened to have been named. The function itself can be used as a parameter in eg a map.

3.1.5 Callbacks

pass function as argument

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}
function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}
myCalculator(5, 5, myDisplayer);
```

3.1.6 Sort

dynamically typed functions

Chapter 4

Strings

4.1 Introduction

4.1.1 Introduction

`text.length;`

iterating over strings

matching text in strings

regex

accessing ith element

Chapter 5

Objects and prototypes

5.1 Introduction

5.1.1 Introduction

iterator stuff: + ”yield” in functions + generators + ”next”

5.1.2 Object literals

can define object literals

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

```
const person = {};
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

5.1.3 Access

```
objectName.property
objectName["property"]
```

can iterate over properties in object for (let variable in object) // code to be executed

Object.getOwnPropertyNames()

Object.values(myObj); gets values

5.1.4 Deleting parts of object

can delete

```
delete person.age;
delete person["age"];
```

5.1.5 Nested objects

can nest objects

```
myObj = {
  name:"John",
  age:30,
  cars: {
    car1:"Ford",
    car2:"BMW",
    car3:"Fiat"
  }
}
```

Can access nested parts of objects.

```
myObj.cars.car2;
myObj["cars"]["car2"];
```

5.1.6 Printing objects

`JSON.stringify(myObj); can print`

5.1.7 Getters and settings

getters and setters for object. can access and update values in a more flexible way.

```
const person = {
  firstName: "John",
  lastName: "Doe",
  language: "en",
  get lang() {
    return this.language.toUpperCase();
  }
};
person.lang; will retrun En
```

const person = firstName: "John", lastName: "Doe", language: "", set lang(lang) this.language = lang.toUpperCase(); ;

```
person.lang = "en"; will make language be "En"
```

methods in object literals

```
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
```

constructors to make objects

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

primatives can be done as objects

```
new String()    // A new String object
new Number()    // A new Number object
new Boolean()   // A new Boolean object
new Object()    // A new Object object
new Array()     // A new Array object
new RegExp()    // A new RegExp object
new Function()  // A new Function object
new Date()      // A new Date object
```

but primatives faster than doing this.

can add things to constructors using prototype

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}
Person.prototype.nationality = "English";

const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

making an object iterable: @@iterator method use of

5.1.8 thing?

is generator stuff?

```
function* my_function(){
```

arrays and sets

+ JavaScript this

5.2 Inheritance

5.2.1 Inheritance

5.2.2 Multiple inheritance

Chapter 6

Concurrency

6.1 Introduction

6.1.1 Introduction

Part II

Javascript for clients

Chapter 7

Manipulating the DOM with Javascript

7.1 Introduction

7.1.1 Introduction

make element

```
const myElem = document.createElement('span');
```

Attributes like classes and the id can be set as well

```
myElem.classList.add('foo');
myElem.id = 'bar';
myElem.dataset.attr = 'baz';
```

Can attach to the body

```
document.body.appendChild(myElem);
```

After setting the tag will look like this:

```
<span class="foo" id="bar" data-attr="baz"></span>
```

changing existing elements

```
document.querySelector('.class'); // get first element with class of "class"
document.querySelector('#id'); // select the first element with the id of id
document.querySelector('[data-other]'); // select the first element with the data-other attr
document.querySelectorAll('.multiple'); // return array of all element with the 'multiple' c
```

Chapter 8

Asynchronous JavaScript and XML (Ajax)

8.1 Introduction

8.1.1 Introduction

How javascript is done in the browser asynchronous in the browser

can use xml or json to send data

http requests in js. post. get

built in js functions in browser

`XMLHttpRequest();`

Chapter 9

Sandboxing/same origin policy and cross-site scripting

9.1 Introduction

9.1.1 Introduction

Part III

Later

Chapter 10

Typescript

10.1 Introduction

10.1.1 Introduction