

X NATIS



Les bases en JavaScript

Michael

X NATIS



Compétence visée :
Créer des classes avec leurs
constructeurs et leurs
propriétés

X  NATIS



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

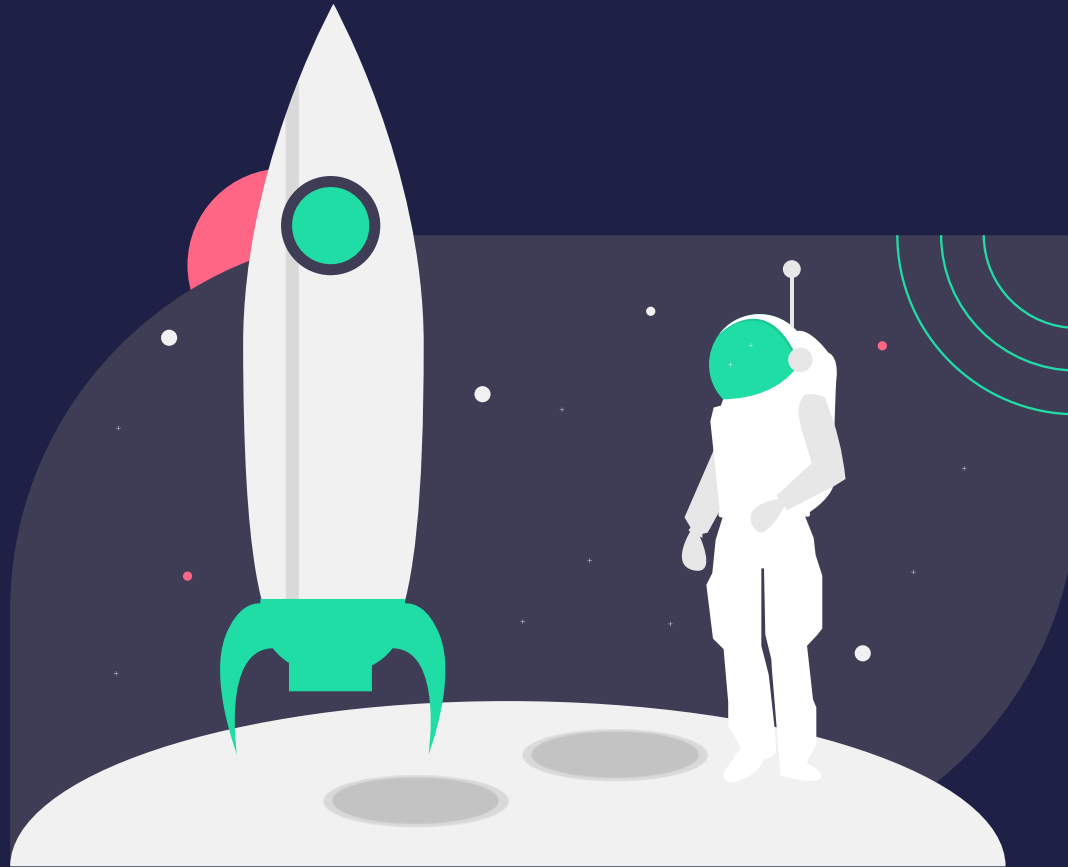
1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique (explorer this)

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique (créer des classes et afficher des objets)



Les bases en JS

1. Les types

2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

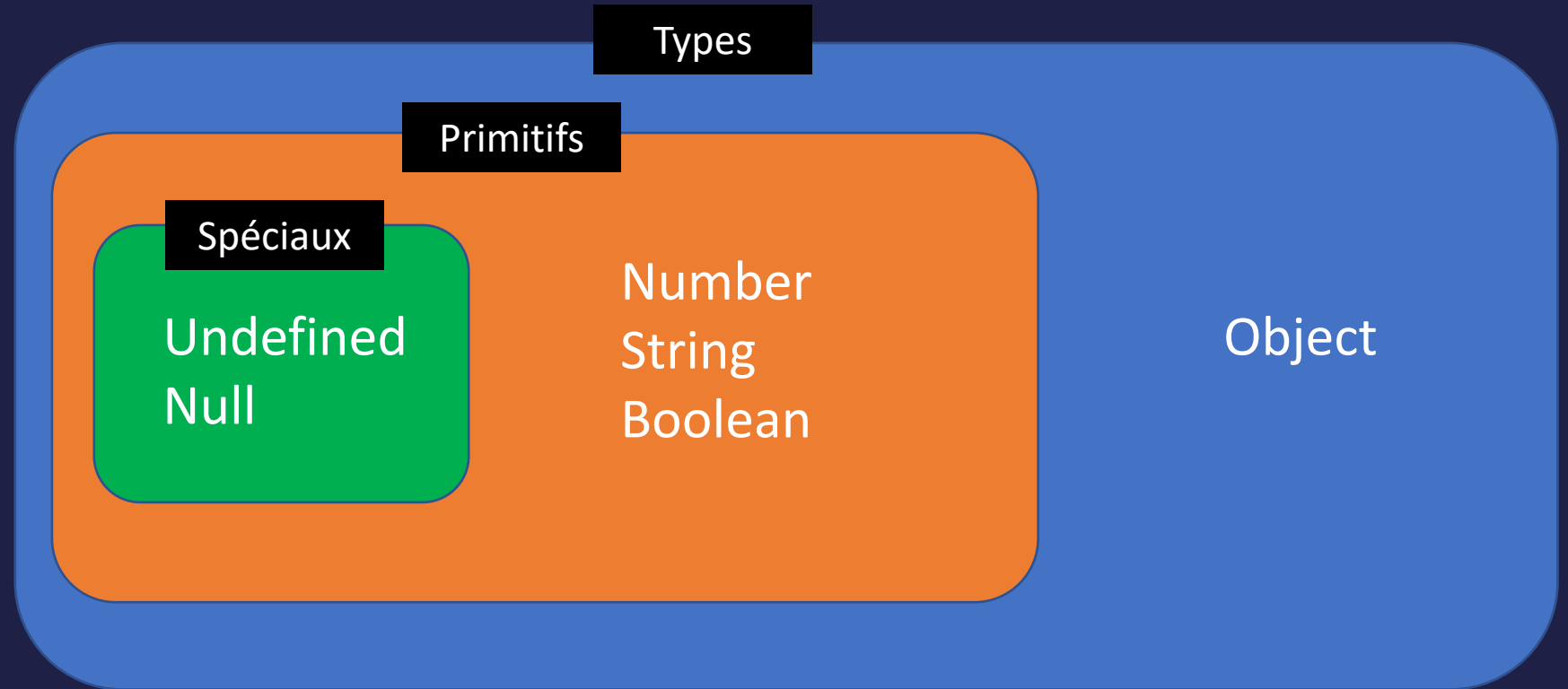
1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

L'opérateur `typeof` renvoie une chaîne qui indique le type de son opérande.

Les bases en JS

1. Les types
2. **La valeur ou la référence**
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Un objet est un pointeur (**référence**)
alors qu'un type primitif est une **valeur**

```
const a = 'hello';  
let b = a;  
b = 'ca va';  
console.log(a === b); // prints false
```



```
const obj = {  
  id: 4  
};  
let copie = obj;  
copie.id = 3;  
console.log(obj === copie); // prints true
```

Les bases en JS

1. Les types
2. La valeur ou la référence
3. **Scope (portée)**
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La **portée (scope)** d'une variable x est la région du programme dans laquelle les **utilisations de x font référence à sa déclaration.**

L'une des raisons fondamentales du scoping est de **garder les variables dans différentes parties** du programme distinctes les unes des autres.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. **Scope (portée)**
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La portée (ou scope) concerne la visibilité des variables. Elle détermine la portion de code à partir du moment où la variable naît jusqu'au moment où elle meurt.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. **Scope (portée)**
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Puisqu'il n'y a qu'un petit nombre de noms de variables courts et que les programmeurs partagent des habitudes concernant la dénomination des variables (par exemple, i pour un index de tableau), dans tout programme de taille modérée, **les conflits seront évités !**

Les bases en JS

1. Les types
2. La valeur ou la référence
3. **Scope (portée)**
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Avec une **portée dynamique (dynamic scoping)**, un identifiant global fait référence à l'identifiant associé à l'environnement le plus récent, et est rare dans les langues modernes.

En termes plus simples, dans la portée dynamique, l'interpréteur recherche d'abord **le bloc courant** puis successivement **toutes les fonctions appelantes**.

```
function hello() {  
    var x = 1;  
}
```

```
function cava() {  
    console.log(x);  
}
```

```
hello();  
cava();
```



```
function hello() {  
    var x = 1;  
}
```

```
function cava() {  
    console.log(x);  
}
```

```
hello();  
cava();
```

Les bases en JS

1. Les types
2. La valeur ou la référence
3. **Scope (portée)**
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La **portée statique** est également appelée **portée lexicale**. Dans cette portée, une variable fait toujours référence à son **environnement de niveau supérieur**.

Il s'agit d'une **propriété du texte du programme** et sans rapport avec la pile d'appels d'exécution. La portée statique facilite également la création d'un code modulaire, car un programmeur peut déterminer la portée simplement en regardant le code.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. **Scope (portée)**
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La portée dynamique oblige le programmeur à anticiper tous les contextes dynamiques possibles !

Les bases en JS

1. Les types
2. La valeur ou la référence
3. **Scope (portée)**
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Les variables ont 3 portées statiques :

- Global scope
- Function scope (ou aussi local scope)
- Block scope (pour let et const)

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. **IIFE**
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Nous pouvons ne pas travailler dans le contexte global en mettant notre code dans une fonction et l'exécuter directement : c'est une IIFE (**Immediately Invoked Function Expression**)

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. **Scope chain**
6. Les closures (fermetures)
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Pour retrouver une variable, l'interpréteur de JS va regarder dans la **portée statique courante**, et s'il ne la trouve pas, il regardera dans la **portée statique externe** ... puis ainsi de suite jusqu'au Global scope.

C'est ce qu'on appelle le **Scope chain** !

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. **Les closures (fermetures)**
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Une closure (fermeture) est la paire formée d'une fonction et des références à son état environnant (l'environnement lexical).

En d'autres termes, une closure donne accès à la portée d'une fonction externe à partir d'une fonction interne (on dit aussi que la fonction « capture son environnement »)

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. **Les closures (fermetures)**
7. Shadowing
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

En JavaScript, une closure (fermeture) est créée chaque fois qu'une fonction est créée.

```
function factory()  
{  
    let x = 'hello';  
    return function sayHello() {  
        console.log(x);  
    }  
}
```

```
let x = 'bye';  
const fun = new factory();  
fun();
```



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. **Shadowing**
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Lorsqu'une variable est déclarée dans une certaine portée **ayant le même nom défini sur sa portée externe** et lorsque nous appelons la variable depuis la portée interne, la valeur attribuée à la variable dans la portée interne est la valeur qui sera stockée dans la variable dans l'espace mémoire.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. **Shadowing**
8. Pratique (créer des objets)

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Ceci est connu sous le nom de **Shadowing** ou **Variable Shadowing**.

```
let number = 10;

{
  function double()
  {
    return number * 2;
  }
}

{
  function double()
  {
    return number * 3;
  }
}

console.log(double());
```



```
let number = 10;  
function double()  
{  
    let number = 20;  
    number = number * 2;  
}  
double();  
console.log(number);
```



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. **Pratique (créer des objets)**

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. **Pratique**

Construction d'un objet

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. **Classe**

Le DOM

1. Définition
2. Manipulation
3. **Pratique**

PRATIQUE



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La portée (ou scope) concerne la visibilité des variables **alors que le contexte** fait référence à **l'objet auquel** appartient une variable ou une fonction.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. **Contexte global**
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

GEC / Global Execution Context est également appelé exécution de base / par défaut.

Tout code JavaScript qui ne réside dans aucune fonction sera présent dans le contexte d'exécution global.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La raison derrière son nom « contexte d'exécution par défaut » où le code commence son exécution lorsque le fichier se charge pour **la première fois**.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Le GEC découle du flow d'exécution suivant :

1. l'interpréteur crée un objet global pour Node.js et un objet Window pour les navigateurs
2. Il référence l'objet ci-dessus au mot-clé 'this'
3. Il crée une memory heap afin de stocker des variables et des références de fonction

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La **moyen standard** d'obtenir l'objet global est **globalThis** !

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. **Création d'un nouveau contexte avec new**
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

On peut créer un nouveau contexte (et donc un objet) en utilisant **l'opérateur new**.

Cet objet **remplacera le contexte courant (this)**.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. **This : le contexte courant**
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Le contexte courant est en effet dénoté par le mot clé **'this'**.

Il référence majoritairement **l'objet courant** sur lequel les opérations s'appliquent.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. **Object.apply**, **Object.call**
5. Pratique

Construction d'un object

1. Construction littérale
2. `getOwnPropertyNames`
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La méthode `apply()` appelle une fonction en lui passant une valeur `this` et des arguments sous forme d'un tableau (ou d'un objet semblable à un tableau).

```
function Person(name) {  
    this.name = name;  
}  
  
const obj = {  
    age: 14  
};  
Person.apply(obj, ['Sophie']);  
console.log(obj);  
console.log(obj instanceof Person);
```



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. **Object.apply**, **Object.call**
5. Pratique

Construction d'un object

1. Construction littérale
2. `getOwnPropertyNames`
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La méthode `call()` réalise un appel à une fonction avec une valeur `this` donnée et des arguments fournis individuellement.

```
function Person(name) {  
  this.name = name;  
}  
  
const obj = {  
  age: 14  
};  
Person.call(obj, 'Sophie');  
console.log(obj);  
console.log(obj instanceof Person);
```



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique (explorer this)

Construction d'un objet

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

PRATIQUE



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Les objets sont constitués de **membres** (**propriétés** ou **méthodes**) qui permettent de les décrire.

Construire un objet, c'est le faire **physiquement exister en mémoire** pour contenir des données.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. **Construction littérale**
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Les valeurs des membres d'un objet peuvent être construites à partir de types de **données primitifs** ou à partir **d'autres objets**.

Comme des poupées russes ! 😊

```
const obj = {  
  id: 3,  
  firstname: 'Annie',  
  lastname: 'Versaire',  
  email: 'annie.versaire@gmail.com'  
};
```

```
const obj = {  
  id: 3,  
  firstname: 'Annie',  
  lastname: 'Versaire',  
  email: 'annie.versaire@gmail.com',  
  children: [  
    {  
      id: 4,  
      age: 12,  
      firstname: 'Jean',  
      lastname: 'Aimarre'  
    },  
    {  
      id: 6,  
      age: 24,  
      firstname: 'Guy',  
      lastname: 'Tarre'  
    },  
  ],  
};
```

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. **getOwnPropertyNames**
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Consultons les membres de nos objets ! 😊
Avec **Object.getOwnPropertyNames**

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. **Constructeur**
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Un constructeur (ou initialiseur) est une fonction qui vous permet de fournir toute initialisation personnalisée qui doit être effectuée avant qu'une autre méthode puisse être appelée sur un objet instancié.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. **Constructeur**
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Lorsqu'une fonction constructeur crée un objet, elle le « marque » :

- Par la propriété : constructor
- Par la propriété : __proto__

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. **Scope-safe**
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La plupart des constructeurs intégrés, tels que Object, Regex et Array, sont **scope-safe**.

Si new n'est pas utilisé, ils renvoient une instance appropriée de l'objet en appelant à nouveau le constructeur avec new.


```
function Voiture(price) {  
    if (!(this instanceof Voiture)) {  
        return new Voiture(price);  
    }  
    this.price = price;  
}  
  
const obj = Voiture(234);  
console.log(obj instanceof Voiture); // prints true
```

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. **Factory**
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

La fonction d'usine (**factory**) est similaire aux fonctions de constructeur, mais au lieu d'utiliser new pour créer un objet, les fonctions factory **créent simplement un objet et le renvoie.**

```
function factory()  
{  
  this.x = 'hello';  
  return function sayHello() {  
    console.log(this.x);  
  }  
}  
this.x = 'bye';  
const fun = new factory();  
fun();
```





Quel est le type de class ?



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

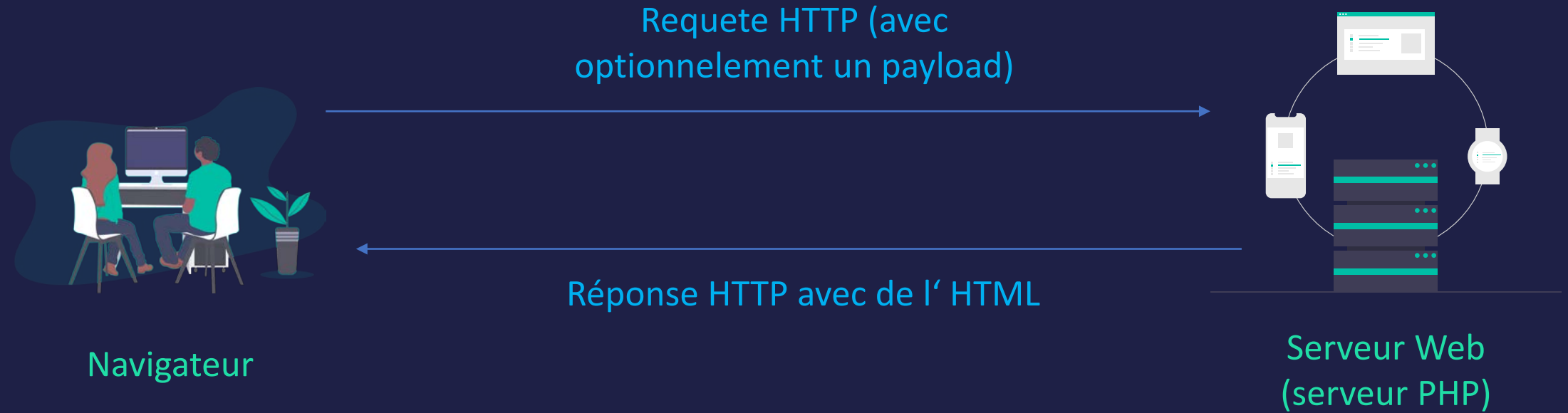
Les classes JavaScript ont été introduites avec ECMAScript 2015. Elles sont un « sucre syntaxique » par rapport à l'héritage prototypique.

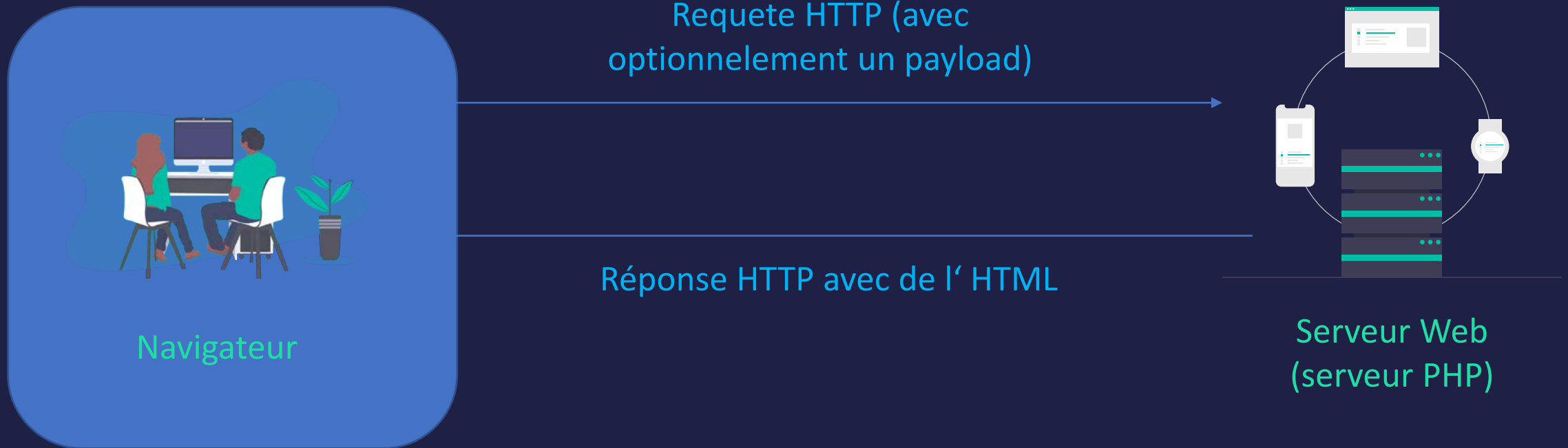
En effet, cette syntaxe n'introduit pas de nouveau modèle d'héritage en JavaScript !

CHIEN
+ age + groupe sanguin + taille + poids
+ mange() + dort() + aboie()

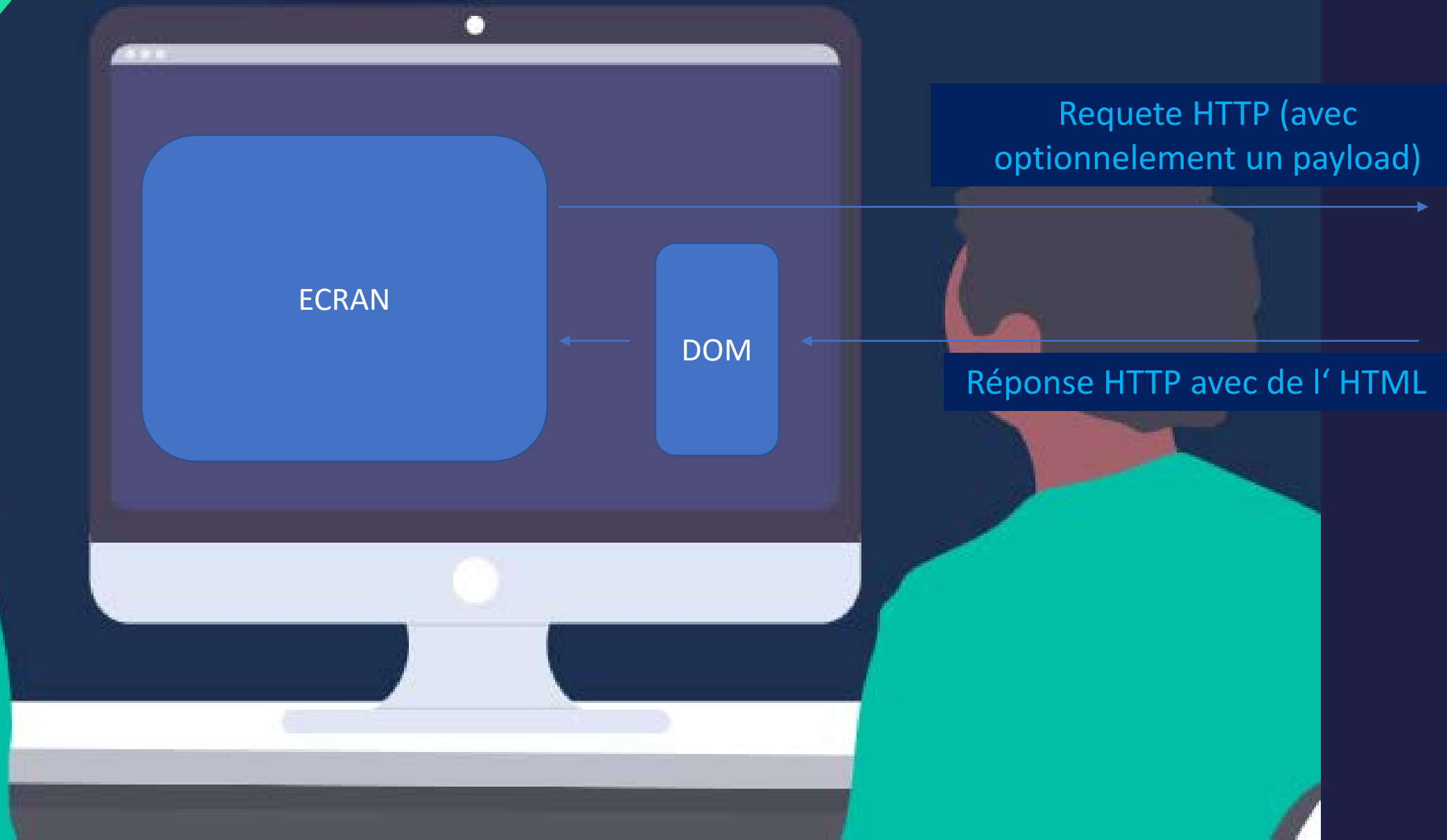
Qu'est-ce que le DOM ?







ZOOMONS DEDANS !



Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Le DOM est l'interprétation de l'HTML par le navigateur et c'est le DOM qui est affiché à l'écran.

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique

Cela peut être donc faux de dire qu'à l'écran, on voit le code HTML, car plus précisément, on voit le DOM sur l'écran (bien que le DOM vient de l'HTML)

Les bases en JS

1. Les types
2. La valeur ou la référence
3. Scope (portée)
4. IIFE
5. Scope chain
6. Les closures (fermetures)
7. Shadowing
8. Pratique

La notion de contexte

1. Contexte global
2. Création d'un nouveau contexte avec new
3. This : le contexte courant
4. Object.apply, Object.call
5. Pratique

Construction d'un object

1. Construction littérale
2. getOwnPropertyNames
3. Constructeur
4. Scope-safe
5. Factory
6. Classe

Le DOM

1. Définition
2. Manipulation
3. Pratique (créer des classes et afficher des objets)

PRATIQUE



X NATIS



Pour aller plus loin

<https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/Basics>