Elements of modern JavaScript

Alessio Vecchio

University of Pisa

for of

- Introduced in ES6
- It can be used with iterable objects
- Iterable objects: strings, arrays, sets, maps

```
let mya = [11, 2, 3, 55, 4, 12, 32];

for(let x of mya) {
  console.log(x);
}
```

```
11
2
3
55
4
12
32
```

for of

- Objects are generally not iterable, generates *TypeError* at runtime
- You can iterate through the properties of an object using ***for in*** or obtain an array of properties and then use *for of*

```
let o1 = {"a": "one",
          "b": "two",
           c: 3};

for(let x of Object.keys(o1)) {
  console.log(x + ": " + o1[x]);
}
```

```
a: one
b: two
c: 3
```

Object.keys() returns an array with the property names

for of

- Same for object values. *Object.values()* returns the properties' values.

```
for(let x of Object.values(o1)) {
  console.log(typeof x + " " + x);
}
```

```
string one
string two
number 3
```

- or for key-value pairs

```
for(let [kk, vv] of Object.entries(o1)) {
  console.log(kk + ": " + vv);
}
```

```
a: one
b: two
c: 3
```

Object.entries() returns an array of arrays. Each internal array is a key-value pair.

for of

- An example with a string

```
let q = {};
for(let c of "Progettazione Web") {
  if(q[c]) q[c]++;
  else q[c] = 1;
}
console.log(q);
```

```
{
  P: 1,
  r: 1,
  o: 2,
  g: 1,
  e: 3,
  t: 2,
  a: 1,
  z: 1,
  i: 1,
  n: 1,
  ' ': 1,
  W: 1,
  b: 1
}
```

Destructuring assignment

- Introduced in ES6
- Syntax:

  one or more variables = a compound object, generally an array

```
let [a, b, c] = [10, 20, 30];
console.log(a + " " + b + " " + c);
// Output:
// 10 20 30
```

```
// Exchanges the value of a, b
[a, b] = [b, a];
console.log(a + " " + b);
// Output:
// 20 10
```

Destructuring assignment

```
// Not all elements must be used
[a, b] = ['ABC', 'DEF', 'GHI'];
console.log(a + " " + b + " " + c);
// Output:
// ABC DEF 30

// If too many variables, last ones are undefined
let [x, y, z] = [1, 2];
console.log(x + " " + y + " " + z);
//Output:
// 1 2 undefined

// Some values can be skipped using commas
[,x,,y,,z] = ['a', 'b', 'c', 'd', 'e', 'f'];
console.log(x + " " + y + " " + z);
// Output:
// b d f
```

Destructuring assignment

```
// Can be used with functions
function myfun(){
  let p = 100, q = 200;
  // Some code
  return [p, q];
}
[x,y] = myfun();
console.log(x + " " + y);
// Output:
// 100 200

// The set of remaining values can //
be collected in a single one
[x, ...y] = [10, 20, 30, 40, 50, 60];
console.log("x=" + x + ", y=" + y);
// Output:
// x=10 y=20,30,40,50,60
```

```
// Works also with objects
// but less used
let o1 = {k1: 42, k2: 24};
let {k1, mk2} = o1;
console.log(k1 + " " + mk2);
// Output:
// 42 undefined
```

Template string literals

- Introduced in ES6
- Delimited by **backticks**
- Everything within **${** and **}** is considered as a JS expression

```
let topic = "Progettazione Web";
let s1 = `My favourite topic is ${topic}`;
console.log(s1);
// Output:
// My favourite topic is
// Progettazione Web

let x = "X";
let y = 42;
let s2 = `Some text, ${x} and ${y*2}`;
console.log(s2);
// Output:
// Some text, X and 84
```

Default parameters

- Introduced in ES6
- Arguments without a value are generally initialized to undefined
- It is now possible to have default values that are used when a value is not provided during call
- Parameters with default values must be at the end of the list

```
function f1(a1, a2 = 10, a3 = 'X'){
  // Some code, here just print
  console.log(a1 + " " + a2 + " " + a3);
}

f1(1, 2, 'Z');
// Output:
// 1 2 Z
f1(1, 2);
// Output:
// 1 2 X
f1(1);
// Output:
// 1 10 X
```

Classes

- Introduced in ES6
- Classes can be defined accrding to a style that is similar to the one of other OO programming languages
- Classes can be declared using the **class** keyword
- The class body **{ }** contains the definition of the constructor and methods
- constructor can be omitted if not needed
- In methods the *function* keyword is not used

Classes

```
class Student {
  constructor(n, d) {
    this.name = n;
    this.degree = d;
    this.marks = [];
  }
  toString() {
    return `${this.name}, ${this.degree}`;
  }
  addMarks(m) {
    this.marks.push(m);
  }
  getAverage(){
    let s = 0;
    for(let m of this.marks) {
      s += m;
    }
    return s/this.marks.length;
  }
}
```

Classes

```
let s1 = new Student("Mario",
                     "Ing. Informatica");
console.log("s1=" + s1);

let s2 = new Student("Luigi",
                     "Computer Engineering");
console.log("s2=" + s2);

console.log(s1.getAverage());
s1.addMarks(27);
s1.addMarks(28);
console.log(s1.getAverage());
```

```
// Output:
s1=Mario, Ing. Informatica
s2=Luigi, Computer Engineering
NaN
27.5
```

Classes

- A class must be declared before being used
- Syntactic sugar, underlying model still the same
- Single inheritance is supported
- super to call superclass constructor and methods

```
class WorkingStudent extends Student {
  constructor(n, d, c, f=true) {
    super(n, d);
    this.company = c;
    this.fulltime = f;
  }
  toString(){
    let s = super.toString();
    return s +
       `, company=${this.company}, fulltime=${this.fulltime}`;
  }
  applyBonus(b){
    if(this.fulltime)
      for(let i=0; i<this.marks.length; i++)
        this.marks[i] = this.marks[i]*b;
  }
}
```

Classes

```
let w1 = new WorkingStudent("Gino",
                            "Ing. Informatica", "Google");
w1.addMarks(30);
w1.applyBonus(1.05);
console.log("w1=" + w1);
console.log(w1.getAverage());

// Output:
// w1=Gino, Ing. Informatica, company=Google, fulltime=true
// 31.5
```

Strict mode

- Introduced in ES5
- A directive that is applied by inserting the special **"use strict";** string expression statement at the beginning of a file
- Code in that file is executed in strict mode
  - all variables must be declared
  - *with* can't be used
  - duplicate parameters generate error
  - additional checks
- Classes are automatically in strict mode

Strict mode

```
"use strict";

x = 21;
// x = 21;
// ^
// ReferenceError: x is not defined

o1 = {p1: 10, p2: 20};
with(o1) {
  console.log(p1);
  console.log(p2);
}
// with(o1) {
// ^^^^
//
// SyntaxError: Strict mode code may not include a with statement

function f(a1, a1) {
  console.log(a1);
}
f(10, 20);
// SyntaxError: Duplicate parameter name not allowed in this context
```

Arrow functions

- Introduced in ES6
- Syntax: **parameters => body of the function**
- The *function* keyword is not used
- Useful when you have to pass a function with limited complexity

```
// No arrow function yet
const f1 = function (a, b) {
  return a + b;
}
console.log(f1(10, 20));
// Output:
// 30

// Now arrow function
const f2 = (a, b) => {return a
+ b};
console.log(f2(10, 20));
// Output:
// 30
```

Arrow functions

```
// If the body just a return statement then
// return keyword and {} can be omitted
const f3 = (x, y, z) => x+y+z;
let r1 = f3(5, 6, 7);
console.log(r1);
// Output:
// 18
```

```
// If a single parameter can omit ()
const square = x => x*x;
console.log(square(11));
// Output:
// 121
```

```
// If no parameters, still have to use ()
const f4 = () => {return Math.random()*10};
let r2 = f4();
console.log(r2);
// Output:
// 7.4960047952343
```

Arrow functions

- Passing as argument

```
let ar1 = [10, 20, 30];
ar1.forEach(function (e) {
            let t = Math.random() * e;
            console.log(t);
           });
// Output:
// 2.880299333025 46
// 5.757679342468269
// 27.80074927780763

// with arrow function
ar1.forEach((e) => {
            let t = Math.random()*e;
            console.log(t);
            });
// Output:
// 9.58533802031983
// 8.773334995700527
// 12.840542468303616
```

Spread operator

- The spread operator ... can be used with arrays (ES6)

```
let a1 = [10, 20, 30, 50];
// a1 is spread and its elements become elements of a2
let a2 = [1, ...a1, 100];
console.log(a2, "\n");

// can be used to create a shallow copy
let v = ['first', 'second', 'third'];
let c = [...v];
console.log(c, "\n");

// the copy is shallow
let x1 = [{AAA: 1, BBB: 2}, {CCC: 3, DDD: 4}];
let x2 = [...x1];
x1[0].AAA = 999;
console.log(x2, "\n");

// strings can be spread into an array of strings
let s = "This is a sentence";
let charactersOfS = [...s];
console.log(charactersOfS, "\n");
```

```
[ 1, 10, 20, 30, 50, 100 ]


[ 'first', 'second',
'third' ]


[ { AAA: 999, BBB: 2 },
  { CCC: 3, DDD: 4 } ]


[
  'T', 'h', 'i', 's', ' ',
  'i', 's', ' ', 'a', ' ',
  's', 'e', 'n', 't', 'e',
  'n', 'c', 'e'
]
```

Spread operator

- can be used with objects as well (ES2018)

```
// objects can be spread as well
let stud1 = {name: "Mario", averageGrade: 25};
let r1 = {...stud1, course: "Programmazione Web"};
console.log(r1, "\n");

// if property name is the same, last one wins
let r2 = {...stud1, name: "Gino"};
console.log(r2, "\n");

// inherited properties are not spread
let p1 = {myproperty: "ABC"};
let p2 = Object.create(p1);
console.log(p2.myproperty);
let p3 = {...p2};
console.log(p3.myproperty);
```

```
{ name: 'Mario', averageGrade: 25,
course: 'Programmazione Web' }


{ name: 'Gino', averageGrade: 25 }


ABC
undefined
```