Javascript Bootcamp

Sommaire

Javascript: les Bases

C’est quoi javascript

- Un langage de programmation
- Haut niveau
  - On ne s’intéresse pas aux détails (gestion de mémoire, processus ...)
- Orienté Objet
  - Toutes sorte de donnée est stockée sous forme d’objet
- Multiparadigm
  - Programmation Procedurale
  - Programmation Fonctionnelle
  - Programmation Orienté Objet

Pourquoi javascript

Pourquoi javascript

- Ajout d'effet dynamique aux pages
- Créer des applications web pour le navigateur
  - React js
  - Angular js
  - Vue js
- Créer des application web côté serveur
  - Node js
- Créer des application Mobile
  - React native
  - Ionic
- On peut même créer des application desktop
  - Electron js

Javascript Historique

1995- Brendan Eich chez Netscape à créer la première version de javascript nommée Mocha

1996- Changement du nom Mocha vers live script puis javascript

1996- Microsoft lance IE et crée une copie de javascript pour son navigateur

1997- ECMA lance le premier standard javascript Ecmascript1 (ES1).

2009- ES5

2015- ES6 (Changements majeurs)

2016- ES7 ...

Ou insérer un code js?

1. Inline

```
<button onclick="alert('Button clicked!!!')">Click Me</button>
```

2. Dans une balise <script>

```
<script>                                                    >
        alert("Bonjour, je suis un script embarqué")
</script>
```

3. Dans un fichier externe

```
<script src="myScript.js"></script>
```

Les variables

- Une variable est un conteneur servant à stocker des informations de manière temporaire.
- Pour déclarer une variable en JavaScript, on utilise le mot clé ***var*** ou le mot clé ***let***.

Les variables

- Le nom d’une variable doit obligatoirement **commencer par une lettre** ou un **underscore** (_) et ne doit pas commencer par un chiffre ;
- Le nom d’une variable ne doit contenir que des lettres, des chiffres et des underscores mais pas de caractères spéciaux ;
- Le nom d’une variable **ne doit pas contenir d’espace**.
- En JavaScript le nom des variables est **sensible à la casse**.

Les types de données en JavaScript

- En JavaScript, il existe 8 types de données:
  - ***String***
  - ***Number***
  - ***Boolean***
  - ***Null***
  - ***Undefined***
  - ***Symbol***
  - ***BigInt***
- En JavaScript on n’a pas besoin de préciser à priori le type de valeur qu’une variable va pouvoir stocker.
- on peut utiliser la fonction ***typeof(variable)***pour vérifier le type d’une variable

Var, let et const

- Avec ***var*** on peut effectuer des manipulations en haut du code et la déclarer en fin de code car le JavaScript va traiter les déclarations de variables effectuées avec var avant le reste du code JavaScript. (remontée)

  nom=”yasser”;

  Var nom;

- les variables utilisant la syntaxe **let** doivent obligatoirement être déclarées avant de pouvoir être utilisées.

  let nom;

  nom=”yasser”;

Var, let et const

- Une constante est similaire à une variable. Cependant, à la différence des variables, on ne va pas pouvoir modifier la valeur d'une constante.
- Pour créer ou déclarer une constante en JavaScript, nous allons utiliser le mot clef ***const***.

  const pi = 3.14;

  const planete = 'Mars';

Var, let et const

- Avec **var**, on a le droit de déclarer plusieurs fois une même variable
- ***let*** et **const** n'autorisent pas cela.
- Toujours utiliser le mot clé ***let*** pour déclarer vos variables.

Les opérateurs arithmétiques

Opérateur	Nom de l'opération associée
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

Les opérateurs d'affectation

Opérateur	Nom de l'opération associée
+=	Additionne puis affecte le résultat
-=	Soustrait puis affecte le résultat
*=	Multiplie puis affecte le résultat
/=	Divise puis affecte le résultat
%=	Modulo (reste d'une division euclidienne)

Opérateurs de comparaison

Opérateur	Définition
==	Permet de tester l'égalité sur les valeurs
===	Permet de tester l'égalité en termes de valeurs et de types
!=	Permet de tester la différence en valeurs
<>	Permet également de tester la différence en valeurs
!==	Permet de tester la différence en valeurs
<	Permet de tester si une valeur est strictement inférieure à une autre
>	Permet de tester si une valeur est strictement supérieure à une autre
<=	Permet de tester si une valeur est inférieure ou égale à une autre
>=	Permet de tester si une valeur est supérieure ou égale à une autre

La concaténation

- La concaténation est un mot généralement utilisé pour désigner le fait de rassembler deux chaînes de caractères pour en former une nouvelle.
- En JavaScript, l'opérateur de concaténation est le signe +.
  - Lorsque le signe + est utilisé avec deux ***nombres***, il sert à les ***additionner***.
  - Lorsqu'il est utilisé avec autre chose que deux nombres, il sert d'opérateur de concaténation.
  - Si on utilise l'opérateur + pour concaténer une ***chaîne*** de caractères puis un ***nombre***, alors le JavaScript va considérer le nombre comme une ***chaîne*** de caractères.

let x=4+'3';

let y="salut"+'tous';

Les littéraux de gabarits

- En javascript les chaînes sont toujours entourés avec des apostrophes ou des guillemets droits
- On peut aussi utiliser les accents graves **`**.
- Toute expression placée entre les accents graves va être interprétée en JavaScript. **Mais** il va falloir placer les expressions entre ***${*** et ***}***.

```
let age =20;

console.log(`j'ai ${age} ans);
```

- l'utilisation des littéraux de gabarits conserve les retours à la ligne et les décalages dans le résultat final.

Structures de contrôle

- On appelle « structure de contrôle » un ensemble d'instructions qui permet de contrôler l'exécution du code.
- Il existe deux grands types de structure de contrôle de base qu'on retrouve dans la plupart des langages informatiques et notamment en JavaScript :
    - ***Les structures de contrôle conditionnelles*** (ou plus simplement les « conditions »)
    - ***Les structures de contrôle de boucles*** (ou plus simplement les « boucles »).

Structures de contrôle conditionnelles

- Les structures de contrôle conditionnelles nous permettent d'exécuter une série d'instructions si une condition donnée est vérifiée ou une autre série d'instructions si elle ne l'est pas.
- Nous avons accès aux structures conditionnelles suivantes en JavaScript :
  - La condition ***if(test)*** ;
  - La condition ***if(test)... else*** ;
  - La condition ***if(test1)... elseif(test2)... else***.

Structures de contrôle conditionnelles

- La condition ***if(test)*** va juste nous permettre d'exécuter un bloc de code si et seulement si le résultat d'un ***test*** vaut ***true***.
- Toute valeur évaluée par le JavaScript dans un contexte booléen va être évaluée à true à l'exception des valeurs suivantes:
    - Le booléen ***false*** ;
    - La valeur ***0*** ;
    - Une ***chaîne de caractères vide*** ;
    - La valeur ***null*** ;
    - La valeur ***undefined*** ;
    - La valeur ***NaN*** (« Not a Number » = « n'est pas un nombre »).

Structures de contrôle conditionnelles: ternaires

- Les structures conditionnelles ***ternaires*** correspondent à une autre façon d'écrire nos conditions en utilisant une syntaxe basée sur l'opérateur ternaire ***? :***.
- Les structures ternaires vont se présenter sous la forme suivante :

test ? code à exécuter si true : code à exécuter si false;

x>10 ? console.log('x>10') : console.log('x<=10');

Structures de contrôle conditionnelles: switch

```
let x=2;

switch(x){
        case val1 :  instructions1; break;
        case val2 :  instructions2; break;
        case val3 :  instructions3; break;
        Default :  instructions4;
}
```

Structures de contrôle boucle

- Les boucles vont nous permettre d'exécuter plusieurs fois un bloc de code, tant qu'une condition donnée est vérifiée.
- Nous disposons de six boucles différentes en JavaScript :
  - La boucle ***while*** (« tant que ») ;
  - La boucle ***do... while*** (« faire... tant que ») ;
  - La boucle ***for*** (« pour ») ;
  - La boucle ***for... in*** (« pour... dans») ;
  - La boucle ***for... of*** (« pour... parmi ») ;

Structures de contrôle boucle

Les boucles se composent de trois choses :

- Une valeur de départ pour initialiser la boucle et nous servir de compteur;
- Un test de sortie qui précise le critère de sortie de la boucle ;
- Un itérateur qui va modifier la valeur de départ de la boucle à chaque nouveau passage jusqu’au moment où la condition de sortie est vérifiée.

Les fonctions

- Une fonction correspond à un bloc de code nommé et réutilisable et dont le but est d'effectuer une tâche précise.
- le code d'une fonction est réutilisable : cela veut dire qu'on va pouvoir appeler une même fonction autant de fois qu'on le souhaite afin qu'elle accomplisse plusieurs fois la même opération.
- Pour exécuter le code d'une fonction, il suffit de l'appeler. Pour faire cela, on n'a qu'à écrire le nom de la fonction suivi d'un couple de parenthèses et préciser des arguments entre les parenthèses.

Les fonctions

Pour définir une fonction, on va utiliser le mot clé ***function***

```
function maFonction(arg1, arg2, ...){
    Instruction1;
    Instruction2;
    return maVar;
}
```

Portée des variables (scop)

- La « portée » d'une variable désigne l'espace du script depuis lequelle elle va être accessible.
- En JavaScript, il existe trois espaces de portée différents :
  - ***l'espace global***
  - ***L'espace fonction***
  - ***L'espace block***

Portée des variables (scop): var vs let

- Lorsqu'on utilise ***let*** pour définir une variable à l'intérieur d'une fonction en JavaScript, la variable va avoir une portée dite « ***de bloc*** » : la variable sera accessible dans le ***bloc*** dans lequel elle a été définie et dans les blocs que le bloc contient.
- Une variable avec le mot clé ***var*** dans une fonction aura une portée élargie puisque cette variable sera accessible dans tous les blocs de la fonction.

Fonctions anonymes

- Les fonctions anonymes sont des fonctions qui ne possèdent pas de nom.
- On utilise les fonctions anonymes lorsqu'on n'a pas besoin d'appeler notre fonction par son nom c'est-à-dire lorsque le code de notre fonction n'est appelé qu'à un endroit dans notre script.
- On crée une fonction anonyme de la même façon qu'une fonction classique, en utilisant le mot clé ***function*** mais en ***omettant le nom*** de la fonction après.

```
function(arg1, arg2, ...){
        ....
}
```

Fonctions anonymes

Pour exécuter une fonction anonyme :

1. Enfermer le code de notre fonction dans une variable et utiliser la variable comme une fonction ;

```
let f=function(){...};   // definition
f();  // execution
```

2. Auto-invoquer la fonction anonyme ;

```
(function(){...})();  //definition et execution
```

3. Utiliser un ***événement*** pour déclencher l'exécution de notre fonction.

Les fonctions récursives

- Une fonction récursive est une fonction qui va s'appeler elle-même au sein de son code.
- Les fonctions récursives vont nous permettre d'exécuter une action en boucle et jusqu'à ce qu'une certaine condition de sortie soit vérifiée.

```
Function decompt(n){
        if(n>0){
                console.log(n);
                Return decompt(n-1)
        }
}
```

La POO en Javascript

POO en javascript

- JavaScript est un langage qui intègre l'orienté objet dans sa définition ce qui fait que tous les éléments du JavaScript vont soit être des objets, soit pouvoir être convertis et traités comme des objets.
- Un objet est un conteneur qui posséde un ensemble de ***propriétés*** et de ***méthodes***.

POO en javascript: objet littéral

```
let personne={
    nom: “yahyaoui”,
    age: 20,
    direBonjour: function(){ console.log(“bonjour!!”);}
};
personne.direBonjour();// ou bien personne[“direBonjour”]();
personne.age = 23; //ou bien personne[“age”] = 23;
pesonne.prenom = “Yasser”;
```

POO en javascript: constructeur d'objets

- En javascript on peut créer d'objet à l'aide de constructeur d'objets qui n'est autre qu'une fonction constructeur.
- Pour construire des objets à partir d'une fonction constructeur, nous allons devoir suivre deux étapes :
  - Définir la fonction constructeur
  - Appeler ce constructeur à l'aide du mot clé ***new***.

POO en javascript: constructeur d'objets

- Dans une fonction constructeur, on définit un ensemble de propriétés et de méthodes.
- Les objets créés à partir de ce constructeur possèdent automatiquement les propriétés et méthodes définies dans le constructeur.

POO en javascript: constructeur d'objets

```
function Utilisateur(n, a, m){
    this.nom = n;
    this.age = a;
    this.mail = m;
    this.bonjour = function(){
        console.log('Bonjour, je suis ' + this.nom + ',
        j\'ai ' + this.age + ' ans');
    }
}
Let user1=new Utilisateur("Adnane",24, "adnane@gmail.com");
Let user2=new Utilisateur("yasser",22, "yasser@gmail.com");
user1.tail = 178;  // on peut attribuer d'autre propriétés à l'objet
```

POO en javascript: prototype

- En utilisant le constructeur plusieurs fois, on va ***copier*** à chaque fois la méthode ***bonjour()*** qui est ***identique*** pour chaque objet.
- l'idéal serait de ne définir cette méthode qu'une ***seule fois*** et que chaque objet puisse l'utiliser lorsqu'il le souhaite.

  ⇒ Il faut utiliser les ***prototypes***.

POO en javascript: prototype

Il existe deux grands types de langages orientés objet :

1. ceux basés sur les **classes**,
2. et ceux basés sur les **prototypes**.

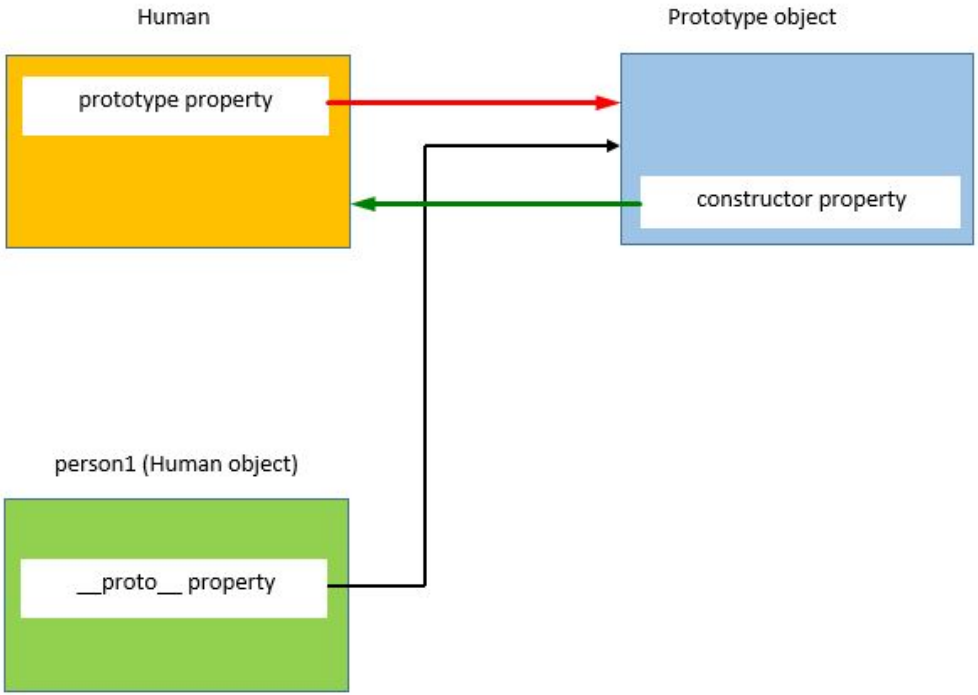
POO en javascript: prototype

1. Dans les langages orientés objet ***basés sur les classes***, tous les objets sont ***créés à partir de classes*** et vont hériter des propriétés et des méthodes définies dans la classe.
2. Dans les langages orientés objet ***utilisant des prototypes*** comme le JavaScript, ***tout est objet*** et il ***n'existe pas de classes*** et l'héritage va se faire au moyen de prototypes.

POO en javascript: prototype

1. Les ***fonctions*** en JavaScript ***sont des objets***.
2. A la création d'une fonction, JavaScript va ***automatiquement*** lui ajouter une ***propriété prototype*** qui n'est utile que lorsque la fonction est utilisée comme constructeur.

POO en javascript: prototype



POO en javascript: prototype

- Le contenu de la propriété **prototype** d'un constructeur va être partagé par tous les objets créés à partir de ce constructeur.
- Comme cette propriété est un objet, on va pouvoir lui ajouter des propriétés et des méthodes que tous les objets créés à partir du constructeur vont partager.
- Les objets créés à partir du constructeur ***ne possèdent pas*** vraiment les ***propriétés*** et les ***méthodes*** définies dans la propriété ***prototype*** du constructeur mais vont ***pouvoir y accéder*** et se « partager » ces membres définis dans l'objet **prototype** du constructeur.

POO en javascript: prototype

- Définir des propriétés et des méthodes dans le prototype d'un constructeur permet de les rendre accessibles à tous les objets créés à partir de ce constructeur sans que ces objets aient à les redéfinir.
- Généralement on définit les ***propriétés*** des objets au sein du ***constructeur*** et les ***méthodes*** dans le ***prototype*** du constructeur.
- Les différents objets se « partagent » les mêmes propriétés et les mêmes méthodes définies dans le constructeur.

La chaîne des prototypes

Lorsqu'on essaie d'accéder à un ***membre d'un objet***, le navigateur va chercher ce membre

1. au sein de l'objet
2. au sein de la propriété ***__proto__*** (***prototype*** du constructeur)
3. ***__proto__*** du constructeur (***prototype*** du constructeur du constructeur).
4. Au sein de l'objet ***Object***.

Object permet aussi de créer des objets génériques vides grâce à la syntaxe ***new Object()***.

Les classes

- dans ses dernières versions, JavaScript a introduit le mot clé ***class***
- JavaScript va ***convertir*** nos « ***classes*** » selon son modèle ***prototypes***.
- Les classes JavaScript ne sont qu'une nouvelle syntaxe pour les gens plus habitués à travailler avec des langages orientés objet basés sur les classes.

Les classes

```
class Ligne{
    constructor(nom, longueur){
        this.nom = nom;
        this.longueur = longueur;
    }
    taille(){
        console.log( 'Longueur de'+this.nom+':'+this.longueur )
    };
}
let geo1 = new Ligne('geo1', 10);
let geo2 = new Ligne('geo2', 5);
geo1.taille();
geo2.taille();
```

POO en javascript: Classes étendues et héritage

```
class Rectangle extends Ligne{
    constructor(nom, longueur, largeur){
        super(nom, longueur); //Appelle le constructeur parent
        this.largeur = largeur;
    }
    aire(){
        console.log('Aire de'+this.nom+':'
        +this.longueur*this.largeur)
    };
}
let geo3 = new Rectangle('geo3', 7, 5);
geo3.aire();
geo3.taille();
```

Valeurs primitives et objets prédéfinis

- Le JavaScript possède deux grandes catégories de types de données : les valeurs ***primitives*** et les ***objets***.
- On appelle valeur primitive en JavaScript une valeur qui n'est pas un objet et qui ne peut pas être modifiée.
- les valeurs primitives sont passées et comparées par valeur tandis que les objets sont passés et comparés par référence.
- Si deux valeurs primitives ont la même valeur, elles vont être considérées égales.
- Si deux objets définissent les mêmes propriétés et méthodes avec les mêmes valeurs, ils ne vont pas être égaux. Pour que deux objets soient égaux, il faut que les deux fassent référence aux mêmes membres.

Valeurs primitives et objets prédéfinis

Chaque type de valeur primitive, à l'exception de ***null*** et de ***undfenied***, possède un équivalent objet prédéfini en JavaScript.

```
let s1="Bonjour";

let s2= new String("Bonjour");

console.log(typeof s1); // string

console.log(typeof s2); // object
```

L'objet String

- Propriété
    - length
    - prototype
- Methodes
    - includes()
    - startsWith(), endsWith()
    - substring(), slice()
    - indexOf(), lastIndexOf()
    - replace()
    - trim()
    - toLowerCase(), toApperCase()
    - match, matchAll() et search()

L'objet Number

- Propriété
  - MIN_VALUE, MAX_VALUE
  - MIN_SAFE_INTEGER, MAX_SAFE_INTEGER
  - NEGATIVE_INFINITY, POSITIVE_INFINITY
  - NaN
- Methodes
  - isFinite(), isInteger(), isNaN()
  - isSafeInteger()
  - parseInt(), parseFloat()
  - toFixed(), toString()

L'objet Math

- Propriété
    - Math.E, Math.pi, Math.SQRT2
- Methodes
    - floor(), ceil(), round() et trunc()
    - random()
    - min(), max()
    - abs()
    - cos(), sin(), tan(), acos(), asin() et atan()
    - exp() et log()

Les tableaux: Array

```
let prenoms = ['yasser', 'adnane', 'anas', 'hiba'];
let ages = [29, 27, 29, 30];
let produits = ['Livre', 20, 'Ordinateur', 5, ['Magnets',
100]];
console.log(prenom[0])
for(let valeur of prenoms){
   console.log(valeur);
}
```

Les tableaux: Array

```
let personne = {
    'prenom' : 'yasser',
    'age' : 29,
    'sport' : 'trail',
    'cours' : ['HTML', ' CSS', ' JavaScript']
};
for(let p in personne){
    console.log(personne[p]);
}
```

Les tableaux: Array

- Propriété
    - Length
    - prototype
- Methodes
    - Push(), pop()
    - unshift() et shift()
    - splice()
    - slice(), join()
    - concat()
    - includes()

L’objet Date

- Plusieur facon pour créer un objet Date:
    - let date = Date();
    - let date = Date.now();
    - let date = new Date();
    - let date = new Date('March 23, 2019 20:00:00');
    - let date = new Date(1553466000000);
    - let date = new Date(2019, 0, 25, 12, 30);
- L’objet date possede plusieur méthodes:
    - getDay() , getDate(), getMonth(), getFullYear(), getHours(), getMinutes() getSeconds() getMilliseconds()

Ecmascript 6 (ES6)

ES6

- Javascript introduit par Netscape
- Puis ECMA International pour la standardisation
- ES6 == EcmaScript 6 (2015), ES7 (2016), ES8(2017)

Le code javascript devient plus simple

- Mais pas supporter par la majorité des navigateur

Il faut passer par un transcompilateur (Babel)

ES6

Javascript

```
var a=function(x,y){
    return x+y;
}
```

ES6

```
const a=(x,y)=>x+y;
```

ES6: destructuring

Javascript

```
const etudiant={
    nom: "Baddi",
    prenom: "Ahmed",
    age: 24
}
const nom=etudiant.nom;
const prenom=etudiant.prenom;
const age=etudiant.age;
```

ES6

```
const etudiant={
    nom: "Baddi",
    prenom: "Ahmed",
    age: 24
}

const {nom, prenom, age}=etudiant;
```

ES6: propriétés des objets

```
1  const a="nom";
2  const b="pre";
3  const etudiant={
4      [a]: "Baddi",
5      [b+a]: "Ahmed",
6      age: 24
7  }
8  console.log(etudiant.prenom);
```

ES6: propriétés des objets

```
1  const nom="Baddi";
2  const prenom="Ahmed";
3  const age=24;
4  const etudiant={
5      nom: nom,
6      prenom: prenom,
7      age: age
8  }
9  //ES6
10 const etudiant={
11     nom,
12     prenom,
13     age
14 }
```

ES6: template Strings

```
1  const nom="Baddi";
2  const prenom="Ahmed";
3  const age=24;
4  let s= "Mon nom est"+nom+"j'ai "+age+"
       ans";
5  //ES6
6  let s=`Mon nom est: ${nom} j'ai ${age}
       ans`;
```

ES6: Arguments par defaut

```
1  function cal(a=0,b=1){
2      return a/b;
3  }
4
5  calc(); // 0
6  calc(4);// 4
7  calc(4,8);// 0.5
```

ES6: Arrow function

```
1  function somme(a,b){
2      return a+b;
3  }
4  //ES6
5  const somme=(a,b)=> {
6      return a+b
7  }
8
9  const somme=(a,b)=> a+b;
```

ES6: Closures

```
1  function f1(){
2      var a="Bonjour";
3      function f2=(){
4          console.log(a);
5      }
6      return f2;
7  }
8  var f=f1();
9  f();
```

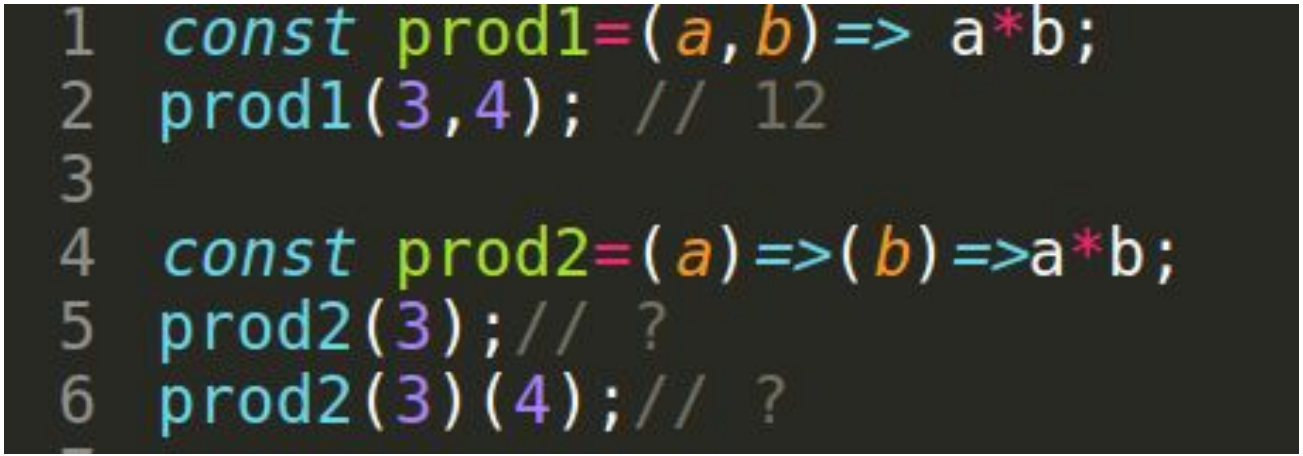
ES6: Closures

```
function f1(){
    var a="Bonjour";
    function f2=(){
        console.log(a);
    }
    return f2;
}
var f=f1();
f();
```

```
//ES6
const f1=()=>{
    const a="Bonjour";
    const f2=()=>{
        console.log(a);
    }
}
var f=f1();
f();
```

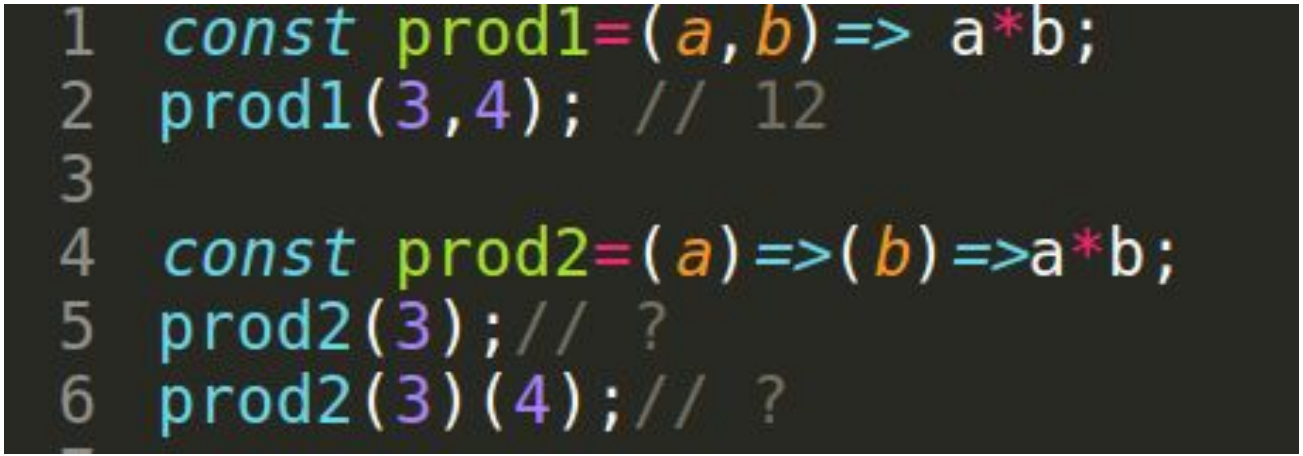
ES6: Currying

Transformer une fonction a plusieurs variable en plusieur fonctions a une seule variable.



ES6: Currying

Transformer une fonction a plusieurs variable en plusieur fonctions a une seule variable.



ES6: Compose

```
1  const comp = (f,g) => (a) => f(g(a));
2  let somme = (n) => n+1;
3  comp(somme,somme)(3);// ?
```

Array

```
1  const a=[1,4,3,8];
2  let b=a.forEach((n)=>{
3      n*2;
4  });
5  console.log(b);// undefined
6  let b=a.map((n)=> n*2);
7  console.log(b);// [2,8,6,16]
8
9  let c=a.filter((n)=> n%2 === 0);
10 console.log(c);// [4,8]
11
12 let d=a.reduce((s,n)=> s+n,2);
13 console.log(d);// 18
14
```

Objets : references

```
1  let a={val: 3};
2  let b=a;
3  let c={val: 3};
4  console.log(a===b);//true
5  console.log(a===c);//false
6  a.val=2;
7  console.log(a.val);//2
8  console.log(b.val);//2
9  console.log(c.val);//3
```

Objets: contexte

```
1  const obj={
2      a:function(){
3          console.log(this);
4      }
5  }
6  obj.a();//obj
7  console.log(this);// window
8  function a(){
9      console.log(this);
10 }
11 a()// window
```

Objets: instanciation

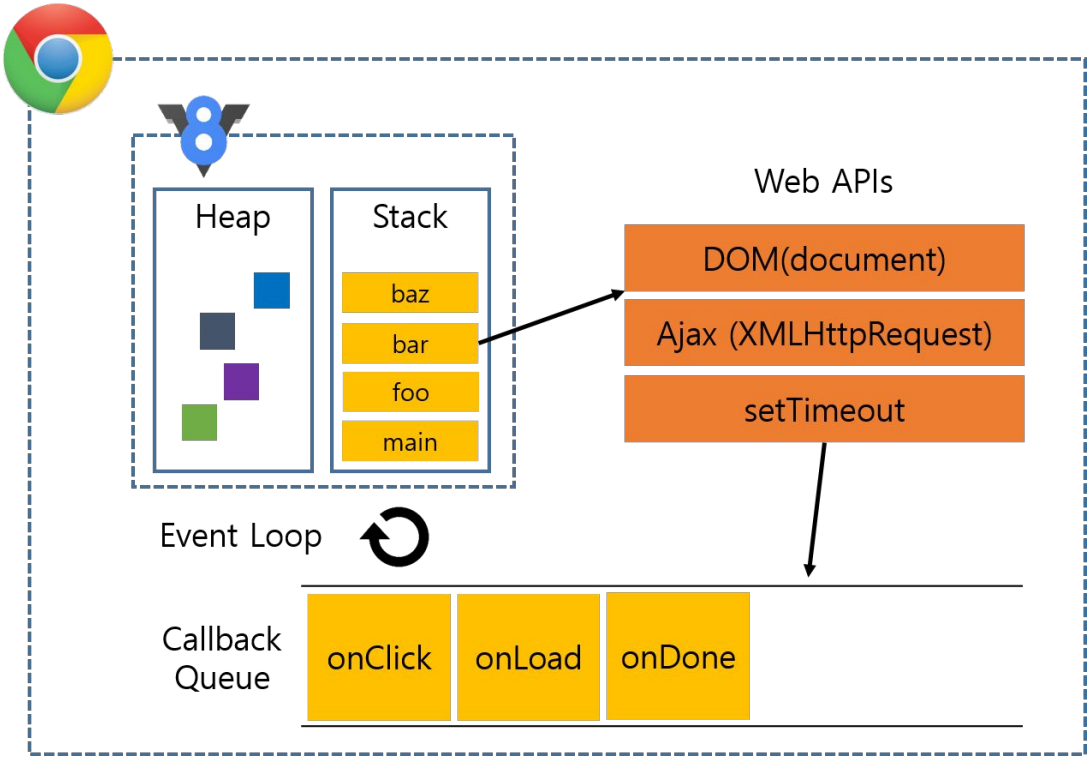
```
class Homme{
    constructor(nam, age){
        this.nom=nom;
        this.age=age;
    }
    hi(){
        console.log(`Je suis ${this.nom}`);
    }
}
let h=new Homme("yakoubi",22);
```

Objets: instanciation

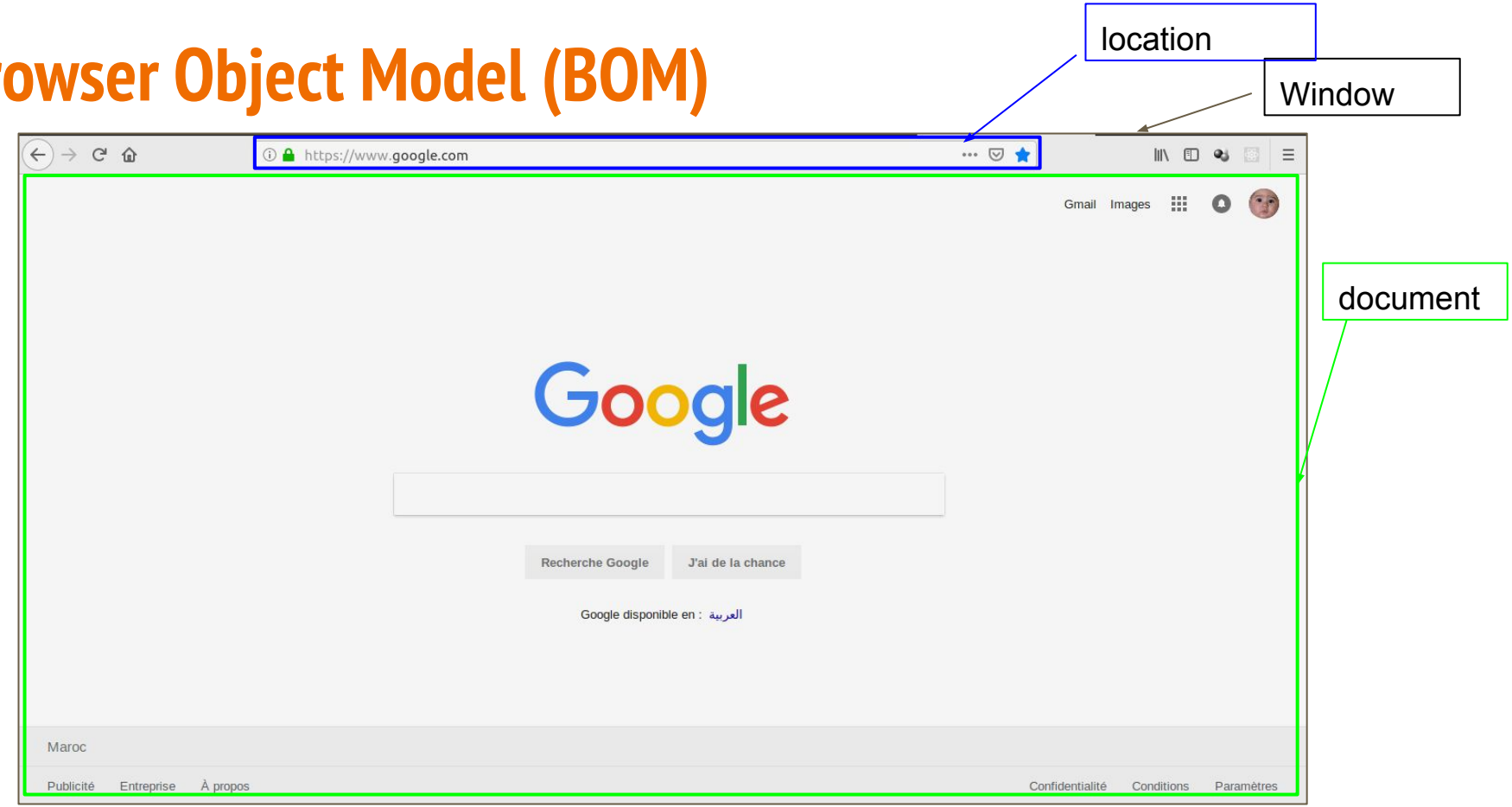
```
class Etudiant extends Homme{
    constructor(nam, age, note){
        super(nom,age);
        this.note=note;
    }
    getNote(){
        console.log(`Note: ${this.note}`);
    }
}
let h=new Homme("yakoubi",22,17);
```

DOM: Document Object Model

C’est quoi le DOM



Browser Object Model (BOM)



Browser Object Model (BOM)

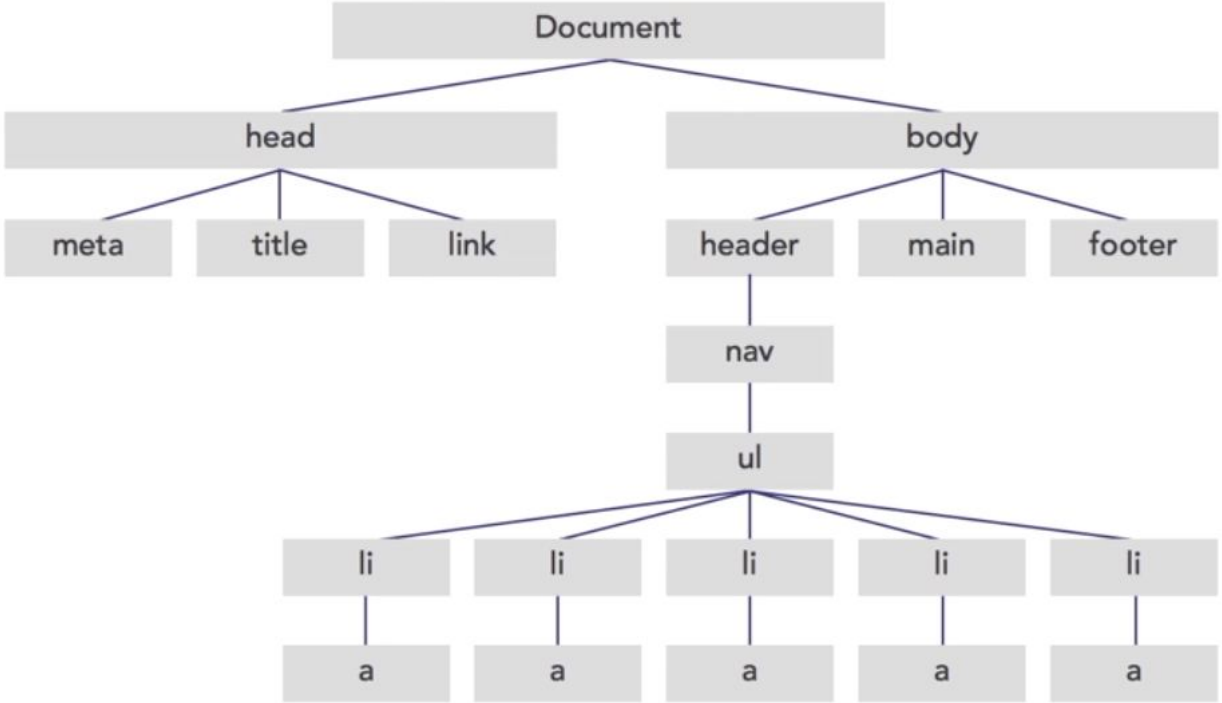
Window est l'objet du plus haut niveau dans le BOM et possède un ensemble de propriétés et méthode pour interagir avec le Browser

- window.innerWidth
- window.open()
- window.location
- Window.**document**
- ....

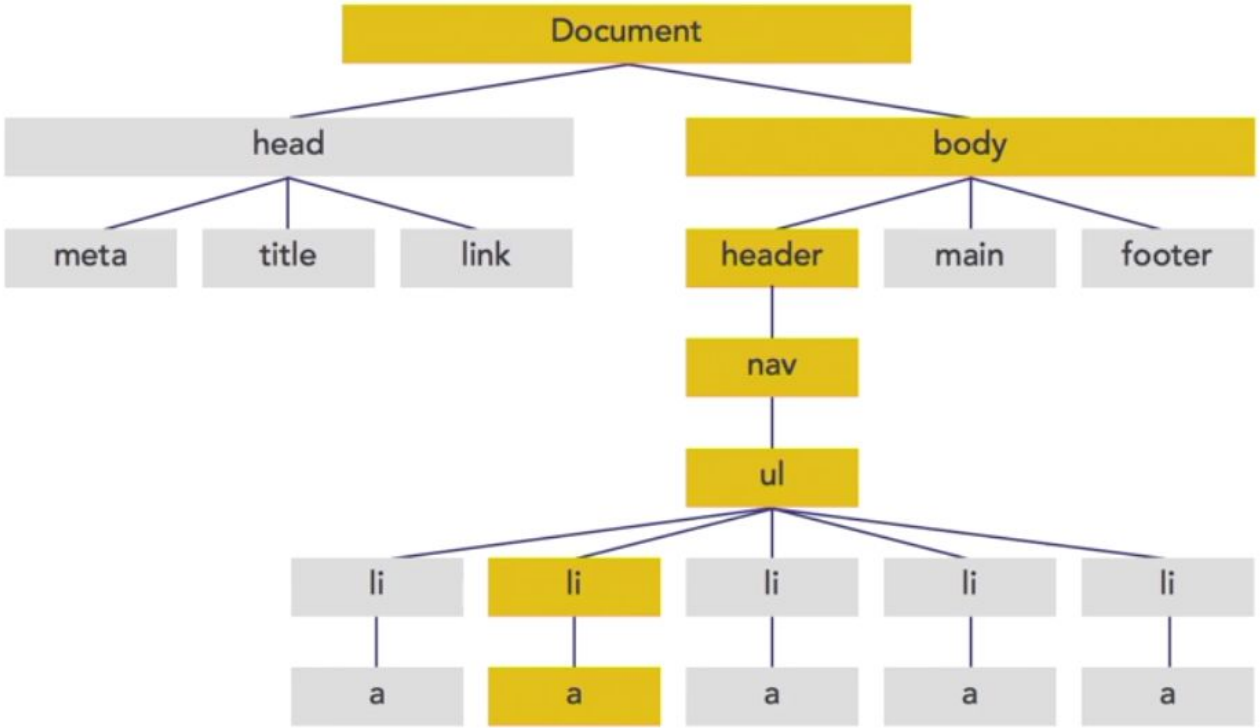
Le Document Object Model (DOM)

- Le DOM est une API qui s'utilise avec les documents HTML, et qui va nous permettre, via le JavaScript, d'accéder au code HTML d'un document.
- C'est grâce au DOM que nous allons pouvoir:
    - ***modifier des éléments HTML (afficher ou masquer un par exemple),***
    - ***ajouter des éléments HTML***
    - ***déplacer des éléments HTML***
    - ***Supprimer des éléments HTML***

Le Document Object Model (DOM)



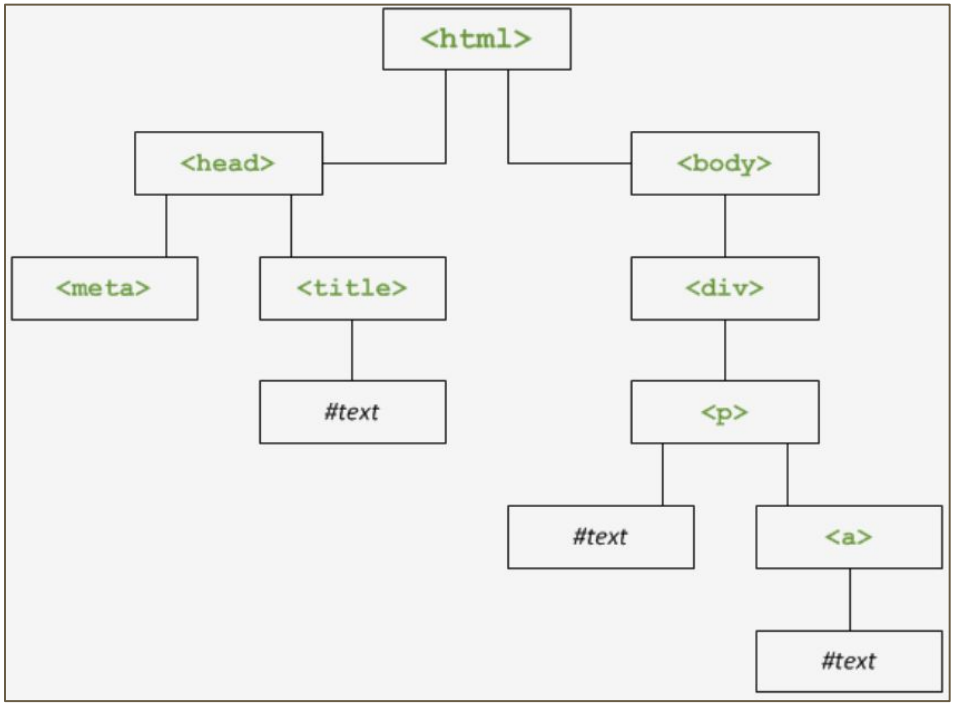
Le Document Object Model (DOM)



Le Document Object Model (DOM)

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
</head>

<body>
    <div>
        <p>Un peu de texte <a>et un lien</a></p>
    </div>
</body>
</html>
```



Accéder aux éléments

- document.body
- Document.title
- document.URL
- document.getElementById(“#id”),
- document.getElementsByTagName(“tagName”)
- document.getElementsByClassName(“.class”)
- document.getElementsByName(“name”)
- ...

Accéder aux éléments

document.querySelector(“css selector”)

=> Retourne Le premier élément qui vérifie le sélecteur css spécifier

document.querySelectorAll(“css selector”)

=> Retourne Tous les éléments vérifiant le sélecteur CSS

Ces deux méthodes sont les plus utilisées

Accéder aux éléments

1. querySelector()
2. querySelectorAll()

```html
<div id="menu">

    <div class="item">
        <span>Élément 1</span>
        <span>Élément 2</span>
    </div>

    <div class="publicite">
        <span>Élément 3</span>
        <span>Élément 4</span>
    </div>

</div>

<div id="contenu">
    <span>Introduction au contenu de la page...</span>
</div>
```

```javascript
var query = document.querySelector('#menu .item span'),
    queryAll = document.querySelectorAll('#menu .item span');

alert(query.innerHTML); // Affiche : "Élément 1"

alert(queryAll.length); // Affiche : "2"
alert(queryAll[0].innerHTML + ' - ' + queryAll[1].innerHTML); // Affiche : "Élément 1
- Élément 2"
```

Accéder aux attributs d'un éléments

Les attributs des élément HTML peuvent êtres accède en lecture/écriture ou en lecture seule

1. element.atributes
2. element.innerHTML
3. element.outerHTML
4. element.clientHight
5. element.className
6. element.classList
7. element.id
8. ...

Accéder aux attributs d'un éléments

Pour modifier les attributs en lecture seul on utilise de methodes:

1. element.classList.add("une nouvelle classe")
2. element.classList.remove("une classe existante")
3. element.classList.contains("une classe")
4. element.haseAttribute(attribut)
5. element.getAttribute(attribut)
6. element.setAttribute(attibut, value) // modifier ou ajouter un attribut
7. element.removeAttribute(attribut)

Exp: ***document.querySellector("a").setAttribut("target","_blank")***

Accéder aux éléments

getAttribute() et setAttribute()

```
<body>
    <a id="myLink" href="http://www.un_lien_quelconque.com">Un lien modifié
dynamiquement</a>

    <script>
        var link = document.getElementById('myLink');
        var href = link.getAttribute('href'); // On récupère l'attribut « href »

        alert(href);

        link.setAttribute('href', 'http://www.siteduzero.com'); // On édite l'attribut
« href »
    </script>
</body>
```

Accéder aux éléments (Elenment.className)

```
3  <head>
4      <meta charset="utf-8" />
5      <title>Le titre de la page</title>
6      <style>
7          .blue {
8              background: blue;
9              color: white;
10         }
11     </style>
12 </head>
13 <body>
14     <div id="myColoredDiv">
15         <p>Un peu de texte <a>et un lien</a></p>
16     </div>
17     <script>
18         document.getElementById('myColoredDiv').className = 'blue';
19     </script>
20 </body>
```

Accéder aux éléments (Elenment.classList)

```
var div = document.querySelector('div');
// Ajoute une nouvelle classe
div.classList.add('new-class');
// Retire une classe
div.classList.remove('new-class');
// Retire une classe si elle est présente ou bien l'ajoute si elle est absente
div.classList.toggle('toggled-class');
// Indique si une classe est présente ou non
if (div.classList.contains('old-class')) {
    alert('La classe .old-class est présente !');
}
// Parcourt et affiche les classes CSS
var result = '';
for (var i = 0; i < div.classList.length; i++) {
    result += '.' + div.classList[i] + '\n';
}
alert(result);
```

Ajout d'un élément au DOM

1. Creer l'element <= document.createElement()
2. Créer le noeud texte de cet élément <= document.createTextNode()
3. Ajouter le noeud texte à l'élément <= document.appendChild()
4. Ajouter l'élément au DOM <= document.appendChild()

Exercice:

Ajouter un élément <***caption>....</caption>*** a l'élément <***figure ..> ...</figure>***

```
 9      <figure class="mafig">
10          <img class="monImg" src="img.jpeg" alt="un texte
            ">
11      </figure>
```

Ajout d'un élément au DOM

```
1  const fig=document.querySelector(".mafig");
2  const img=fig.querySelector(".monImg");
3  var altTxt=img.getAttribute("alt");
4  var capElmt=document.createElement("figcaption");
5  var capTxt=document.createTextNode(altTxt);
6  capElmt.appendChild(capTxt);
7  fig.appendChild(capElmt);
8  console.log(fig);
```

Une nouvelle methode : .append()

```
1  const fig=document.querySelector(".mafig");
2  const img=fig.querySelector(".monImg");
3  var altTxt=img.getAttribute("alt");
4  var capElmt=document.createElement("figcaption");
5  capElmt.append(altTxt);
6  fig.append(capElmt);
```

Style CSS Inline

Avec l’attribut style on peut ajouter n’import quel propriété CSS à n'importe quel élément.

- ***Element.style;*** => **uniquement** le **Inline** CSS de l'élément {attribut:“value”, …. } mais pas les autre style définie dans des fichier css au dans le head.
- ***Element.style.color=”blue”;***
- ***Element.style.backgroundColor=”yellow”;*** //**backgroundColor** pas **background-color**
- ***Element.style.cssText=”color: blue; backgound-color: yellow; …”***
- ***Element.style.setAttribute(“style”, ”color: blue; backgound-color: yellow; …”);***

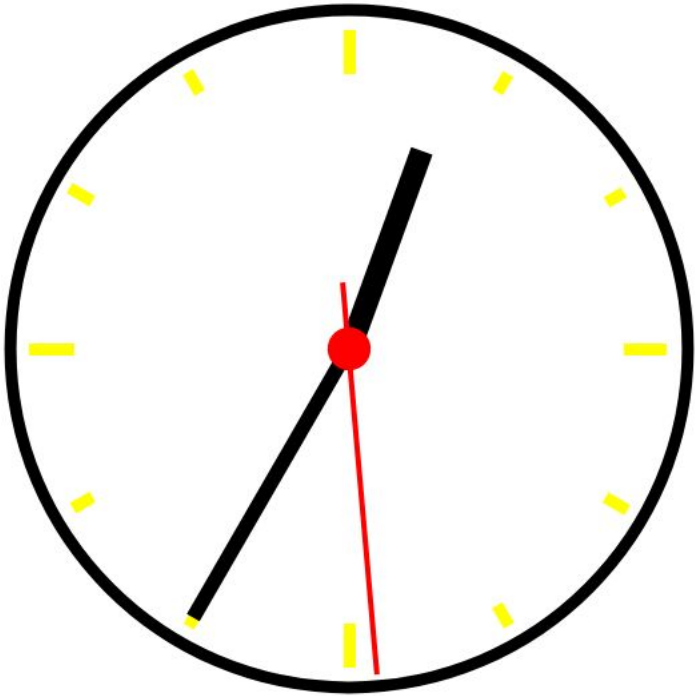
Style CSS Inline

Les style CSS inline remplace les style definie dans les feuille de style.

Dans la plupart des cas il vaut mieu définir des règles CSS et gérer les class avec javascripte

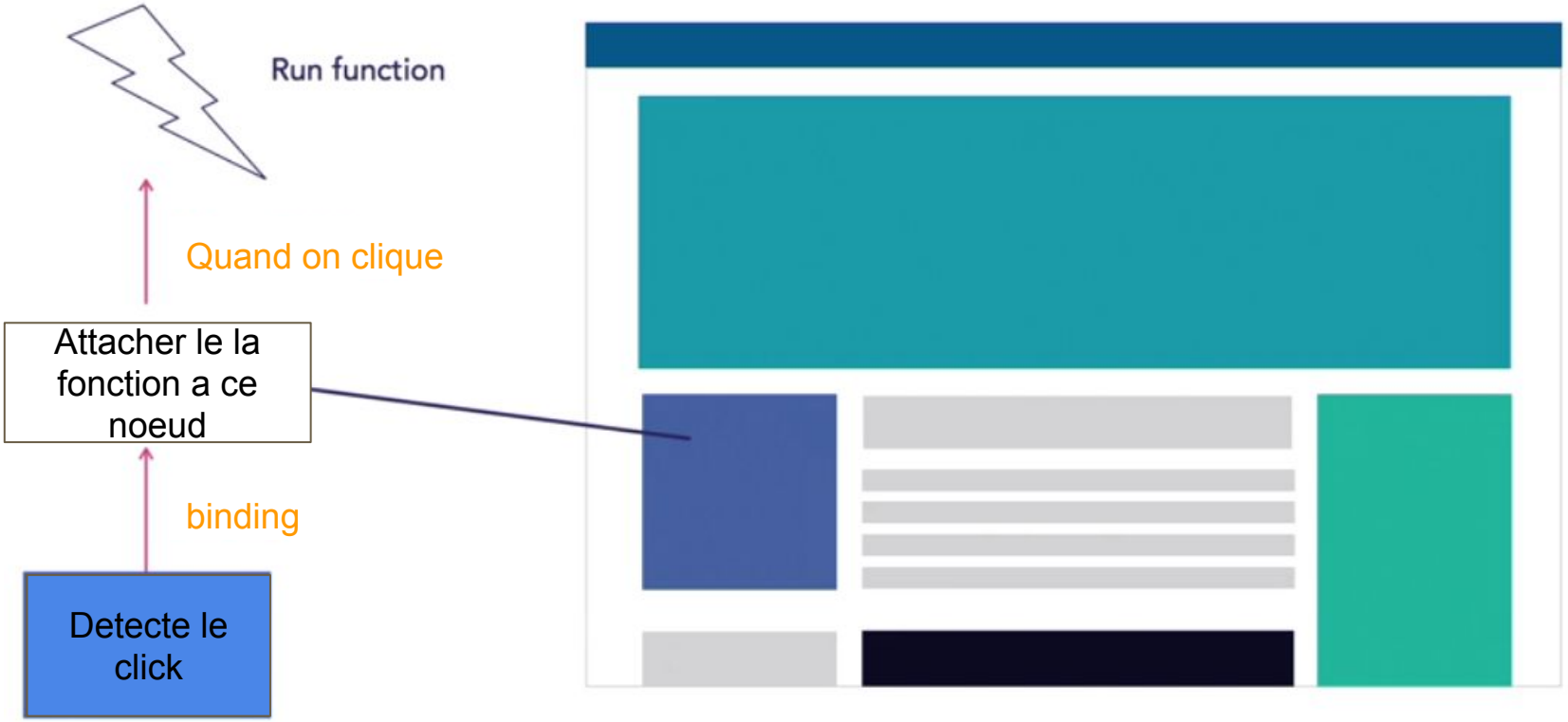
Exercice

1. Créer une horloge en utilisant une image SVG à l'aide d’un éditeur de graphiques vectoriel.
2. Créer un fichier css pour modifier son apparence.
3. Écrire un code javascript pour animer cette horloge.



Les événements

Les événements



Les événements

load
error
resize
online
Offline
Scrol
...

Click
Dblclick
Mouseover
Mouseout
Mousedown
Mouseup
Mousemove

Keydown
Keyup
Keypress
Focus
Blur
...

Change
Input
Select
Reset
Submet
...

https://developer.mozilla.org/en-US/docs/Web/Events

Les événements

```
1  function eventclbk(e){
2      e.preventDefault();// annule le comportement par defaut
3      // autres traitement si l evennement e surgit
4  }
5
6  //element.onevent= eventclbk;
7  element.onclick = eventclbk;
```

Les événements

```
1  const elmt=document.querySelector(".normal");
2  function eventclbk(e){
3      console.log("cliked");
4      elmt.classList.toggle("normal");
5  }
6  elmt.onclick = eventclbk;
```

event.js

event.html

```
7          <style type="text/css">
8              .normal{
9                  background-color: yellow;
10             }
11             div{
12                 background-color: blue;
13                 text-align: center;
14                 font-size: 20px;
15                 height: 40px;
16             }
17         </style>
18 </head>
19 <body>
20     <div class="normal">click me</div>
21 </body>
```

Les événements

```
1  const elmt=document.querySelector(".normal");
2  function eventclbk1(e){
3      elmt.classList.toggle("normal");
4  }
5  function eventclbk2(e){
6      console.log("Div clicked");
7  }
8  elmt.onclick = eventclbk1;
9  elmt.onclick = eventclbk2;
```

Problème: uniquement la fonction eventclbk2 sera exécutée

Les événements

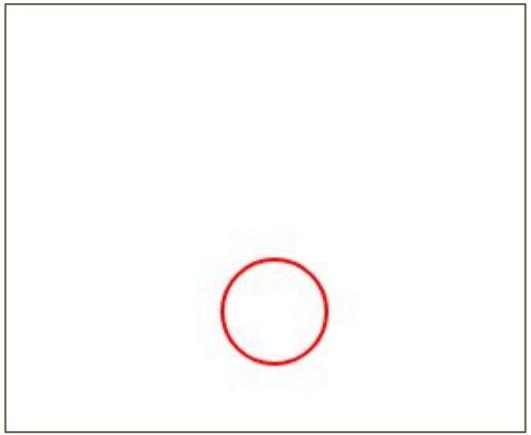
element.addEventListener("event", function_clbck, [*true*/false]);

```
1  const elmt=document.querySelector(".normal");
2  function eventclbk1(e){
3      elmt.classList.toggle("normal");
4  }
5  function eventclbk2(e){
6      console.log("Div clicked");
7  }
8  elmt.addEventListener("click",eventclbk1,false);
9  elmt.addEventListener("click",eventclbk2,false);
```

Les événements

Exercice:

1. Le cercle se déplace pour garder son centre en symétrie avec le curseur de la souris.
2. Le cercle change de couleur lorsqu'il touche curseur.
3. Utiliser ***.clientX*** et ***.clientY*** de l'objet ***event***



AJAX

Principe

Client

Javascript

CSS

HTML

Principe

Client

Serveur

Principe

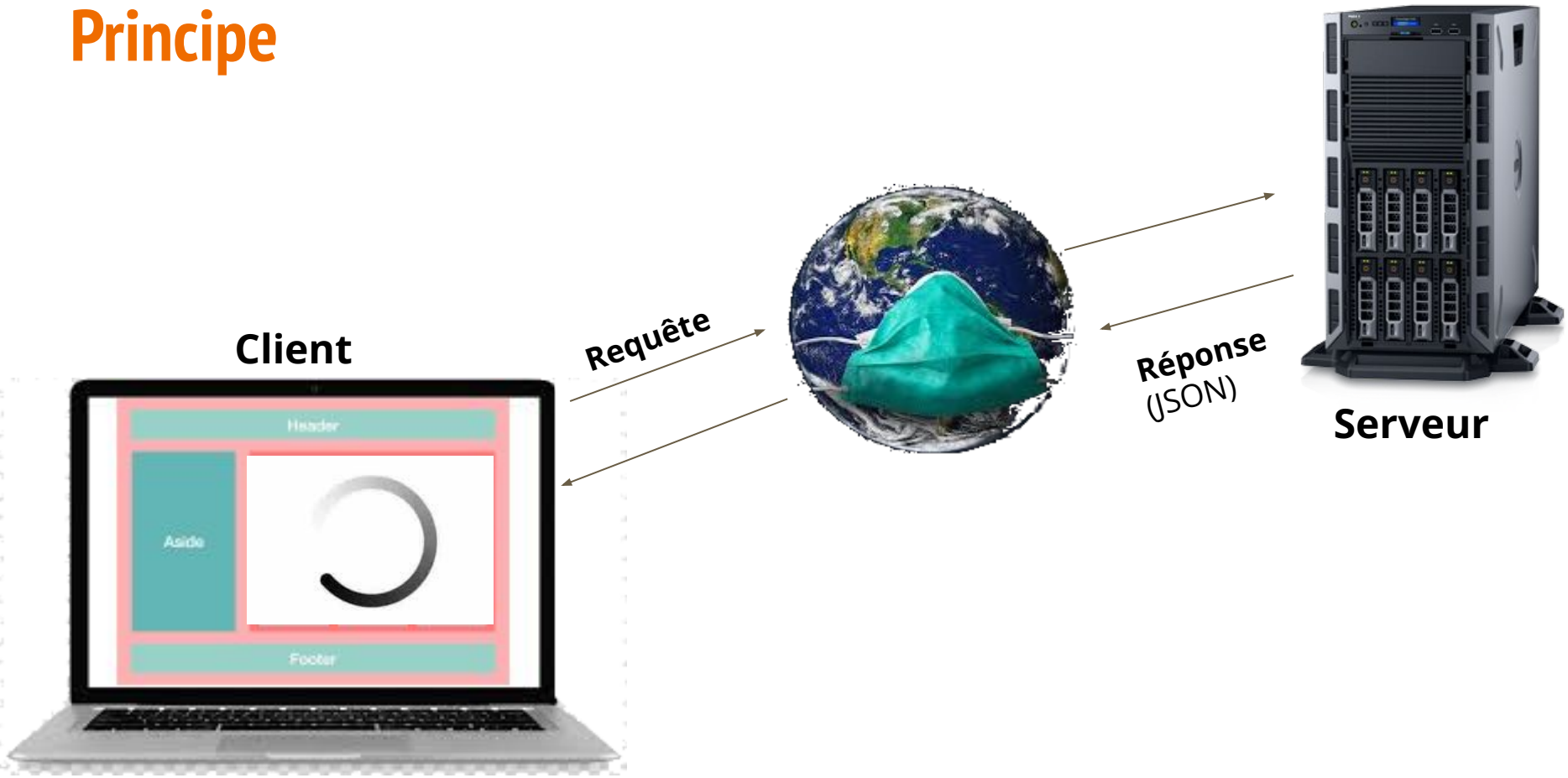
Client

Requête

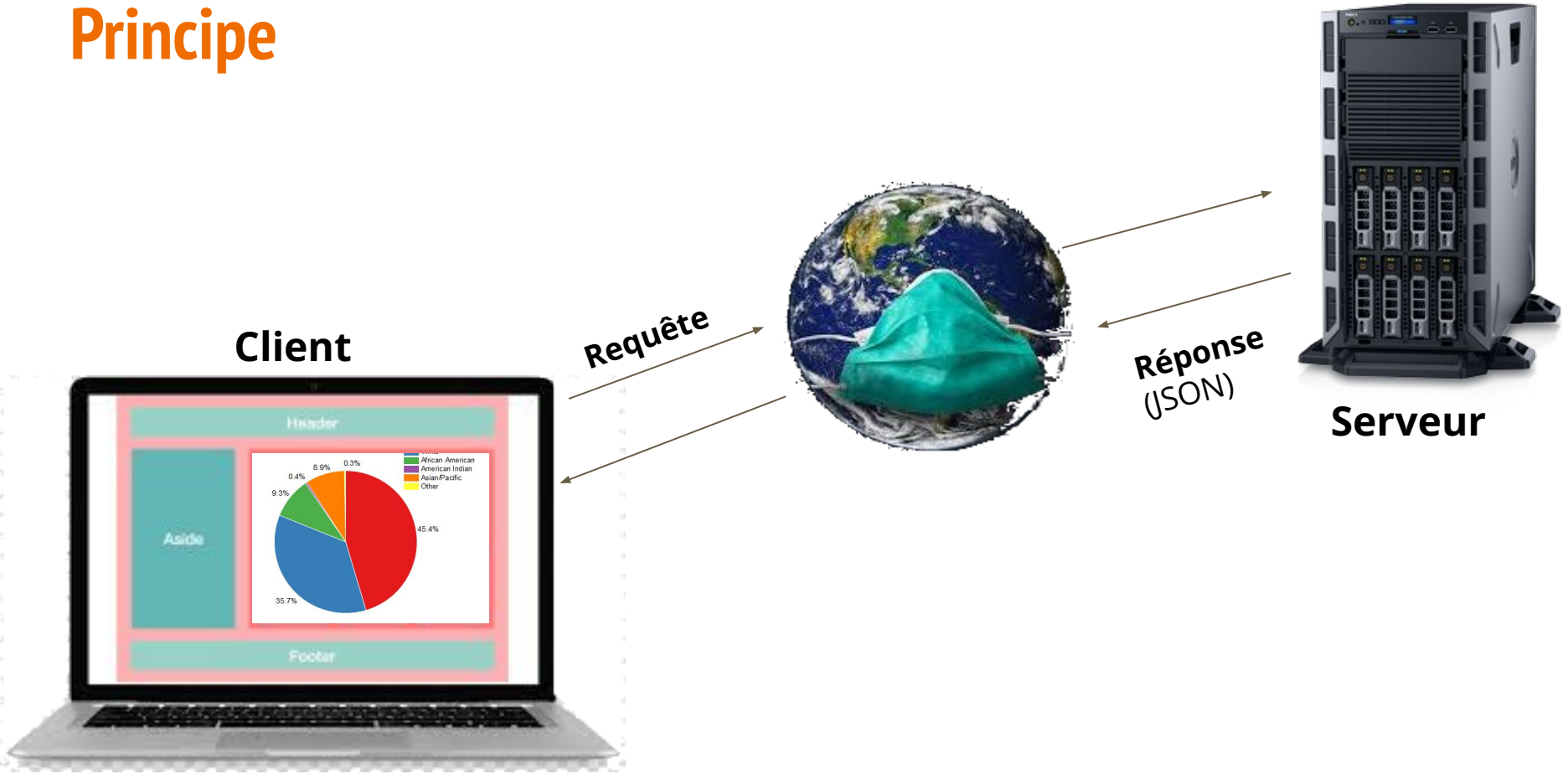
GET
POST
UPDATE
PATCH
DELETE
....

Serveur

Principe



Principe


Client
Requête
Réponse
(JSON)
Serveur
Header
Aside
Footer
African American
American Indian
Asian/Pacific
Other
0.3%
8.9%
0.4%
9.3%
45.4%
35.7%

XMLHttpRequest

Méthodes

- **new XMLHttpRequest()** : Créer un objet XMLHttpRequest
- **open(method,url,async,user,psw)** : Définir les paramètres de la requête
- **send()** : Envoyer la requete
- **abort()** : Annuler la requete

Propriétés

- **readyState :** Etat actuel de la requête (0, 1, 2, 3, 4)
- **Onreadystatechange :** La fonction à invoquer au changement de **readyState**
- **Status :** L'état du retour du serveur (200, 403, 404 ...)
- **Response:** Les données renvoyées par le serveur

XMLHttpRequest (GET)

```
let req=new XMLHttpRequest();
req.open("GET","https://api.covid19api.com/summary");
req.onreadystatechange=()=>{
    console.log(req.readyState);
    if(req.readyState==4 && req.status==200){
        let resp=JSON.parse(req.responseText)
    }
    console.log(resp);
}
req.send()
```

XMLHttpRequest (POST)

```
let httpReq = new XMLHttpRequest();
let url = "localhost/api/user";
let params = JSON.stringify({ name: "Gounane", age:21});
httpReq.open("POST", url, true);
httpReq.setRequestHeader("Content-type", "application/json; charset=utf-8");
httpReq.setRequestHeader("Content-length", params.length);
httpReq.setRequestHeader("Connection", "close");
httpReq.onreadystatechange = function() {
   if(http.readyState == 4 && httpReq.status == 200) {
       alert(httpReq.responseText);
   }
}
httpReq.send(params);
```

NPM, Webpack et Babel

Les Modules

index.html

```html
<script type="module" src="./script.js"></script>
```

index.js

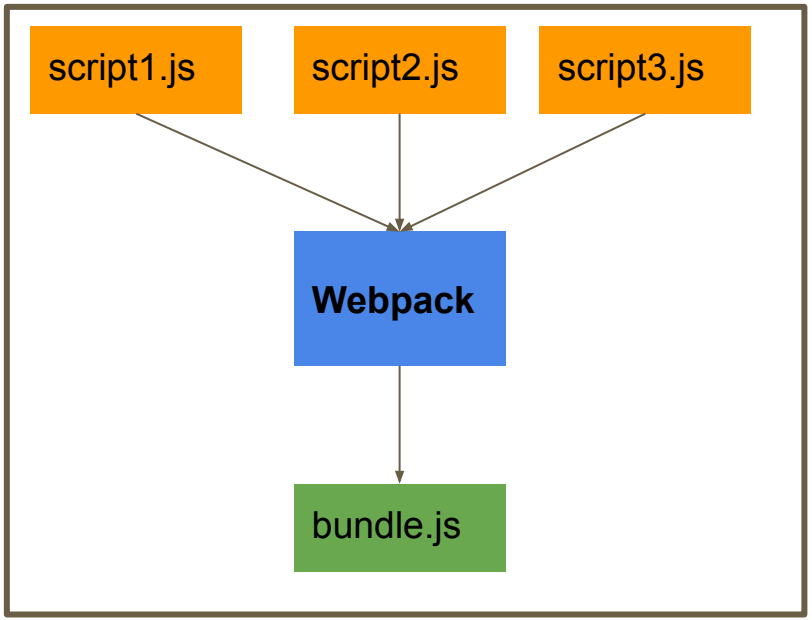
```
import "./shop.js"
console.log("From script");
```

shop.js

```
console.log("inside Module shop");
export const prix=10;
export const panier=[12,19,10]
```

Webpack

- Est un module bundler
- Création des fichiers statiques
- Automatisation des tâches



Installation

$npm init -y

$npm i -D webpack

$npm i -D *webpack-dev-server*

$ npm i -D babel-loader @babel/core @babel/preset-env

Configuration

```
// editor le fichier webpack.config.js
const path=require("path")
module.exports={
    entry:"./script.js",
    output:{
        path: path.resolve("./dist"),
         filename: "build.js"
    },
    watch: true,
    module: {
        rules: [
          {
            test: /\.js$/,
            exclude: /node_modules/,
            use: {
              loader: 'babel-loader',
              options: {
                presets: [
                  ['@babel/preset-env', { targets: "defaults" }]
                ]
              }
            }
          }
        ]
      }
}
```

Execution

- $***./node_modules/.bin/webpack***
- Ou bien, dans le fichier ***package.json*** modifier la ligne “start”:

```
"scripts": {
   "start": "webpack",
 },
```

Puis executer la commande

$ npm start