Note Web Application

Brendon Mendicino

March 2, 2023

Contents

1   Introduction

is backward compapatible, to be able to use the previous features is use the directive:

```
"use strict";
```

JS has primitive types and non-primitive types, JS is also and strongly typed language, the primitive types are: string, number, boolean, null, undefined. The non-primitive are the objects, which can be: array, function, user-defined.

The all possible false values in JS: `0, -0, NaN, undefined, null,''`, in JS there are two main comparison operators:

```
a == b     // equal, convert types and compare
a === b    // strict equal, inhibits automatic type conversion
```

In JS you can create variable with:

```
// modern
let a = 10;     // can be changed
const b = 'a'; // constant

// old
var k = 9;
j = 30;
```

The difference between null and undefined, is that variable with null they old a value which is null, on the other way if a variable is declared and nothing is associated with it the value olds by default undefined.

A scope is defined by a **block**, which is created with   . . .

There two kinds of `foreach` in JS, using `in` allows iterating over objects, while `of` allows iterating over iterable objects:

```
for (let a in object) {
   ...
}

for (let b of iterable) {
   ...
}
```

Using arrays:

```
let a = [1, 2, 'ok', false];
let b = Array.of(1, 2, true);
a.push(5);        // append an element
b.unshift(2);    // insert at the beginning

let copy = Array.from(a);  // shallow copy, it does not deep copy
```

The **destructuring assignment** can be done, it extracts the values from the mast left-hand side:

```
let [x, y] = [1, 2];
[x, y] = [y, x]      // swap
```

The **spread operator** $(\ldots)$ expands on iterable object into it's values:

```
let [x, ...y] = [1, 2, 3, 4];          // y == [2, 3, 4]

const a = [1, 2];
const b = [0, ...a, 3]; // [0, 1, 2, 3]
```

Spreding can be from the left or from the right, usually the spread operator is used for copying array:

```
const a = [1, 2];
const b = [...a];
```

A **string** is JS is an immutable type (like python) encoded in Unicode. The **template literals** can be done with the **tick** operator `‘ (expression like Kotlin):

```
let name = 'Bre';
let sur = 'Mend';
// Template literal
let fullName = '${name} ${sur}';
```

1.1   Objects

JS is **prototype based language**, which means that there are no declarations of classes. In JS property names must be strings and can be modified, the value of the property can be any other type of type or object. To create and object in JS you use curly braces and the defined properties:

```
const movie = {
  title: 'Inception',
  genre: 'sci-fi',
  duration: 180
}

console.log(movie)
console.log(movie['title'])
console.log(movie.title)
```

It is also possible to add a property by simple assigning a new name to a type, it is also possible to delete a property with the keyword `delete`. There are two helper functions:

- `Object.key(object)`: return only the key;

- `Object.entries(object)`: return an array with the key and value;

To copy an object it is possible to use:

```
const copied = Object.assign({}, original)
const withSpread = {...original}    // it also possbible to use the
    spread operator

```

```
4 // assign can also be used to merge objects
5 const merged = Object.assign({}, copied, {something: 'test'})
```

1.2   Functions

In JS functions are objects, so it is possible to assign a function to a property or use it in a parameter in another function. There three possible ways to define a function:

```
1  // 1. Function
2  function do(a, b = 1) {
3    ...
4  }
5
6  // parameters can also hava a deafult value
7  function some(par1, par2, ...variable) { // ... is the 'rest'
      operator, like varargs, rest parameters can be iterated
8    ...
9  }
10
11 // 2. Function Expression
12 const fn = function(params) { }
13
14 // 3. Arrow Function
15 const func = (params) => { }
```

In JS **Closure** can be created, with closure it is possible to use parameters of the scope where the function is defined, even if that scope does not exist any more.

```
1  function greeter(name) {
2    const myname = name;
3
4    const hello = () => {
5      return "Hello " + myname;
6    }
7
8    return hello;
9  }
10
11 const helloTest = greeter('test');
12
13 console.log(helloTest());    // 'Hello test'
```

To create an object there are also **constructor functions**:

```
1  function Movie(title, director, duration) {
2    this.title = title;
3    this.director = director;
4    this.duration = duration;
5    this.isLong = () => this.duration > 120;
6  }
7
```

```
8 cosnt movie = new Movie('Inception', 'Nolan', 180);
9 console.log(movie.isLong);  // true
```

test