# I. Programmation javascript es5

## Table des matières

I. Programmation javascript es5....1
1. Types de données primitifs et Objets....2
Number....2
Booléen....2
Date....3
infini....4
Tableaux....4
String....6
Expression régulière....7
2. Opérateurs....8
3. Instructions....9
4. Fonctions....9
Traitement Asynchrone....10
Générateur....11
Exceptions....12
5. Programmation Orientée Objets....13
Classes....14
Méthodes et données static....14
Héritage et redéfinition....15

## 1. Types de données primitifs et Objets

- nombres : Number
- chaînes de caractères : String
- booléens : Boolean
- fonctions : Function
- objets : Object (Function, Array, Date)
- symbo: Symbol
- Valeurs spéciale : null, undefined

### *Number*

Objet : Number

```
var n = new Number(11)
console.log(n)
console.log(typeof n)
var n = new Number(11.11)
console.log(n)
console.log(typeof n)
```

Affiche

```
[Number: 11]
object
[Number: 11.11]
object
```

Litéral : number

```
var n = 11.11
console.log(n)
console.log(typeof n)
```

Affiche

```
11.11
number
```

### *Booléen*

Vrai

```
var n = new Boolean(true)
console.log(n)
console.log(typeof n)
var n = Boolean("true")
console.log(n)
```

```
console.log(typeof n)
var n = new Boolean("true")
console.log(n)
console.log(typeof n)
```

Faux

```
var n = Boolean(0)
console.log(n)
console.log(typeof n)
var n = new Boolean(null)
console.log(n)
console.log(typeof n)
```

Méthodes communes

- X.prototype.toSource()
- X.prototype.toString()
- X.prototype.valueOf()

```
var n = new Boolean(1)
console.log(n.valueOf())
console.log(typeof n.valueOf())
console.log(n.toString())
console.log(typeof n.toString())
```

Affiche

```
true
boolean
true
string
```

### *Date*

Méthodes :
getDate()
getDay()
getFullYear()
getMonth()
getHours()
getMinutes()
getSeconds()
getMilliseconds()
getTime()

```
var n = new Date('January 17, 2018 1:24:00');
console.log(n)
console.log(typeof n)
console.log(n.toString())
```

```
console.log(typeof n.toString())
```

Affiche

```
2018-01-17T00:24:00.000Z
object
Wed Jan 17 2018 01:24:00 GMT+0100 (Paris, Madrid)
string
```

```
console.log(n.getDate())
console.log(n.getDay())
console.log(n.getFullYear())
console.log(n.getMonth())
console.log(n.getHours())
console.log(n.getMinutes())
console.log(n.getSeconds())
console.log(n.getTime())
```

Affiche

```
17
3
2018
0
1
24
0
1516148640000
```

### *infini*

```
var v = 1/-0;
console.log(v);
var v = 1/+  0;
console.log(v);
```

Affiche

```
-Infinity
Infinity
```

### *Tableaux*

Quelques méthodes :

- push()
- pop()
- shift()
- unshift()
- indexOf()
- find()

- join()
- slice()
- sort()
- reverse()

Array

```
var n = ["AA","BB","CC"]
console.log(n)
console.log(n[0])
console.log(typeof n)
n.forEach(function (item, index, array){
      console.log(item, index, array)
})
```

Affiche

```
[ 'AA', 'BB', 'CC' ]
AA
object
AA 0 [ 'AA', 'BB', 'CC' ]
BB 1 [ 'AA', 'BB', 'CC' ]
CC 2 [ 'AA', 'BB', 'CC' ]
```

```
var v=n.pop()
console.log (v)
console.log (n)

var v=n.shift()
console.log (v)
console.log (n)

n.unshift("ZZ")
console.log (n)

console.log(n.indexOf("BB"))
```

Affiche

```
[ 'AA', 'BB', 'CC' ]
AA
object
AA 0 [ 'AA', 'BB', 'CC' ]
BB 1 [ 'AA', 'BB', 'CC' ]
CC 2 [ 'AA', 'BB', 'CC' ]
CC
[ 'AA', 'BB' ]
AA
```

```
[ 'BB' ]
[ 'ZZ', 'BB' ]
1
```

### *String*

Fonctions classiques

```
var s = "ZAZERTYA"
console.log(s)
console.log(typeof s)
console.log(s[0])
console.log('length', s.length)
console.log('charAt', s.charAt(0))
console.log('substring',s.substring(2))
console.log('substr',s.substr(2))
console.log('indexOf',s.indexOf("A",2))
console.log('slice',s.slice(2,5))
console.log('repeat',s.repeat(3))
console.log('concat',s.concat("azerty"))
console.log('toLowerCase',s.toLowerCase())
console.log('trim',s.trim())
console.log('startsWith',s.startsWith("Z"))
```

Affiche

```
ZAZERTYA
string
Z
length 8
charAt Z
substring ZERTYA
substr ZERTYA
indexOf 7
slice ZER
repeat ZAZERTYAZAZERTYAZAZERTYA
concat ZAZERTYAazerty
toLowerCase zazertya
trim ZAZERTYA
startsWith true
```

Opérations

```
var s1 = "AAAA"
var s2 = "BBBB"
var s3 = "AAAA"
var s4 = s1+s2
console.log('>',s1 > s2)
```

```
console.log('>',s1 < s2)
console.log('>',s1 == s3)
console.log('>',s1 === s3)
console.log('+',s4)
console.log('+',s1+=s2)
```

Affiche

```
> false
> true
> true
> true
+ AAAABBBB
+ AAAABBBB
```

Expression regulière

```
console.log('match',s.match(/^Z/))
console.log('search',s.search(/A/))
console.log('split',s.split(/A/))
```

Affiche

```
match [ 'Z', index: 0, input: 'ZAZERTYA' ]
search 1
split [ 'Z', 'ZERTY', '' ]
```

html

```
console.log(s.anchor())
console.log(s.bold())
console.log(s.link("chaine"))
```

Affiche

```
<a name="undefined">ZAZERTYA</a>
<b>ZAZERTYA</b>
<a href="chaine">ZAZERTYA</a>
```

Relation entre String et string

```
var s2 = new String("ZAZERTYA")
console.log(typeof s2)
console.log(typeof s2.valueOf())
```

Affiche

```
object
string
```

### *Expression régulière*

- match
- split

- replace

match

```
var exp = '^[AB]'
// var exp = /[AB]/
//var exp = new RegExp ('^[AB]')
console.log("AZERTY".match(exp))
```

Affiche

```
[ 'A', index: 0, input: 'AZERTY' ]
```

Sous expression : replace

```
var exp = /(.*)-(.*)/
console.log("AAA-BBB")
console.log("AAA-BBB".replace(exp, '$2**$1'))
```

Affiche

```
AAA-BBB
BBB**AAA
```

Split

```
var exp = /^(.*)-(.*)$/
console.log("AAA-BBB")
console.log("AAA-BBB".split(exp))
```

Affiche

```
AAA-BBB
[ '', 'AAA', 'BBB', '' ]
```

## 2. Opérateurs

| | |
|---|---|
| ET binaire | a & b |
| OU binaire | a \| b |
| OU exclusif binaire (XOR) | a ^ b |
| NON binaire | ~ a |
| Décalage à gauche | a << b |
| Décalage à droite avec propagation du signe | a >> b |
| Décalage à droite avec introduction de zéros | a >>> b |

opérateur in avec un dictionnaire ou Object

```
let t = { "A":"AA","B":"BB","C":"CC"}
console.log ("A" in t)
```

Affiche

```
true
```

## 3. Instructions

Do/while

```
var i=0;
do
{
    console.log("Je compte", i)
    i++;
} while(i<5)
```

Affiche

```
Je compte 0
Je compte 1
Je compte 2
Je compte 3
Je compte 4
```

switch

```
var i = 10;
switch (i)
{
    case 1:
            console.log("Unité")
            break;
    case 10:
            console.log("Dixaine")
            break;
    default :
            console.log("Trop grand")
}
```

## 4. Fonctions

Trois facon de créer une fonction

```
function somme (i,j)
{
            return  i + j;
}
console.log(somme(2, 6));
var somme = new Function('i', 'j', 'return i + j');
console.log(somme(2, 6));
var somme = (i,j)=>{ return i + j};
console.log(somme(2, 6));
```

Afficher

```
8
8
8
```

Appeler autrement

```
console.log(somme.name)
console.log(somme.length)
console.log(somme.call(this,10,20))
console.log(somme.apply(this,[10,20]))
```

Affiche

```
somme
2
30
30
```

### *Traitement Asynchrone*

```
var promise1 = new Promise(function(resolve, reject) {
  setTimeout(function () {resolve(1234)}, 2000, '');
});

promise1.then(
      function (v) {
            console.log(v)
      }
      ).catch(
      function (error) {
        console.log(error.message);
       // output: 'mom is not happy'
});
```

await + async

```
function montraitement (x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}

async function f1() {
  var x = await montraitement(10);
  console.log(x);
  console.log("fini");
```

```
}

console.log(f1());
console.log("ici");
```

Affiche

```
Promise { <pending> }
ici
10
fini
```

### *Générateur*

```
function * compteur (max) {
  var index =0;
  while (index < max) {
    yield index++;
  }
}

const iterator = compteur (2);
console.log(iterator.next().value);
console.log(iterator.next().value);
console.log(iterator.next().value);
```

Affiche

```
0
1
undefined
```

avec for of

```
const iterator = compteur (5);
for (var i of iterator)
{
      console.log(i);
}
```

Affiche

```
0
1
2
3
4
```

## *Exceptions*

try/catch

```
try {
    i+=1
    console.log("OK")
}
catch (e)
{
    console.log("Probleme", e.message)
}
```

Affiche

```
Probleme i is not defined
```

try/catch/finally

```
try {
    i+=1
    console.log("OK")
}
catch (e)
{
    console.log("Probleme", e.message)
}
finally
{
    console.log("Dans tous les cas")
}
```

affiche

```
Probleme i is not defined
Dans tous les cas
```

Filtrer

```
try {
    // throw new Exception("Mon Exception")
    i+=1
    console.log("OK")
}
catch (e)
{
    if (e instanceof ReferenceError)
    {
        console.log("Probleme", e.message)
    }
```

```
    }
```

Mon exception

```
    try {
        throw "Mon Exception"
        console.log("OK")
    }
    catch (e)
    {
        console.log("Probleme", e)
    }
    finally
    {
        console.log("Dans tous les cas")
    }
```

Affiche

```
    Probleme Mon Exception
    Dans tous les cas
```

Mon exception en tant que objet

```
    try {
        throw new Error("Mon Exception")
        console.log("OK")
    }
    catch (e)
    {
        console.log("Probleme", e.message)
    }
```

Affiche

```
    Probleme Mon Exception
```

## 5. Programmation Orientée Objets

getter/setter

```
    var personne = {
        nom : "TOTO",
        prenom : "toto",
        get getNom() {
                return this.nom
        },
        set setNom(nom) {
                this.nom  = nom;
```

```
        }
    }

    console.log(personne.getNom)
    personne.setNom = "Un autre TOTO"
    console.log(personne.getNom)
```

Affiche

```
    TOTO
    Un autre TOTO
```

### *Classes*

Constructeur et méthodes

```
    class Personne
    {
        constructor (nom,prenom,age)
        {
                this.nom=nom;
                this.prenom=prenom;
                this.age=age;
        }
        getAll (){
                return this.nom + "::"+this.prenom + "::"+this.age
        }
    }

    var p = new Personne("TOTO1","toto1",1)
    console.log(p.getAll())
```

Affiche

```
    TOTO1::toto1::1
```

### *Méthodes et données static*

```
    class Personne
    {
        constructor (nom,prenom,age)
        {
                this.nom=nom;
                this.prenom=prenom;
                this.age=age;
        }
        getAll (){
                return this.nom + "::"+this.prenom + "::"+this.age
        }
```

```
    static getType()
    {
        return Personne.type;
    }
}
Personne.type='Personne';
console.log(Personne.getType());
```

### *Héritage et redéfinition*

```
class Personne
{
    constructor (nom,prenom,age)
    {
        this.nom=nom;
        this.prenom=prenom;
        this.age=age;
    }
    getAll (){
        return this.nom + "::"+this.prenom + "::"+this.age
    }
}

class Salarie extends Personne
{
    constructor (nom,prenom,age,anciennete)
    {
        super(nom, prenom,age)
        this.anciennete=anciennete;
    }
    getAll (){
        return super.getAll() + "::" + this.anciennete
    }
}

var s = new Salarie("TOTO1","toto1",1, 10)
console.log(s.getAll())
```

Affiche

```
TOTO1::toto1::1::10
```