



PARTIE 3 : **JAVASCRIPT**

CHAPITRE 1 : JS

PREAMBULE

**HISTORIQUE, MOTIVATION,
ENVIRONNEMENT**

HISTORIQUE

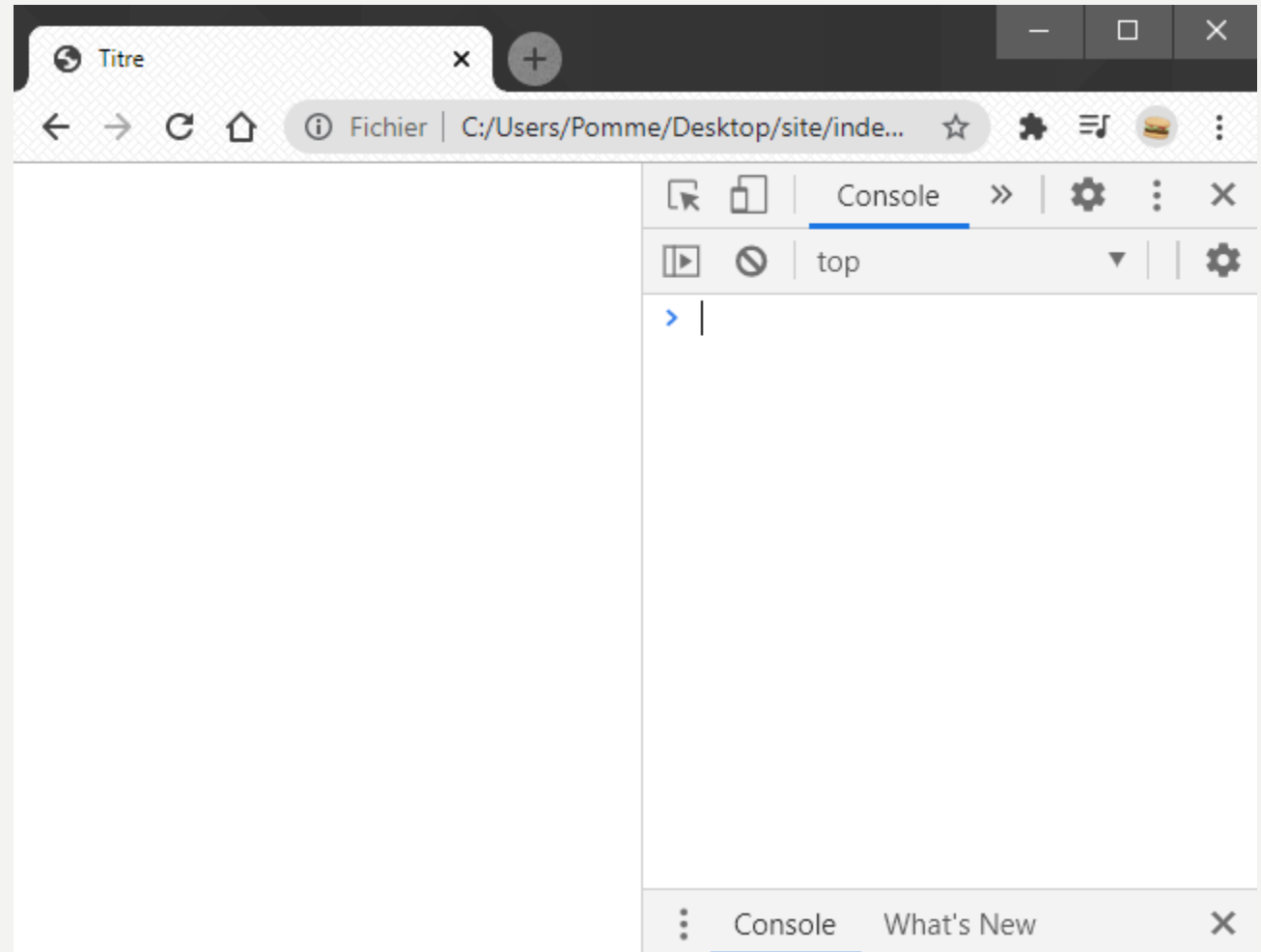
- Brendan Eich
- LiveScript → Devient JavaScript (marketing)
- 1995 première version
- Soumis à l'ECMA International et devient ECMAScript
- 1999 : ES3
- ES4 abandonné
- 2009 : ES5
- 2015 : ES6

JS KESAKO

- Client-side
- Langage script
- Orienté objet
- Standardisé sous le nom ECMA Script
- Supporté par tous les navigateurs
 - Le plus aboutit des langages de ce genre (Jscript ou Jscript.NET)
- Pas seulement pour le web
 - Firefox par exemple
 - Node.js

ENVIRONNEMENT

- Editeur de texte
- HTML
- Navigateur web
- Console (CTRL+MAJ+J)



OU PLACE-T-ON LE JS ?

- Externe
 - `<script src></script>`
- Head
 - `<script>codeJs</script>`
- Inline
 - `<balise onEvent="codeJS" />`
- Encore une fois les trois sont valables, il faut penser à la factorisation et l'optimisation

CHAPITRE 2 :

PREMIER PAS EN

JS

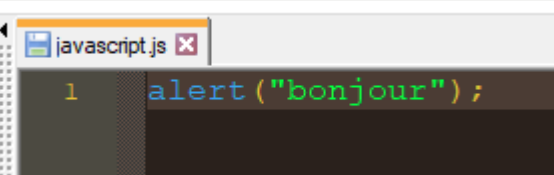
LA BOITE DE DIALOGUE

- alert(...);

1

```
<body onload="alert('Bonjour')">
</body>
```

2

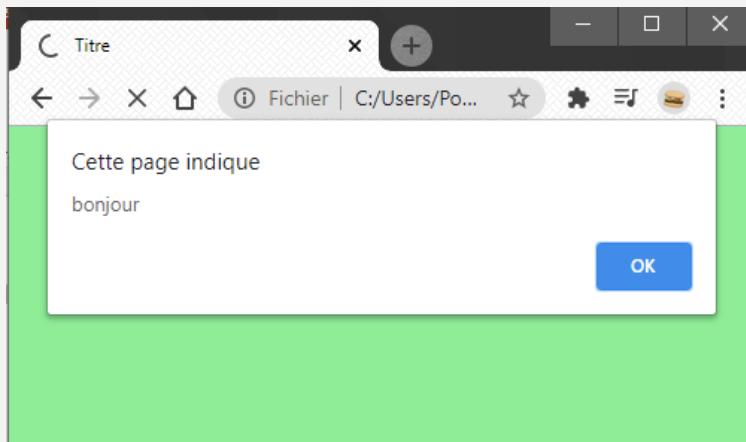


+

```
<head>
  <title>Titre</title>
  <link rel="stylesheet" href="css/style.css" />
  <script type="text/javascript" src="js/javascript.js"></script>
</head>
```

3

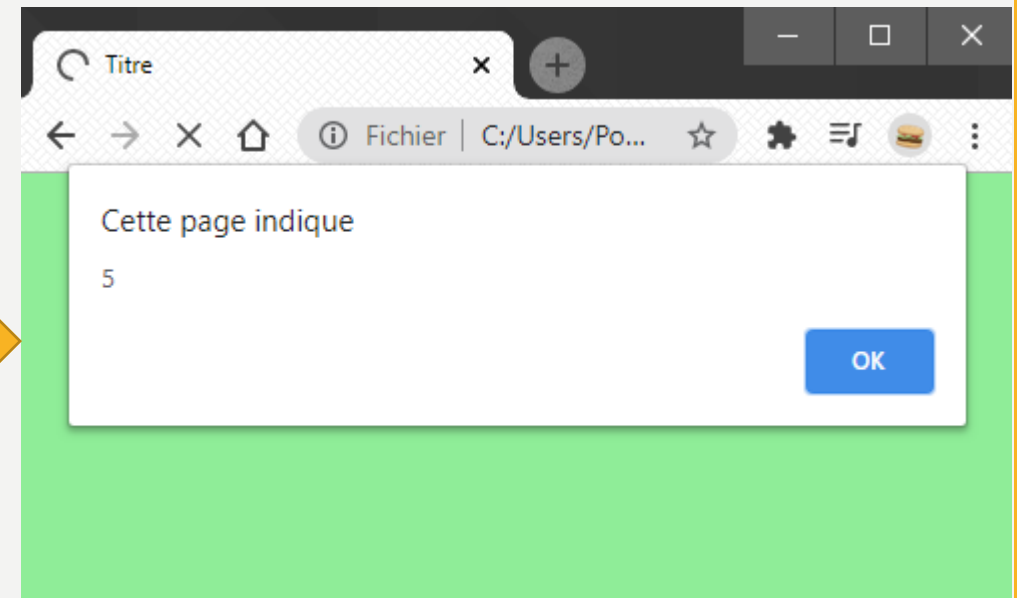
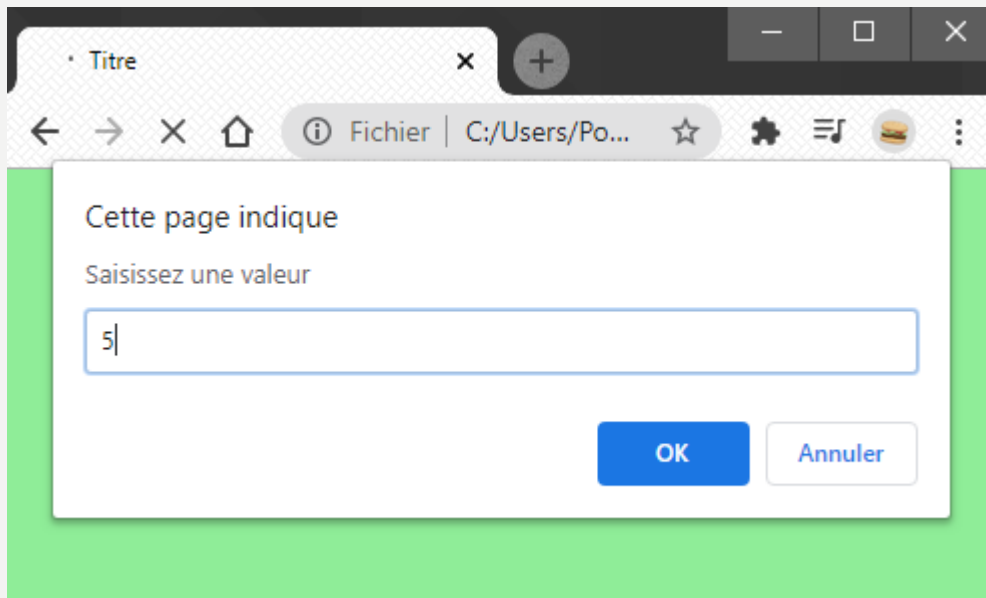
```
<script>
  alert('Bonjour');
</script>
```



LA BOITE DE DIALOGUE

- `prompt(...);`

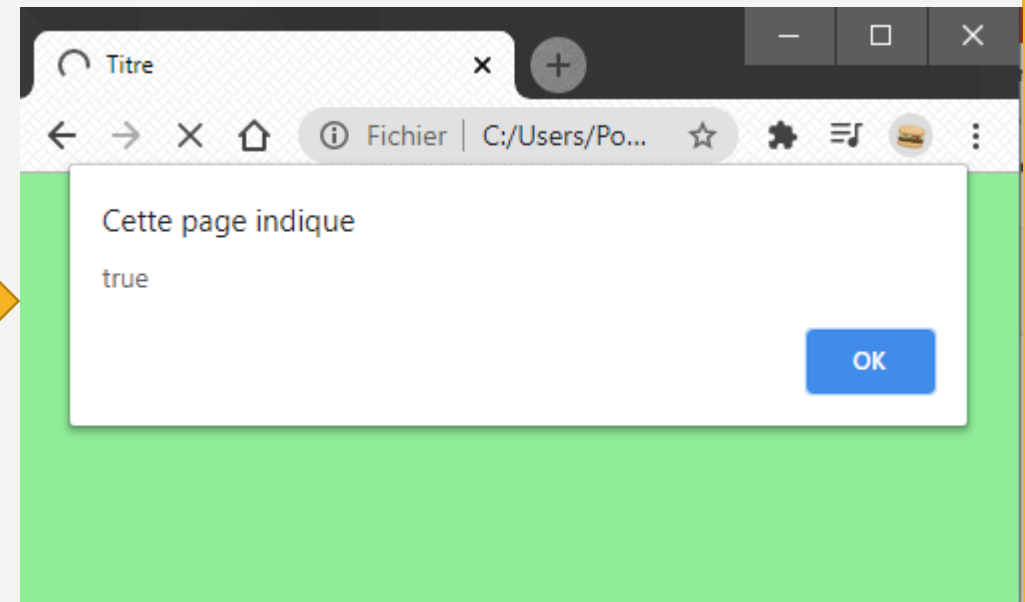
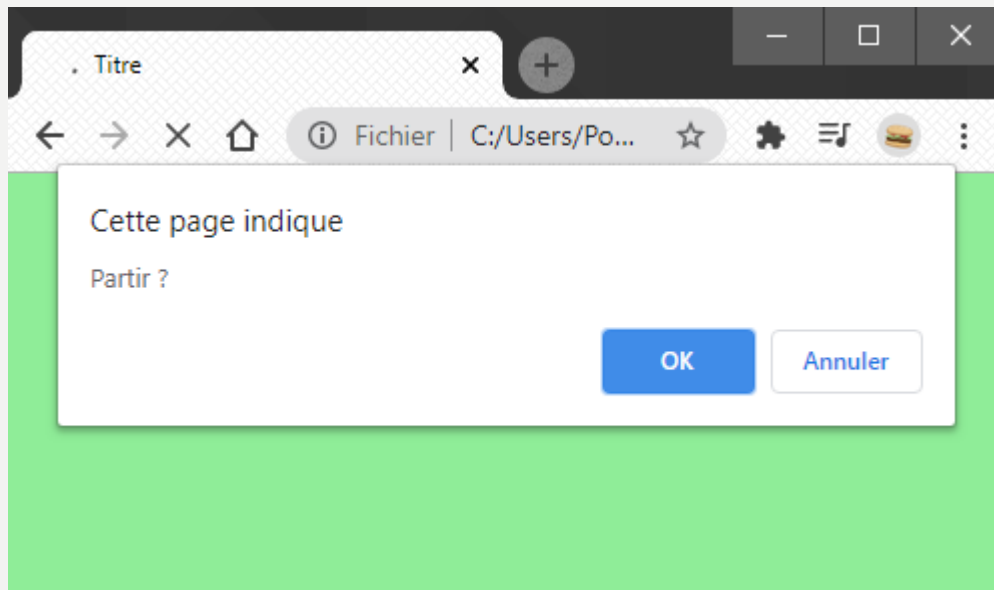
```
javascript.js  
1 var x = prompt("Saisissez une valeur");  
2 alert(x);
```



LA BOITE DE DIALOGUE

- `confirm(...);`

```
javascript.js x
1 boolean x = confirm("Partir ?");
2 alert(x);
```



ECRIRE UN RÉSULTAT

- `alert()` → ?
- `document.write()` **!!!!!!** Cette fonction PEUT supprimer tous les éléments HTML existants
- `console.log()` → Permet d'écrire dans la console
- `innerHTML` → Permet d'écrire au sein de l'HTML
 - représente le Contenu d'une balise

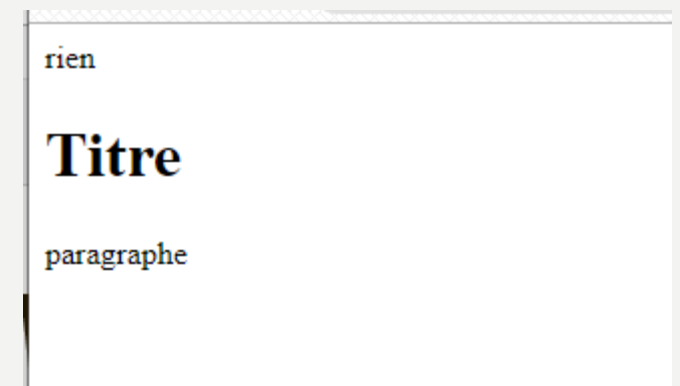
DOCUMENT.WRITE

```
9 <body>
10 <h1>Titre</h1>
11 <p>paragraphe</p>
12 </body>
```

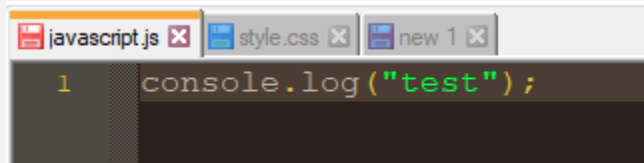


```
1 <title>Titre</title>
5 <link rel="stylesheet" href="css/style.css" />
6 <script type="text/javascript" src="js/javascript.js">
7 </script>
8 </head>
9 <body>
10 <h1>Titre</h1>
11 <p>paragraphe</p>
12 </body>
13 </html>
```

```
1 document.write("rien");
```

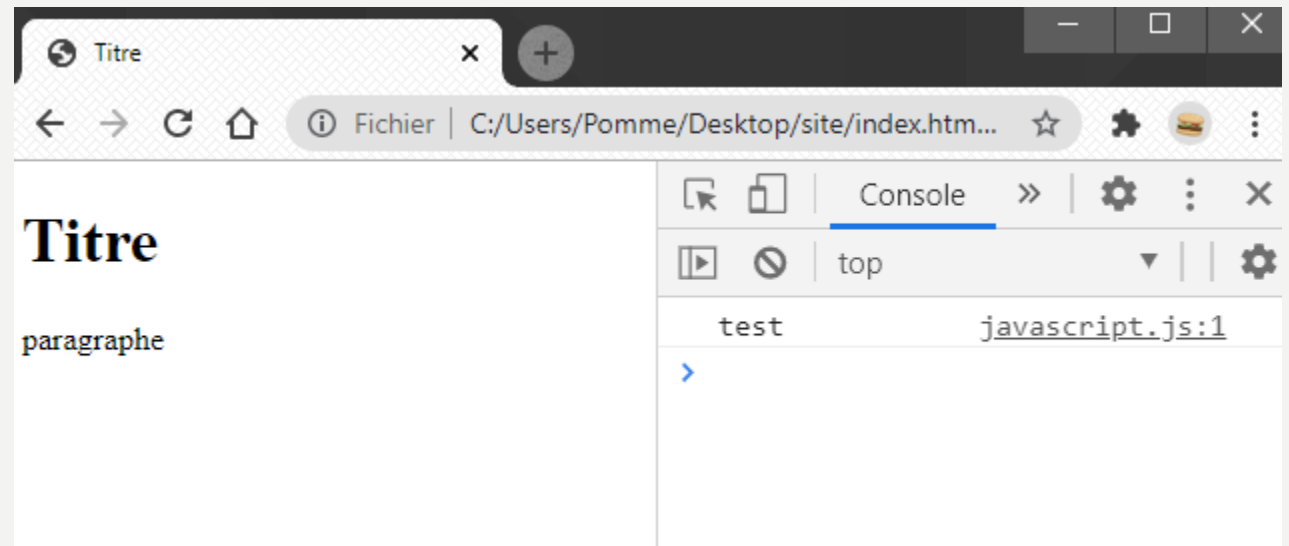


CONSOLE.LOG



```
1 console.log("test");
```

A screenshot of a code editor with three tabs: 'javascript.js', 'style.css', and 'new 1'. The 'javascript.js' tab is active, showing a single line of code: `1 console.log("test");`. The line number '1' is in the left margin, and the code is syntax-highlighted with 'console.log' in blue, the string 'test' in red, and the semicolon in blue.

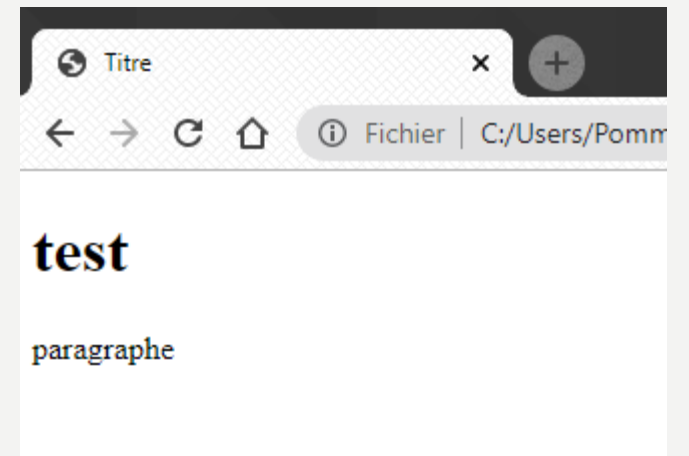


INNERHTML

- On peut y mettre de l'HTML

```
<body>
  <h1 id="test">Titre</h1>
  <p>paragraphe</p>
</body>
```

```
javascript.js x style.css x new 1 x
1 document.getElementById("test").
2 innerHTML="test";
```



TERMINOLOGIE

- Variable
 - Fonction
 - **Instruction**
 - Condition
 - Bloc
 - Boucle
 - Tableaux
-
- Valable pour d'autres langages de programmation

INSTRUCTION

- C'est une action qu'on demande à l'ordinateur d'exécuter
 - Déclaration
 - Affectation
 - Appel de fonction
- Elle se finit par un “;”
- On peut tout écrire sur une seule ligne tant qu'il y a des points-virgules

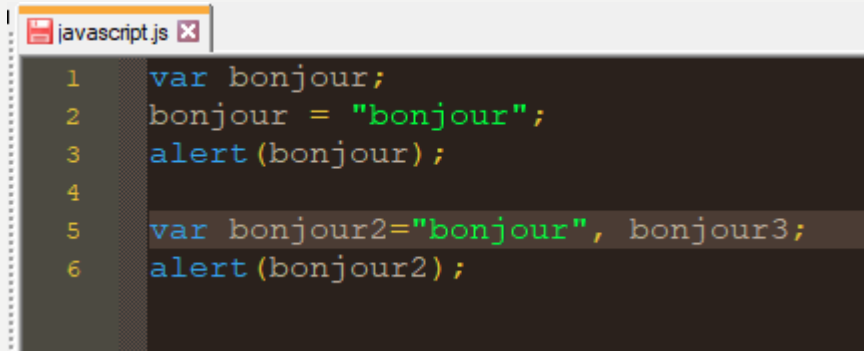
TERMINOLOGIE

- **Variable**
- Type
- Fonction
- Instruction
- Condition
- Bloc
- Boucle
- Tableaux

LES VARIABLES

- Espace de stockage nommé
 - On lui donne un nom (**déclaration**)
 - Qui ne peut pas commencer par un chiffre
 - Qui ne peut pas être un mot-clef du langage
 - On écrit dedans (**affectation**)
 - Avec le symbole “=”
 - Et on le lit (**lecture/accès**)
 - Grâce à son nom
- Le mot-clef : **var**
- Typage for / Typage faible

LES VARIABLES



```
1 var bonjour;  
2 bonjour = "bonjour";  
3 alert(bonjour);  
4  
5 var bonjour2="bonjour", bonjour3;  
6 alert(bonjour2);
```

- 2 types de notations
 - **Déclaration** puis **affectation** (ligne 1)
 - **Déclaration** et **affectation** (ligne 5)
- Ligne 3 et 6 accès à la variable en **lecture**, lors d'un appel de **fonction**
- Nous n'avons pas oublié de mettre un ";" après chaque **instruction**
- Déclarer plusieurs variables en une seule **instruction** grâce à ","

TERMINOLOGIE

- Variable
- **Type**
- Fonction
- Instruction
- Condition
- Bloc
- Boucle
- Tableaux

TYPES PRIMITIFS

- Que peut-on mettre dans une variable ?
 - Number
 - Entier
 - Décimal
 - Booléen
 - String
 - Chaîne de caractères
 - Caractère
 - Le mot-clef : **typeof**

```
javascript.js x style.css x
1  var a = 5;
2  var b = 5.5;
3  var c = true;
4  var d = "Une phrase";
5  var e = 'e';
6  var f;
7
8  console.log(typeof(a));
9  console.log(typeof(b));
10 console.log(typeof(c));
11 console.log(typeof(d));
12 console.log(typeof(e));
13 console.log(typeof(f));
```

Console		top	
number	javascript.js:8		
number	javascript.js:9		
boolean	javascript.js:10		
string	javascript.js:11		
string	javascript.js:12		
undefined	javascript.js:13		

TYPES COMPLEXES

- Objets
 - Ensemble d'attributs
 - De types potentiellement différents
- Tableaux
 - Ensemble de valeurs
 - De types potentiellement différents
- Les tableaux sont un type particulier d'objet

UN OBJET

- Un objet contient
 - Des propriétés
 - Des fonctions
- Les mots clefs qui vont nous intéresser : **new** et **this**
 - **new** pour instancier un objet
 - **this** pour désigner l'objet courant
 - **null** : vide
- La programmation orientée objet s'accompagne d'un ensemble de fonctionnalité
 - Héritage
 - Polymorphisme
 - ...

UN OBJET SIMPLE

```
javascript.js x style.css x new 1 x
1  =function Personne(prenom, nom) {
2      this.nom = nom;
3      this.prenom = prenom;
4      = this.presentation = function() {
5          console.log("Je m'appelle "+
6              this.prenom+" "+this.nom);
7      };
8  }
9  var moi = new Personne("Geoffrey", "WILHELM");
10 moi.presentation();
```

top Filter Default

Je m'appelle Geoffrey WILHELM	javascript.js:5
Show console details	
>	

UN TABLEAU

- Un tableau est c'est un ensemble de valeur de même type ou non
 - Une case d'un tableau possède
 - Un indice numérique (index)
 - Une valeur
- L'indice de la première case d'un tableau vaut 0
- La taille d'un tableau, c'est le nombre d'éléments dans le tableau
- Le dernière indice d'un tableau est donc : $\text{La taille du tableau} - 1$
- On déclare un tableau grâce à des crochets
- On sépare les éléments avec une virgule
- J'accède à la valeur d'une case grâce
 - Au nom du tableau suivi d'un crochet ouvrant
 - L'index de la case suivi d'un crochet fermant

UN TABLEAU

- Déclaration :

```
1 var tableau = ['element1', 2];  
2
```

- Lecture :

```
3 console.log(tableau[0]);
```



element1	javascript.js:3
----------	-----------------

- Ecriture :

```
2 tableau[0]=1;  
3 console.log(tableau[0]);
```



1	javascript.js:3
---	-----------------

- La taille :

```
3 console.log(tableau.length);
```



2	javascript.js:3
---	-----------------

TABLEAU ASSOCIATIF

- Au lieu d'un index les valeurs sont associés à un identifiant

```
1 var tableau = {  
2   'nom' : 'element1',  
3   'value' : 2  
4 };  
5 console.log(tableau[0]);  
6 console.log(tableau['nom']);  
7 console.log(tableau.nom);
```



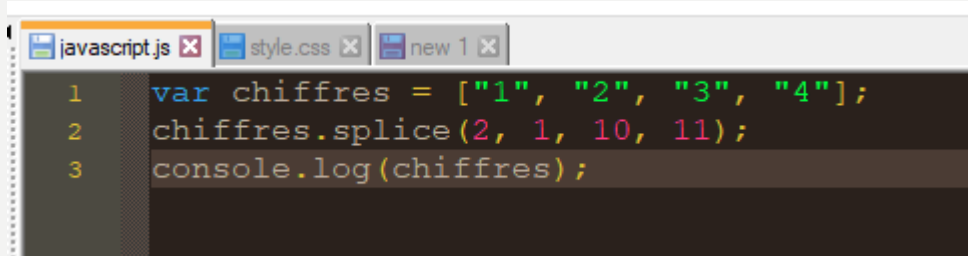
Elements Console >>		⚙️ ⋮ ✕
▶ ⛔	top ▼	👁 Filter Default le ⚙️
undefined		javascript.js:5
element1		javascript.js:6
element1		javascript.js:7
>		

OPÉRATIONS SUR LES TABLEAUX

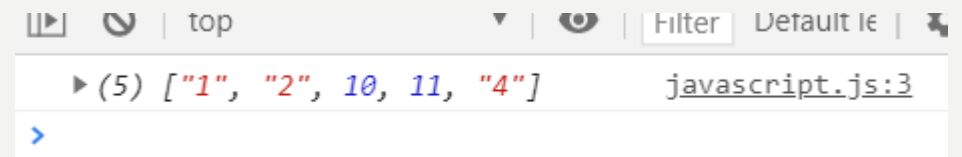
- toString() : commun à tous les objets
 - Peut-être appelé automatiquement
- Pop() : retirer le dernier élément
- Push() : ajouter un élément à la fin
- Shift() : supprime le premier élément
- Unshift() : ajouter un élément au début
- Length : donne la taille du tableau
- Delete : Supprime un élément mais sans supprimer la case du tableau
- Splice() : insertion d'éléments à un indice dans le tableau
- Concat() : concaténation de tableaux
- Slice() : découpe d'un tableau
- Filter() : filtrer grâce à une fonction de condition

TABLEAUX : SPLICE

- 3 à N paramètres :
 - Indice d'insertion
 - Nombre d'éléments à supprimer
 - Des éléments à ajouter



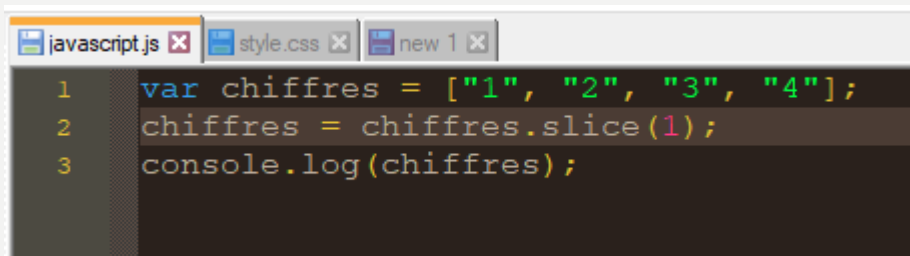
```
javascript.js x style.css x new 1 x
1 var chiffres = ["1", "2", "3", "4"];
2 chiffres.splice(2, 1, 10, 11);
3 console.log(chiffres);
```



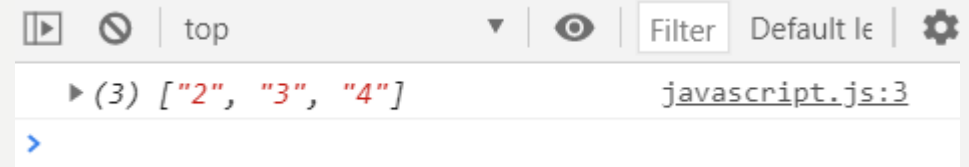
```
top | Filter | Default | 
▶ (5) ["1", "2", 10, 11, "4"] javascript.js:3
>
```

TABLEAUX : SLICE

- Indice de départ de la découpe en paramètre



```
javascript.js x style.css x new 1 x
1 var chiffres = ["1", "2", "3", "4"];
2 chiffres = chiffres.slice(1);
3 console.log(chiffres);
```

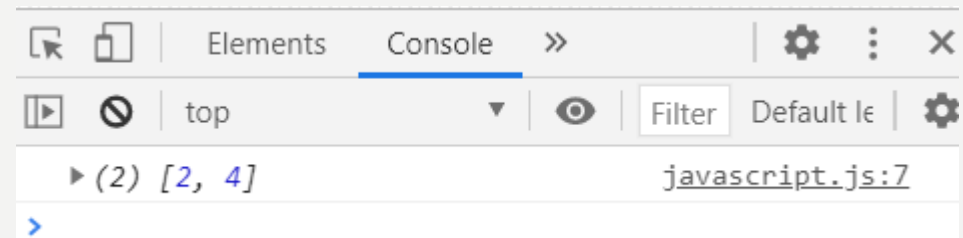


```
▶ (3) ["2", "3", "4"] javascript.js:3
>
```

TABLEAUX : FILTER

- Une fonction de filtre
- Un tableau à filtrer
- La fonction retourne vrai ou faux

```
javascript.js style.css new 1 x
1 var tableau = ['element1', 2,
2   'element2', 4];
3 function checkNumber(value) {
4   return Number.isInteger(value);
5 }
6 tableau = tableau.filter(checkNumber);
7 console.log(tableau);
8
```

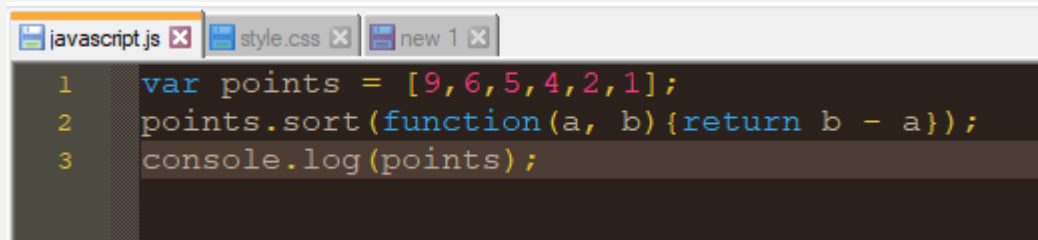


The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the result of the filter operation: `(2) [2, 4]`, indicating that the array now contains only the integer values. The source is identified as `javascript.js:7`. The console interface includes standard controls like a search icon, a refresh icon, and a filter input field.

TRI SUR LES TABLEAUX ET LES OBJETS

– Sort()

- Triage naturel (ordre alphabétique ou ordre croissant)
- Fonction de comparaison



```
javascript.js x style.css x new 1 x
1 var points = [9,6,5,4,2,1];
2 points.sort(function(a, b){return b - a});
3 console.log(points);
```



```
Show console log [9, 6, 5, 4, 2, 1] javascript.js:3
>
```

– Reverse() : inverse

OPERATIONS

- Opérations mathématiques
 - + - * / % ++ --
 - Combiné avec un = pour une écriture compressé
- Concaténation
 - +
 - Une, au moins, des opérandes doit être une chaine de caractères
- Opérations logiques
 - || && !
- Opérations de comparaison
 - == === != !== > < >= <= ?
- Cast (conversion)
 - https://www.w3schools.com/jsref/jsref_type_conversion.asp

OPERATIONS MATHÉMATIQUES

- Addition
- Soustraction
- Multiplication
- Division
- Modulo
- Incrémentation/Décrémentation
 - ++var
 - var++
- Combiné avec un =

```
javascript.js x style.css x
1  var a = 5, b = 6;
2
3  console.log("addition = "+(a+b));
4  console.log("soust = "+(a-b));
5  console.log("mult = "+(a*b));
6  console.log("division = "+(a/b));
7  console.log("modulo = "+(a%b));
8  console.log("incre = "+(a++));
9  console.log("incre2 = "+(++a));
10
11  b+=6;
12
13  console.log("result = "+b);
```

Console	
top	
addition = 11	javascript.js:3
soust = -1	javascript.js:4
mult = 30	javascript.js:5
division = 0.8333333333333334	javascript.js:6
modulo = 5	javascript.js:7
incre = 5	javascript.js:8
incre2 = 7	javascript.js:9
result = 12	javascript.js:13

OPERATIONS COMPARAISON

- La comparaison aboutit sur un boolean (true ou false)
- Instaure une **condition**
 - == égalité simple
 - === égalité et même type
 - != différent
 - !== différent de valeur OU différent de type
 - > supérieur
 - < inférieur
 - >= supérieur ou égal
 - <= inférieur ou égal
 - ? Condition ternaire

```
javascript.js x style.css x
1  var a = 5, b = 6;
2
3  console.log("5" == a);
4  console.log("5" === a);
5  console.log("5" != a);
6  console.log("5" !== a);
7  console.log(a < b);
8  console.log(a > b);
9  console.log(a <= b);
10 console.log(a >= a);
11 console.log(a <= 5 ? a : b);
12 console.log(a <= 4 ? a : b);
13
```

true	javascript.js:3
false	javascript.js:4
false	javascript.js:5
true	javascript.js:6
true	javascript.js:7
false	javascript.js:8
true	javascript.js:9
true	javascript.js:10
5	javascript.js:11
6	javascript.js:12

OPERATIONS LOGIQUES

- Combinaison de comparaison
 - && ET
 - || OU
- ! N'EST PAS

```
1 var a = 5, b = 7, c = 6;  
2  
3 console.log(c < a || c < b);  
4 console.log(c > a && c > b);  
5 console.log(c > a && c < b);  
6 console.log(!c > a && c < b);  
7 console.log(!(c > a && c < b));  
8
```

Console

top

true	javascript.js:3
false	javascript.js:4
true	javascript.js:5
false	javascript.js:6
false	javascript.js:7

>

CAST

- Tout peut être converti en string
- Un type complexe ne peut être converti en type primitif
- Un booléen peut être converti en nombre
 - False → 0
 - True → 1
- Une chaîne de caractère qui ne contient que des nombres peut être converti en nombre

```
javascript.js style.css
1 var a = "11";
2 var b = "0"
3
4 console.log(Number(a));
5 console.log(Boolean(a));
6 console.log(Boolean(Number(b)));
7
```

Console >> ⚙️ ⋮ ✕

▶ ⛔ top ▼ ⚙️

11	javascript.js:4
true	javascript.js:5
false	javascript.js:6
>	

UN MOT SUR LA CASSE ET LE CAMEL CASE

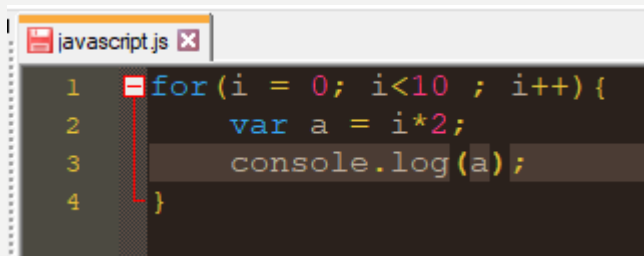
- Casse
 - Sensible ou insensible à la casse
 - Les majuscules et les minuscules
 - Le javascript est case sensitive
- camelCase
 - Convention d'écriture en informatique
 - Les mots sont collés
 - Une capitale à chaque mot
 - Sauf au premier qui peut être en capitale ou non
- ExempleDeCamelCase
- exempleDeCamelCase
- Exempledecamelcase
- ExempledeCamelcase

TERMINOLOGIE

- Variable
- Type
- Fonction
- Instruction
- Condition
- Bloc
- Boucle
- Tableaux

BLOC

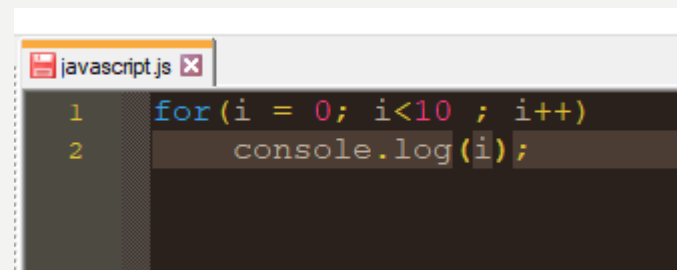
- Un bloc contient un ensemble d'instruction
- Une seule instruction ne nécessite pas de bloc
- Un bloc est délimité par des accolades



A screenshot of a code editor window titled 'javascript.js'. It shows a JavaScript code block defined by curly braces. The code is as follows:

```
1 for(i = 0; i<10 ; i++){  
2     var a = i*2;  
3     console.log(a);  
4 }
```

ACCOLADES



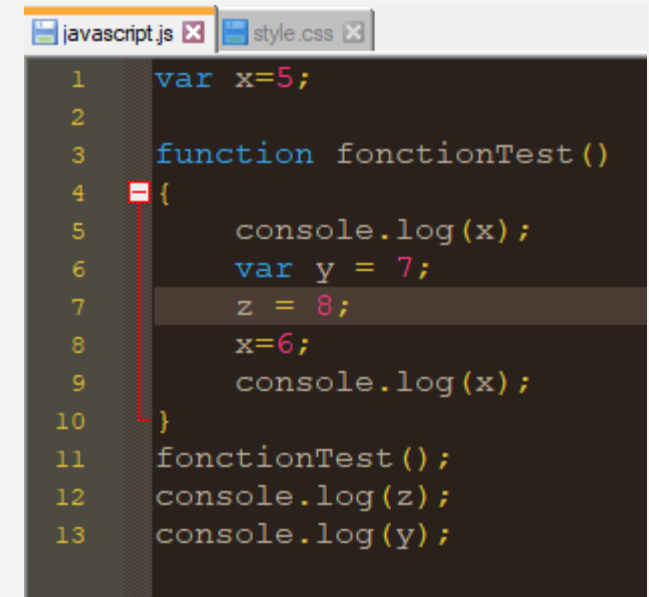
A screenshot of a code editor window titled 'javascript.js'. It shows a JavaScript statement without curly braces. The code is as follows:

```
1 for(i = 0; i<10 ; i++)  
2     console.log(i);
```

PAS D'ACCOLADE

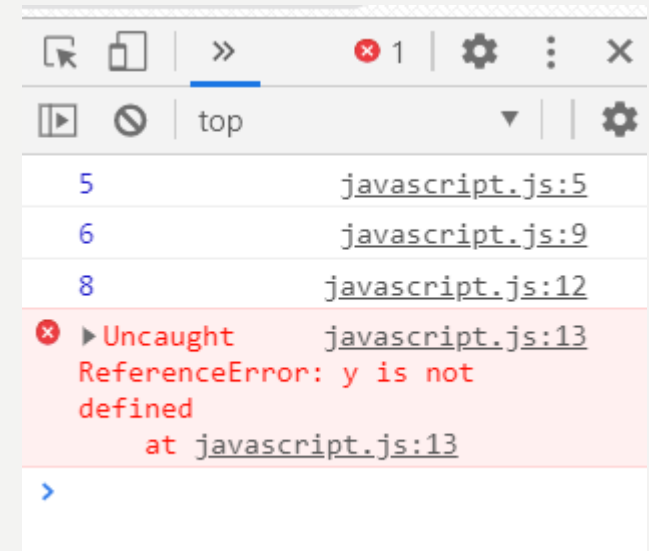
PORTÉE DES VARIABLES

- Une variable possède un scope.
 - Une variable existe dans son bloc et dans les blocs fils (x)
 - SAUF si on n'utilise pas le mot-clef **var** pour la déclaration (z)
 - N'est pas accessible dans un bloc parent (y)



```
1  var x=5;
2
3  function fonctionTest()
4  {
5      console.log(x);
6      var y = 7;
7      z = 8;
8      x=6;
9      console.log(x);
10 }
11 fonctionTest();
12 console.log(z);
13 console.log(y);
```

The screenshot shows a code editor with two tabs: 'javascript.js' and 'style.css'. The JavaScript code is as follows: Line 1: `var x=5;` Line 2: (empty) Line 3: `function fonctionTest()` Line 4: `{` Line 5: `console.log(x);` Line 6: `var y = 7;` Line 7: `z = 8;` Line 8: `x=6;` Line 9: `console.log(x);` Line 10: `}` Line 11: `fonctionTest();` Line 12: `console.log(z);` Line 13: `console.log(y);` A red vertical line with a horizontal bar at the top (line 4) indicates the scope of the function. It extends down to line 10, covering lines 5 through 9. This visualizes that variables `x` and `y` are in the function's scope, while `z` is in the global scope.



The screenshot shows the bottom part of a web browser window, specifically the developer console. The top bar shows navigation icons, a search icon, and a red 'x' icon with the number '1'. Below this, there's a 'top' button and a settings gear icon. The console log shows the following entries:

Line	File
5	javascript.js:5
6	javascript.js:9
8	javascript.js:12
13	javascript.js:13

The entry for line 13 is highlighted in red and shows an error:

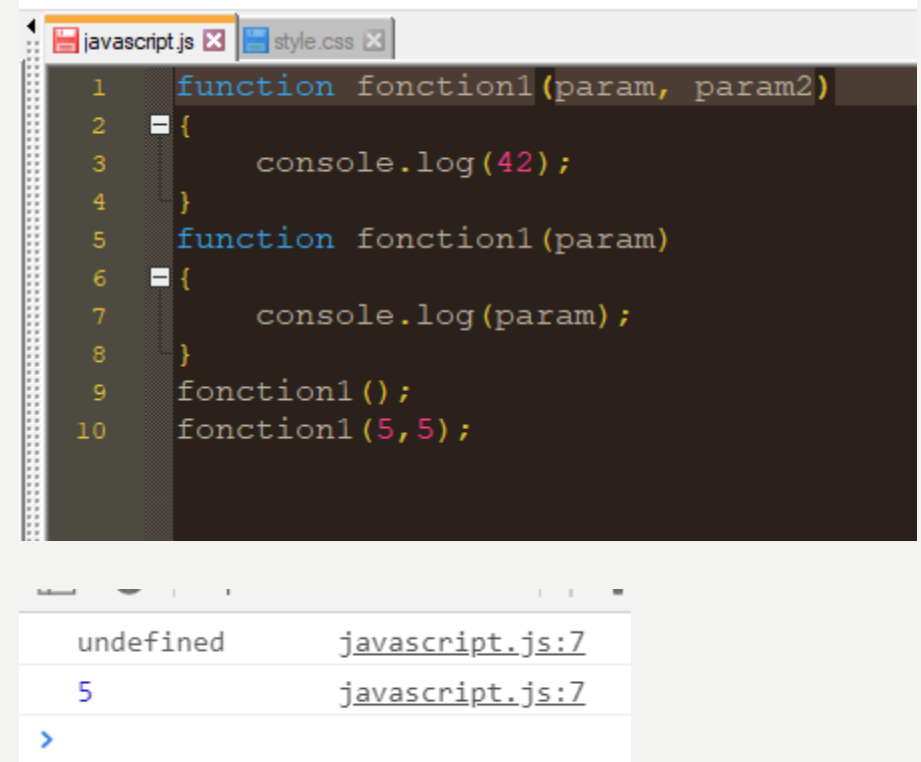
```
Uncaught ReferenceError: y is not defined
    at javascript.js:13
```

TERMINOLOGIE

- Variable
- Type
- Fonction
- Instruction
- Condition
- Bloc
- Boucle
- Tableaux

FONCTION EN JS

- Un **prototype** permet de décrire une fonction
 - Peut retourner une valeur ou non
 - A un nom
 - Possède 0 à N paramètres
- Invocation
- Contrairement à d'autre langage la surcharge de fonction n'existe pas
- S'il y a conflit c'est la dernière définition qui l'emporte
- Le retour d'une valeur (l'envoi d'une valeur à un bloc parent dans la pile d'exécution) se fait avec le mot clef **return**



```
1 function fonction1(param, param2)
2 {
3     console.log(42);
4 }
5 function fonction1(param)
6 {
7     console.log(param);
8 }
9 fonction1();
10 fonction1(5,5);
```

undefined	javascript.js:7
5	javascript.js:7
>	

FONCTION EN JS

- Le retour peut se capturer dans une variable
- Le retour peut servir de paramètre à une fonction

```
1 function fonction1(param)
2 {
3     param+=2
4     return param;
5 }
6 var x = fonction1(3);
7 console.log(x);
8 console.log(fonction1(5));
9
```



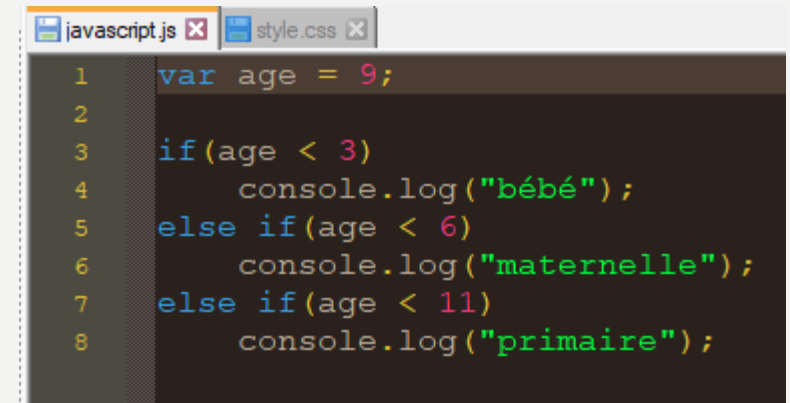
Elements		Console		
top			Filter	Default le
5		javascript.js:7		
7		javascript.js:8		

TERMINOLOGIE

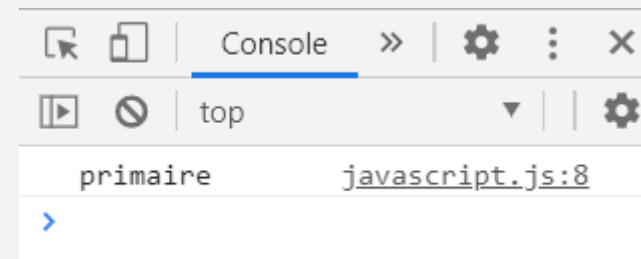
- Variable
- Type
- Fonction
- Instruction
- Condition
- Bloc
- Boucle
- Tableaux

CONDITION

- IF(condition){
 instruction;
} ELSE IF(condition){
 instruction;
} ELSE {
 instruction;
}
- SI...SINON SI...SINON
- La condition est un booleen



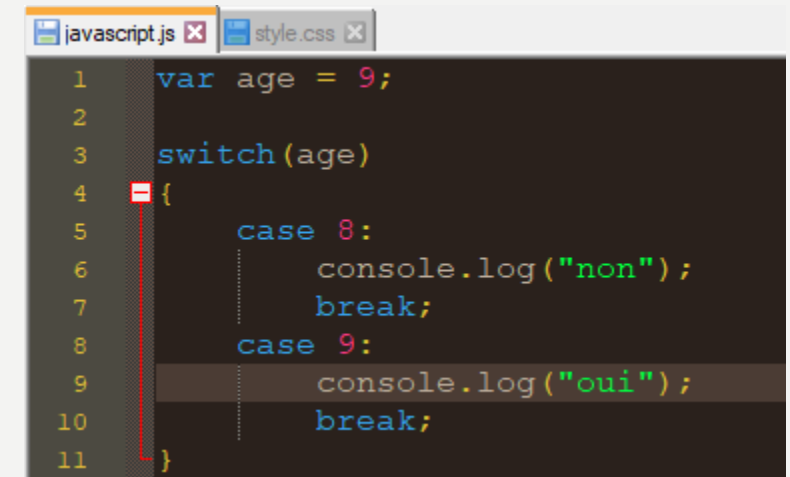
```
1 var age = 9;  
2  
3 if(age < 3)  
4     console.log("bébé");  
5 else if(age < 6)  
6     console.log("maternelle");  
7 else if(age < 11)  
8     console.log("primaire");
```



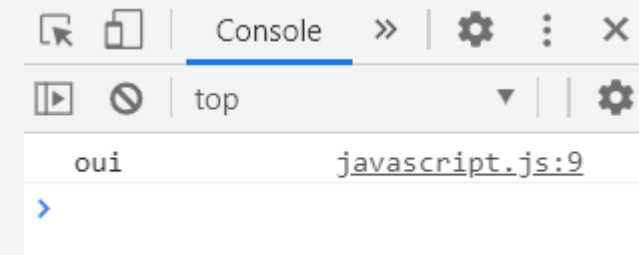
```
Console >> | [Settings] [Close]  
top [Settings]  
primaire javascript.js:8  
>
```

CONDITION

- SWITCH(variable)
{
 CASE valeur:
 instruction;
 break;
 default:
 instruction;
}
- Sélection d'un bloc à exécuter en fonction de l'égalité entre la variable et un des cas



```
1 var age = 9;  
2  
3 switch(age)  
4 {  
5     case 8:  
6         console.log("non");  
7         break;  
8     case 9:  
9         console.log("oui");  
10        break;  
11 }
```



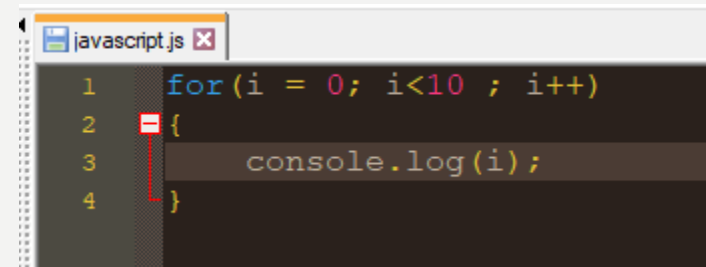
```
Console >> [Settings] [Close]  
top [Filter] [Settings]  
oui javascript.js:9  
>
```

TERMINOLOGIE

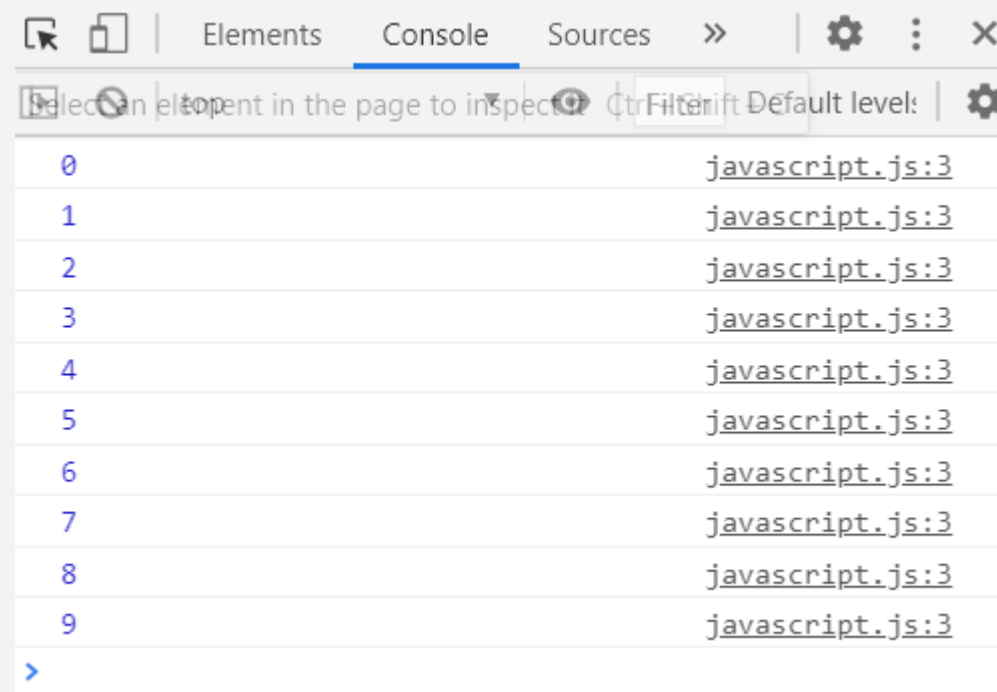
- Variable
- Type
- Fonction
- Instruction
- Condition
- Bloc
- **Boucle**
- Tableaux

BOUCLE POUR

- Mot-clef : **for**
- Valeur de départ
- Condition d'arrêt
- Pas d'incrémentation
 - `i++` → `i+=1` → `i = i + 1`

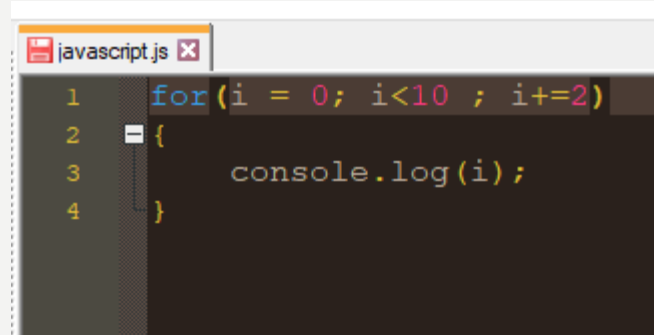


```
1 for(i = 0; i<10 ; i++)  
2 {  
3     console.log(i);  
4 }
```



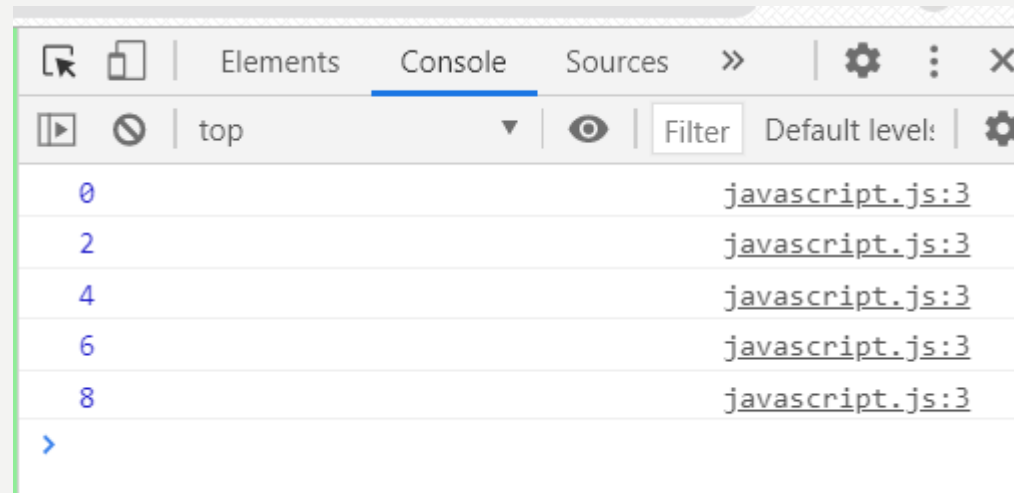
0	javascript.js:3
1	javascript.js:3
2	javascript.js:3
3	javascript.js:3
4	javascript.js:3
5	javascript.js:3
6	javascript.js:3
7	javascript.js:3
8	javascript.js:3
9	javascript.js:3
>	

BOUCLE POUR



```
1 for(i = 0; i<10 ; i+=2)
2 {
3     console.log(i);
4 }
```

- Le pas d'incrémentation est de 2



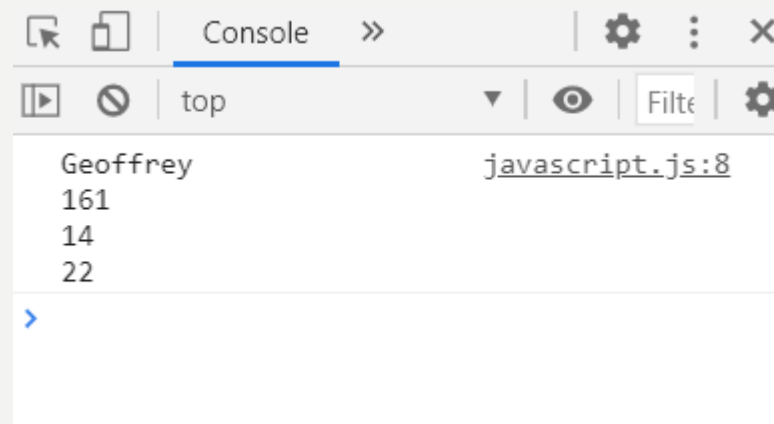
Value	Source
0	javascript.js:3
2	javascript.js:3
4	javascript.js:3
6	javascript.js:3
8	javascript.js:3
>	

BOUCLE POUR DANS : OBJET

- Mots-clefs : **for** et **in**

```
javascript.js
1 var info0102 = {prof:"Geoffrey", Etudiants:161, CM:14, TP:22};
2 var toPrint = "";
3 var x;
4
5 for (x in info0102) {
6     toPrint += info0102[x]+"\n";
7 }
8 console.log(toPrint);
```

- Permet de parcourir les différents attributs d'un objet
- La condition d'arrêt est implicite
- Pas besoin de pas d'incrément



```
Console
top
Geoffrey
161
14
22
javascript.js:8
```

BOUCLE POUR DE : COLLECTION

- Un tableau est une collection
- Mots-clefs : for et of
- Les objets “iterable”
 - Array
 - Map
 - Set
 - String
 - ...

```
javascript.js x
1  var matieres = ['info0101', 'info0102', 'info0103'];
2  var x;
3
4  for (x of matieres) {
5      console.log(x+"\n");
6  }
```

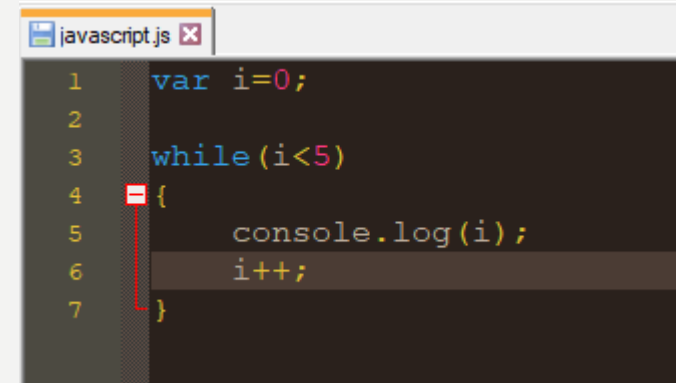
Console >> [Settings] [Filter]

info0101	javascript.js:5
info0102	javascript.js:5
info0103	javascript.js:5

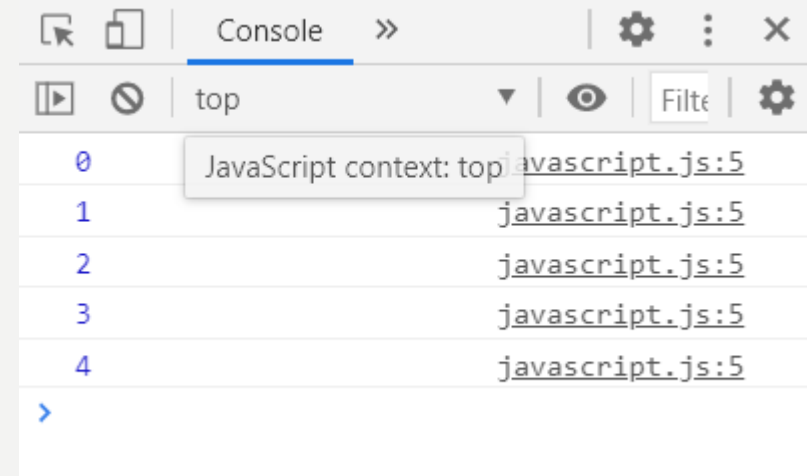
>

BOUCLE TANT QUE

- Mot-clef : while
- Condition de poursuite
- Evolution de la condition de poursuite
- On ne connaît pas à l'avance le nombre d'itération



```
javascript.js
1  var i=0;
2
3  while (i<5)
4  {
5      console.log(i);
6      i++;
7  }
```



Console >> | Settings | Filter

top

0	JavaScript context: top javascript.js:5
1	javascript.js:5
2	javascript.js:5
3	javascript.js:5
4	javascript.js:5
>	

LA BOUCLE INFINIE

- Dérouler un algorithme
- La condition d'arrêt n'est jamais remplie

```
javascript.js x
1  var i=0;
2
3  while(i<5)
4      console.log(i);
```

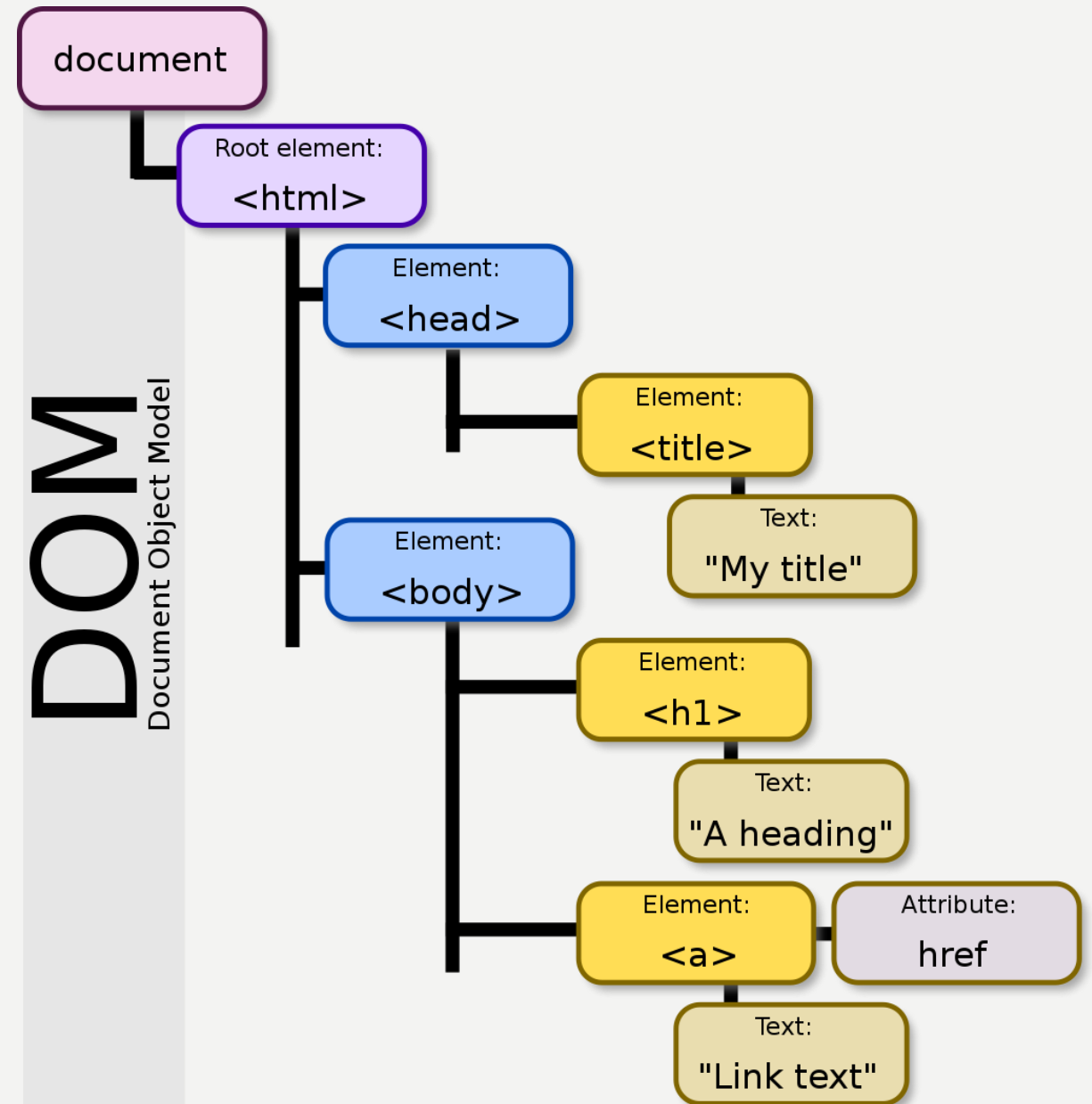
```
javascript.js x
1  var i=0;
2
3  while(true)
4      console.log(i);
```

CHAPITRE 3 :

DOM

ARBRE

- Document Object Model
- Contient tout des éléments html
 - Balise
 - Attributs
 - Texte



Source : wikipedia

LES ACTIONS POSSIBLES

- Changer un élément html
- Changer un attribut html
- Changer un style css
- Supprimer un élément
- Supprimer un attribut
- Ajouter un élément
- Ajouter un attribut

ACCEDER A UN ELEMENT

- Un élément est une node du DOM
- Cette node est composée d'attributs que sont
 - Ses attributs HTML
 - Ses propriétés de style
- Depuis la racine
 - `document.getElementById()` → retour un élément
 - `document.getElementsByTagName()` → retourne une collection d'éléments
 - `document.getElementsByClassName()` → retourne une collection d'éléments
 - `document.querySelectorAll()`; → retourne une collection d'éléments

ACCEDER A UN ELEMENT

- `document.getElementById()`
- Se réfère à l'attribut html *id*

```
index.html x test.html x
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Titre</title>
5     <link rel="stylesheet" href="css/style.css" />
6     <script type="text/javascript" src="js/javascript.js">
7   </script>
8 </head>
9 <body id="body">
10   Du texte
11 </body>
12 </html>
13
```

+

Elements Console >> 1

top Filter Default I

✖ ▶ Uncaught TypeError: Cannot read property 'innerHTML' of null
at javascript.js:3

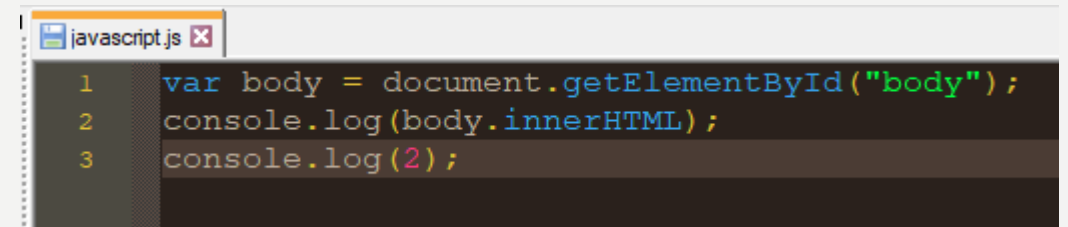
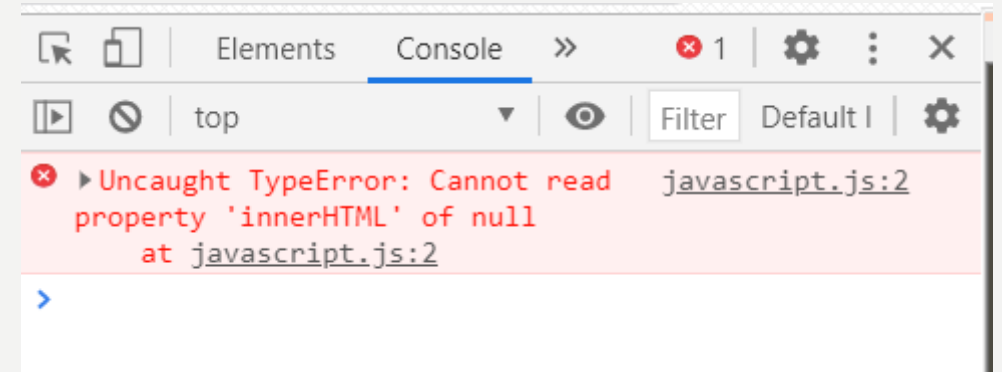
>



```
javascript.js x
1 var body = document.getElementById("body");
2 console.log(body.innerHTML);
```

NOTRE PREMIÈRE ERREUR

- javascript.js:2
- Type de l'erreur : Uncaught TypeError
- Description
- Comprendre l'erreur

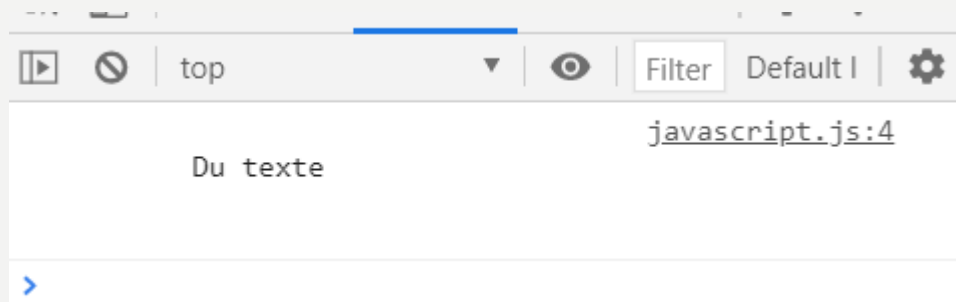


ACCEDER A UN ELEMENT

- `document.getElementById()`
- Se réfère à l'attribut html ***id***

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Titre</title>
5 <link rel="stylesheet" href="css/style.css"
6 <script type="text/javascript" src="js/java
7 </script>
8 </head>
9 <body onload="function()" id="body">
10 Du texte
11 </body>
12 </html>
13
```

+



```
1 function fonction()
2 {
3   var body = document.getElementById("body");
4   console.log(body.innerHTML);
5 }
```

ACCEDER A UN ELEMENT

- `document.getElementsByTagName()` → retourne une collection d'éléments

```
<table>
  <tr>
    <td class="test">a</td>
    <td>b</td>
    <td class="test">c</td>
    <td>d</td>
  </tr>
</table>
```

```
javascript.js style.css new 1
1 function function()
2 {
3   var tds = document.getElementsByTagName("td");
4   console.log(tds.length);
5 }
```

Elements Console >> | ⚙️ ⋮ >

top ▼ | 🔍 Filter Default le ⚙️

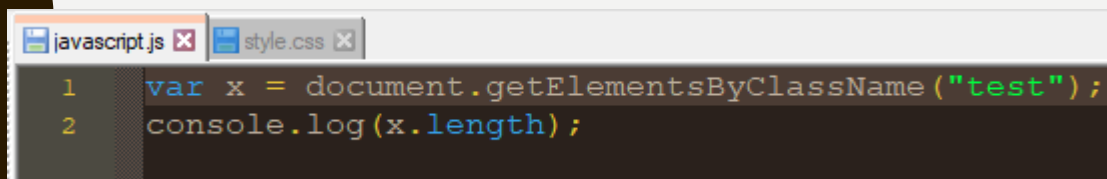
4 javascript.js:4

>

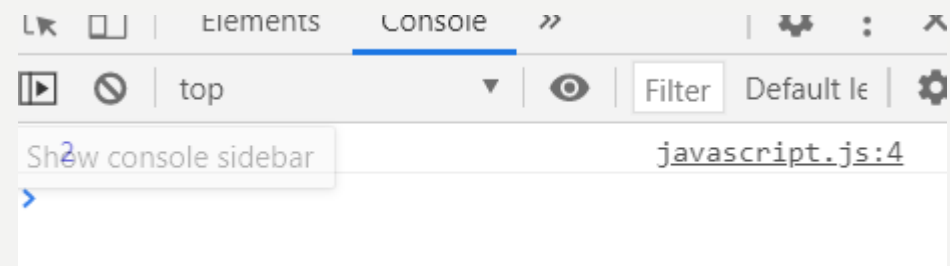
ACCEDER A UN ELEMENT

- `document.getElementsByClassName()` → retourne une collection d'éléments

```
<table>
  <tr>
    <td class="test">a</td>
    <td>b</td>
    <td class="test">c</td>
    <td>d</td>
  </tr>
</table>
```



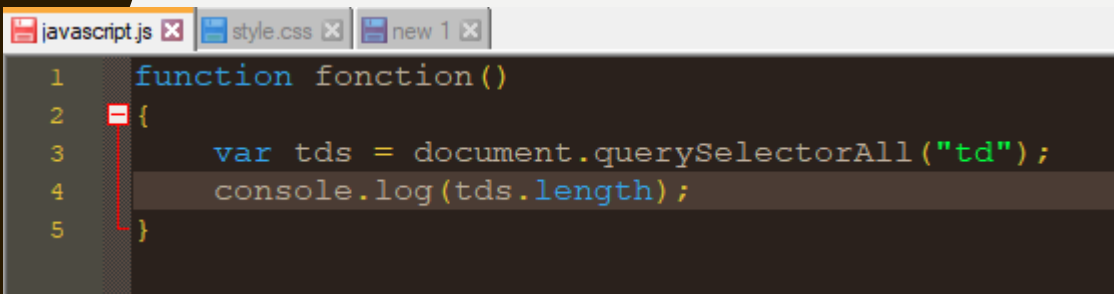
A screenshot of a code editor with two tabs: 'javascript.js' and 'style.css'. The 'javascript.js' tab is active, showing two lines of code: `1 var x = document.getElementsByClassName("test");` and `2 console.log(x.length);`.



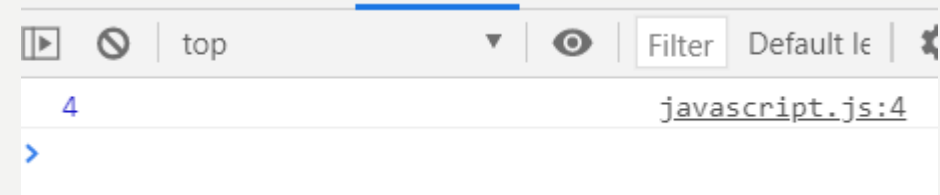
ACCEDER A UN ELEMENT

- `document.querySelectorAll();` ➔ retourne une collection d'éléments
- En paramètre on met un selecteur css

```
<table>
  <tr>
    <td class="test">a</td>
    <td>b</td>
    <td class="test">c</td>
    <td>d</td>
  </tr>
</table>
```



```
javascript.js x style.css x new 1 x
1 function function()
2 {
3   var tds = document.querySelectorAll("td");
4   console.log(tds.length);
5 }
```



```
top Filter Default le
4 javascript.js:4
>
```


EXEMPLE

- Cas d'usage : mettre les valeurs d'un tableau en rouge
 - Boucle
 - Condition
 - Opérations

NAVIGATION

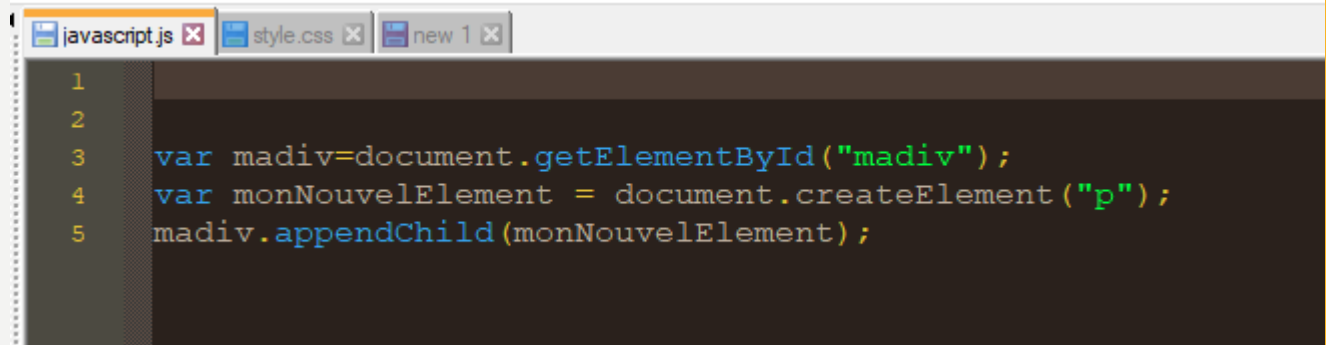
- Ces propriétés s'applique à une node
 - parentNode ⇔ ../ : récupérer le nœud parent
 - childNodes[nodeIndex] : un collection des nœuds enfants
 - firstChild : le premier nœud enfant
 - lastChild : le dernier nœud enfant
 - nextSibling : le nœud enfant suivant
 - previousSibling : le nœud enfant précédent

LES ACTIONS POSSIBLES

- Changer un élément html
- Changer un attribut html
- Changer un style css
- Supprimer un élément
- Supprimer un attribut
- Ajouter un élément
- Ajouter un attribut

```
15 var madiv=document.getElementById("madiv");  
16 madiv.setAttribute("id", "monID");  
17 madiv.setAttribute("required", "");
```

MANIPULATION

A screenshot of a web browser's developer console or a code editor. It shows three tabs: 'javascript.js', 'style.css', and 'new 1'. The 'javascript.js' tab is active and displays five lines of JavaScript code. Line 1 is empty. Line 2 is empty. Line 3 is 'var madiv=document.getElementById("madiv");'. Line 4 is 'var monNouvelElement = document.createElement("p");'. Line 5 is 'madiv.appendChild(monNouvelElement);'.

```
1
2
3 var madiv=document.getElementById("madiv");
4 var monNouvelElement = document.createElement("p");
5 madiv.appendChild(monNouvelElement);
```

- createElement : création d'un nouvel élément
- createTextNode
- appendChild(child) : ajout d'un nœud parmi les enfants
- removeChild(child) : suppression d'un nœud enfant
- remove() : un nœud s'efface lui-même
- replaceChild(c1, c2) : on remplace un enfant par un autre

EXEMPLE

- Calculer les nombres premiers
 - Boucle
- Construire un tableau et mettre les valeurs dedans

CHAPITRE 4 :

LES ÉVÉNEMENTS

LES ÉVÉNEMENTS

- Événements de page et fenêtre
- Événements souris
- Événements clavier
- Événements formulaire

ATTRIBUTION

- Attribut

- onclick par exemple

```
<div onclick="uneFonction();" /></div>
```

- DOM

- Element.onclick = fonction; → SANS LES PARENTHESES

```
22 var element=document.getElementById("maDiv");  
23 element.onclick=maFonction;  
24  
25
```

- EventListener

- addEventListener("click", fonction); → SANS LES PARENTHESES
- removeEventListener

```
21 }  
22 var element=document.getElementById("maDiv");  
23 element.addEventListener("click", maFonction);  
24
```


UNE LISTE D'ÉVÉNEMENTS

• Événements page et fenêtre

- `onabort` — s'il y a une interruption de chargement
- `onerror` — en cas d'erreur pendant le chargement de la page
- `onload` — après la fin du chargement de la page
- `onbeforeunload` — se produit juste avant de décharger la page en cours (par changement de page, en quittant)
- `onunload` — se produit lors du déchargement de la page (par changement de page, en quittant)
- `onresize` — quand la fenêtre est redimensionnée

• Événements souris

- `onclick` — sur un simple clic
- `ondblclick` — sur un double clic
- `onmousedown` — lorsque le bouton de la souris est enfoncé, sans forcément le relâcher
- `onmousemove` — lorsque le curseur est déplacé
- `onmouseout` — lorsque le curseur sort de l'élément
- `onmouseover` — lorsque le curseur se trouve sur l'élément
- `onmouseup` — lorsque le bouton de la souris est relâché
- `onscroll` — lorsque le scroll de la souris est utilisé

• Événements clavier

- `onkeydown` — lorsqu'une touche est enfoncée
- `onkeypress` — lorsqu'une touche est pressée et relâchée
- `onkeyup` — lorsqu'une touche est relâchée

• Événements formulaire

- `onblur` — à la perte du focus
- `onchange` — à la perte du focus, si la valeur a changé
- `onfocus` — lorsque l'élément obtient le focus (ou devient actif)
- `onreset` — lors de la remise à zéro du formulaire (via un bouton ou une fonction `reset()`)
- `onselect` — quand du texte est sélectionné
- `onsubmit` — quand le formulaire est validé (via un bouton ou une fonction `submit()`)

CHAPITRE 5 :

JS AVANCÉ

LOCATION

- Opération sur l'URL
 - Target un élément qui a l'id "test"
 - Naviguer vers une ancre dans la page qui a le *name* "test"

```
22  
23 <location.hash="#test" />
```

- Changer de page

```
21 ,  
22  
23 <location.href="index.html" />
```

INTERVAL

- setInterval : déclenche une fonction toutes les X ms
 - Fonction anonyme
- setTimeout : déclenche une fonction après un certain temps
- clearInterval : annule un interval (nécessite d'avoir récupéré ce dernier dans une variable)

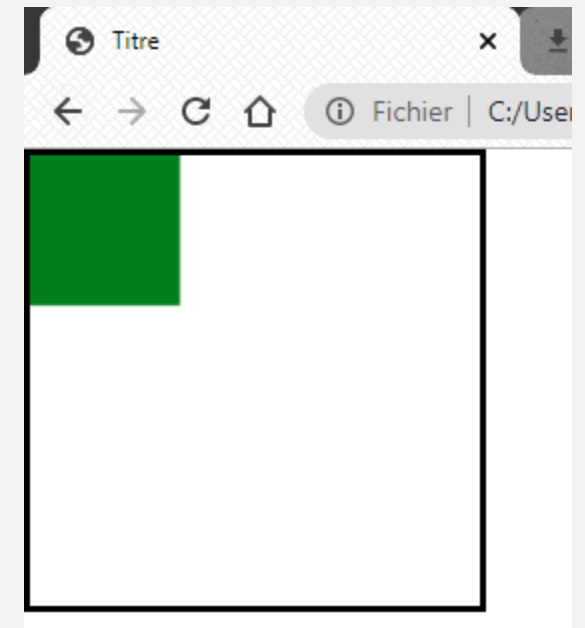
```
3
4  var x = 1;
5
6  var monInterval = setInterval(function() {
7      if (x <= 3)
8          console.log(x);
9      else
10         clearInterval(monInterval);
11         x++;
12     }
13     , 3000);
14
```

CANVAS

- Récupération du canvas
- Récupération de son contexte
 - On donne une couleur au contexte : chaque dessin effectué aura la couleur en cours dans le contexte
 - On dessine ensuite une forme : trait, rectangle, cercle, etc.

```
<canvas id="canvas" width="150" height="150">  
  <p>Canvas pas supporté</p>  
</canvas>
```

```
24 var canvas = document.querySelector('#canvas');  
25 var context = canvas.getContext('2d');  
26  
27 context.fillStyle = "green";  
28 context.fillRect(0, 0, 50, 50);
```



MATH

- Bibliothèque native au javascript qui contient toutes les opérations standards
 - Trigonométrie : `cos()`, `sin()`, `tan()`
 - Le `random()` : retourne une valeur entre 0 et 1
 - Pour avoir une valeur entre x et y : `Math.random()*(y-x)+x`
 - Les arrondis : `round()`, `ceil()`
 - `Min()`, `max()`

COOKIE

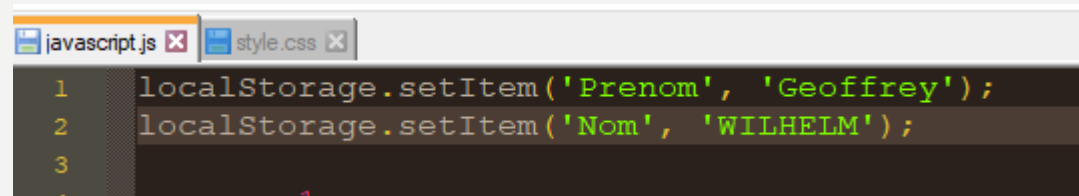
- Document.cookie
- Pair de valeur
 - Identifiant
 - Value
- Séparé par un ;

```
29  
30 document.cookie = "prénom=Geoffrey;nom=WILHELM";
```

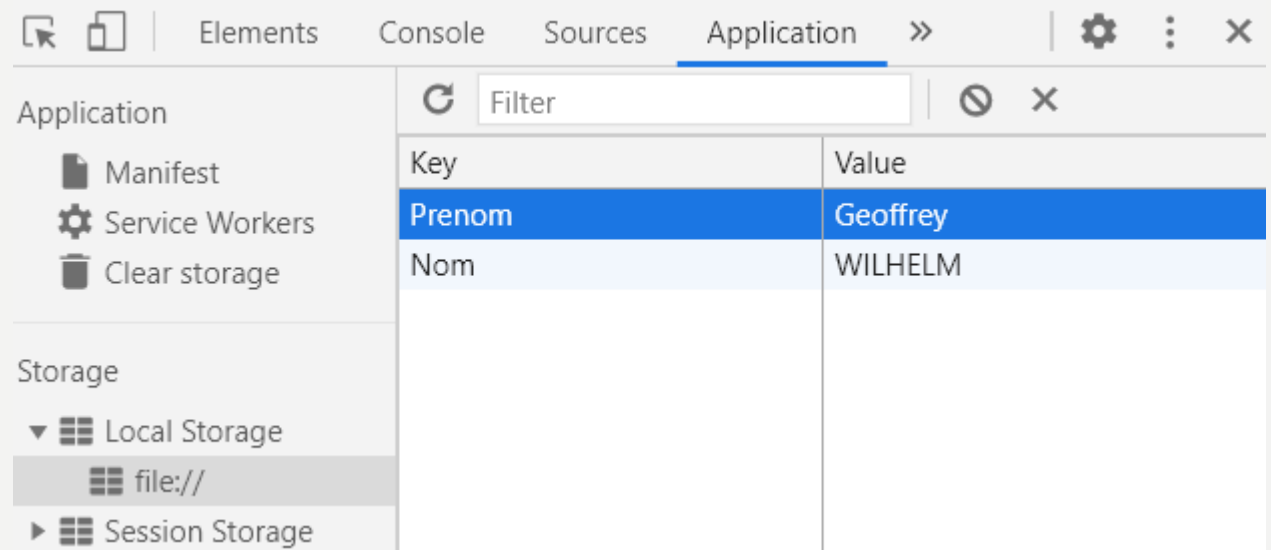
```
31  
32 var prenom = document.cookie.split(";")[0].split(";")[0]  
33 console.log(prenom);
```

LOCAL STORAGE

- localStorage
 - setItem
 - Nom
 - Valeur
 - getItem
 - Nom
- sessionStorage
 - IDEM



```
1 localStorage.setItem('Prenom', 'Geoffrey');  
2 localStorage.setItem('Nom', 'WILHELM');  
3  
4
```

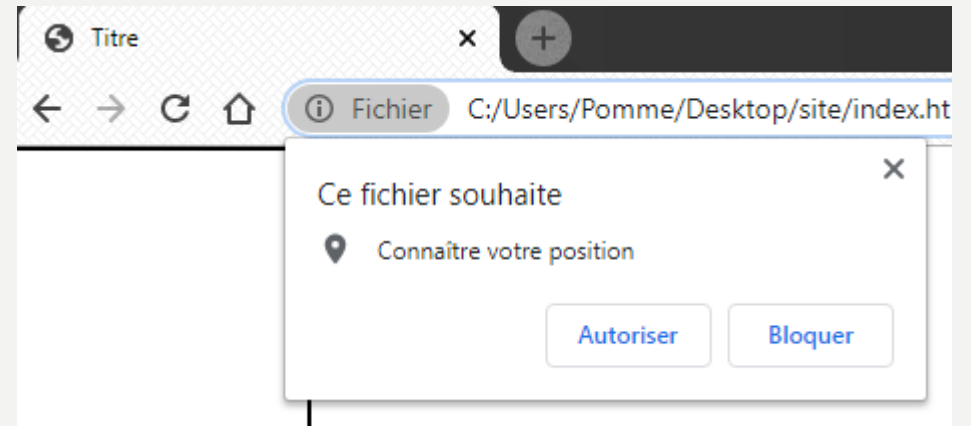


Application	Filter	Key	Value
Manifest		Prenom	Geoffrey
Service Workers		Nom	WILHELM
Clear storage			
Storage			
Local Storage			
file://			
Session Storage			

GEOLOCALISATION

- `Navigator.getLocation`
- `getPosition()`
- `watchPosition()`
- `clearWatch`
- Afficher une map google api
 - Nécessiter de créer une clef sur le site
 - La version libre limite le nombre de requêtes possibles

GEOLOCALISATION



```
1 var mdiv=document.getElementById("mdiv");
2 function getLocation() {
3   if (navigator.geolocation) {
4     navigator.geolocation.getCurrentPosition(print);
5   } else {
6     console.log("pas de geoloc");
7   }
8 }
9 function print(position) {
10   mdiv.innerHTML = "latitude: " + position.coords.latitude +
11     "<br />longitude: " + position.coords.longitude;
12 }
13 getLocation();
```

latitude: 49.2634112
longitude: 4.0173568

HISTORY

- Permet de naviguer dans l'historique du navigateur
- Back() : retour à la page précédente
- Forward() : aller à la page suivante
- Go(Number) :
 - Number = -2 : retour de deux pages en arrières
 - Number = 4 : aller 4 pages en avant

ALLER PLUS LOIN

- Interagir avec un script php grâce à AJAX
- JQuery : pour simplifier un certain nombre d'opérations