



JAVASCRIPT

GÉNÉRALITÉS



JavaScript®, souvent abrégé en JS, est le langage de script développé par Netscape utilisé dans des millions de pages web et d'applications serveur dans le monde entier. (Wikipédia)"



VERSIONS

ECMAScript

- Création du Javascript en 1995
- Première version standard en 1997: ES1
- A partir de la version ES6 --> changement de format de nom: ES2015
- Aujourd'hui: **ES2020 (ES11)**
- Tous les navigateurs modernes supportent au minimum **ES6**



SYNTAXE - VARIABLES

Définition d'une variable

```
| let variableName1  
| const variableName2
```

Valeur par défaut: *undefined*

Affectation d'une valeur

```
| let variableName1 = 'test'  
| const variableName2 = 'test2'
```



SYNTAXE - TYPES

Types *primitifs*:

- boolean
- number
- string
- object
- function
- undefined
- bigint



SYNTAXE - TYPES

Créer un *string*

```
| const myVariable = 'test'  
| const myVariable = "test"  
| const myVariable = `test`
```

Typage *dynamique*

```
| let myVariable = 42  
| myVariable = 'Ceci est une chaine de caractères'
```



OPÉRATEURS

Obtenir le type d'une variable ***typeof***

```
| const myString = 'test'  
| typeof myString  
| typeof 3
```

Affectation d'une valeur **=**

```
| const myVariable = 1
```

Addition **+**

```
| myVariable + 3
```

Incrément **++**

```
| myVariable++
```



OPÉRATEURS

Soustraction -

| myVariable - 3

Décrément --

| myVariable--

Multiplication *

| myVariable * 2

Division /

| myVariable / 2



OPÉRATEURS

Egalité ==

| myVar1 == myVar2

Inégalité !=

| myVar1 != myVar2

Egalité stricte ===

| myVar1 === myVar2

Inégalité stricte !==

| myVar1 !== myVar2



OPÉRATEURS

Supériorité stricte >

| myVar1 > myVar2

Supérieur ou égale >=

| myVar1 >= myVar2

Infériorité stricte <

| myVar1 < myVar2

Inférieur ou égale <=

| myVar1 <= myVar2





OPÉRATEURS

ET logique **&&**

| myVar1 **&&** myVar2

OU logique **||**

| myVar1 **||** myVar2



STRUCTURES LOGIQUES

if

```
if (condition)
  statement1
} else {
  statement2
}
```

Exemple:

```
if (myVar > 0)
  console.log('myVar is > 0')
} else {
  console.log('myVar is <= 0')
}
```



STRUCTURES LOGIQUES

for

```
for ([initialisation]; [condition]; [expression_finale]) {  
    instruction  
}
```

Exemple:

```
for (let i = 0; i < 10; i++) {  
    console.log(i)  
}
```



STRUCTURES LOGIQUES

for...of

```
for (variable of iterable) {  
    instruction  
}
```

Exemple:

```
for (val of ['a', 'b', 'c']) {  
    console.log(val)  
}
```



STRUCTURES LOGIQUES

for...in

```
for (property in iterable) {  
  instruction  
}
```

Exemple:

```
const object = { firstName: 'Riri', lastName: 'Duck' }  
  
for (property in object) {  
  console.log(object[property])  
}
```



STRUCTURES LOGIQUES

while

```
while (condition) {  
  instruction  
}
```

Exemple:

```
let i = 0  
  
while (i < 10) {  
  console.log(i)  
  i++  
}
```



STRUCTURES LOGIQUES

do...while

```
do {  
    instruction  
}  
while (condition)
```

Exemple:

```
let i = 0  
  
do {  
    console.log(i)  
    i++  
} while (i < 10)
```



FONCTION

- Brique fondamentale
- Encapsule une **série d'instructions**
- Peut accepter des **arguments**
- Plusieurs manières de définir une fonction
- Une fonction **doit être appelée** pour déclencher la série d'instructions encapsulée
- Une fonction peut retourner quelque chose à l'aide du mot clé **return**



FONCTION

Définition d'une fonction

```
// named function
function namedFunction(arg1, arg2) { /* instructions */ }

// Anonymous function
const variableAnonymousFunction = function (arg) { /* instructions */ };

// Arrow function
const variableArrowFunction = (arg) => { /* instructions */ };
```



FONCTION

Appel d'une fonction

```
function sayHelloWorld() {  
  console.log('hello world')  
}
```

```
sayHelloWorld()  
sayHelloWorld()  
sayHelloWorld()
```



FONCTION

Définition d'arguments

```
function sayHelloSomething(arg) {  
  /* Instructions */  
}  
  
function sayHelloSomething(arg = 'world') {  
  /* Instructions */  
}
```

Utilisation d'arguments

```
function sayHelloWorld (arg = 'world') {  
  console.log('hello ' + arg)  
}
```





FONCTION

Portée d'une fonction

- Une fonction a accès à toutes les variables et fonctions définies au sein de celle-ci.
- Une fonction peut également accéder aux variables et fonctions définis dans la même portée qu'elle même.
- Les variables et fonctions définies dans une fonction ne peuvent pas être accédées depuis l'extérieur de la dite fonction.



TABLEAUX

- Permet la manipulation d'une liste de données
- 2 syntaxes disponibles

Syntaxe littéral

```
| const list = [1, 2, 3];
```

Syntaxe utilisant le constructeur Array

```
| const list = new Array(1, 2, 3);
```



TABLEAUX

Accéder à un élément d'un tableaux

```
const myArray = ['a', 'b', 'c'];  
console.log(myArray[1])
```

Accéder à la taille d'un tableaux

```
const myArray = ['a', 'b', 'c'];  
console.log(myArray.length)
```



TABLEAUX

Boucler sur un tableau

```
const myArray = ['a', 'b', 'c'];

for (let i = 0; i < myArray.length; i++) {
  console.log(myArray[i])
}

for (val of myArray) {
  console.log(val)
}

myArray.forEach(val => {
  console.log(val)
})
```



TABLEAUX

Ajouter un élément dans un tableau

```
const myArray = ['a', 'b', 'c'];  
myArray.push('d')
```

Retirer un élément d'un tableau

```
const myArray = ['a', 'b', 'c'];  
myArray.shift()
```

```
const myArray = ['a', 'b', 'c'];  
const indexOfB = myArray.indexOf('b')  
myArray.splice(indexOfB, 1)
```



OBJETS

Définition d'un objet

```
| const myObject = {}
```

Définition d'un objet avec propriétés et méthodes

```
| const myObject = {  
  firstName: 'Riri',  
  lastName: 'Duck',  
  sayHello() {  
    console.log('hello')  
  }  
}
```



OBJETS

Définition d'une propriété

```
const myObject = {}  
myObject.firstName = 'Loulou'
```

Suppression d'une propriété

```
const myObject = {  
  firstName: 'Loulou'  
}  
  
delete myObject.firstName
```

Définition d'une méthode

```
const myObject = {  
  sayHello: () => { console.log('hello') }  
}
```



OBJETS

Accéder à une propriété

```
const myObject = {  
  firstName: 'Fifi'  
}  
  
console.log(myObject.firstName)
```

Appeler une méthode

```
const myObject = {  
  sayHello() {  
    console.log('hello')  
  }  
}  
  
myObject.sayHello()
```



OBJETS

Accéder à une propriété depuis une méthode

```
const myObject = {  
  firstName: 'Riri',  
  sayHello() {  
    console.log('hello' + this.firstName)  
  }  
}  
  
myObject.sayHello()
```

this représente le contexte de l'objet.



CLASS

Définition d'une classe

```
class MyClasss {  
  firstName  
  lastName  
  
  sayHello() {  
    console.log('Hello, my name is ' + this.firstName + this.lastName)  
  }  
}
```



CLASS

Constructeur d'une classe

```
class MyClasss {  
  firstName  
  lastName  
  
  constructeur () {  
    this.firstName = firstName  
    this.lastName = lastName  
  }  
  
  sayHello() {  
    console.log('Hello, my name is ' + this.firstName + this.lastName)  
  }  
}
```



CLASS

Instantiation d'une classe

```
class MyClass {
  firstName
  lastName

  constructeur () {
    this.firstName = firstName
    this.lastName = lastName
  }

  sayHello() {
    console.log('Hello, my name is ' + this.firstName + this.lastName)
  }
}

const myInstance = new MyClass('Riri', 'Duck')
console.log(myInstance.firstName)
myInstance.sayHello()
```



CLASSES - HÉRITAGES

- Mot clé **extends**
- Possibilité de faire référence à la classe parent via le mot clé **super**

```
class Person {  
  speak() { console.log('speak from parent') }  
}  
  
class Child extends Person {  
  speak() {  
    console.log('speak from child')  
    super.speak()  
  }  
}
```





DOCUMENT

- Représente la page web courante
- Fournit des fonctionnalités globales
- Donne accès au **DOM** de la page web courante



QUERYSELECTOR

- retourne le premier élément dans le document correspondant au sélecteur.
- prends en paramètre un sélecteur css
- renvoie **null** si le sélecteur ne correspond à aucun élément

```
<body>  
  <h1>Ceci est un titre</h1>  
</body>
```

```
const h1 = document.querySelector('h1')
```



QUERYSELECTORALL

- retourne une liste d'éléments dans le document correspondant au sélecteur.
- prends en paramètre un sélecteur css
- renvoie un objet de type **NodeList**
- renvoie une **NodeList** vide si le sélecteur ne correspond à aucun élément

```
<body>  
  <p>Ceci est un premier paragraphe</p>  
  <p>Ceci est un second paragraphe</p>  
</body>
```

```
const pList = document.querySelectorAll('p')
```





HTMLELEMENT

Représente le type d'objet le plus couramment renvoyé par `querySelector` et `querySelectorAll`

Propriétés

- `innerText`
- `style`
- `className`

Méthodes



DOM EVENTS

- click
- mouseenter
- mouseleave
- keyup

Ecouter un évènement

- addEventListener
- on{eventName}

```
| <button onclick="console.log('clicked')">
```

```
| const button = document.querySelector('button')  
| button.addEventListener('click', () => console.log('click'))
```

© Copyright 2021 Zenika. All rights reserved

