

JAVASCRIPT

Antonio Pisanello



Sommaire

- Histoire
- Environnement
- Programmer
- Structures de données
- Format de données



Présentation

Histoire | Qu'est ce que Javascript

Langage de programmation multiparadigme dynamiquement faiblement typé pour la programmation frontend et backend

Javascript accepte la programmation séquentielle/procédurale, orientée objet et programmation fonctionnelle

C'est le langage de programmation le plus mal compris au monde (et le plus populaire) !



Présentation

Histoire | Dynamiquement, statiquement, faiblement vs fortement typé

Dynamiquement Typé : Pas besoin de déclaré de type pour les variable (Int, Float, etc.), le type est connu au runtime

Statiquement typé : Obligation de donner le type de variable lors de la déclaration, sinon on reçoit une erreur de compilation

Fortement typé : Une fois que la variable est déclarée avec son type, elle restera toujours de ce type. Cependant on peut la “caster” dans d'autres types

Faiblement typé : Les variables n'ont pas de types prédéfinis, on peut choisir de stocker un autre type plus tard dans la même variable



Présentation

Histoire

- 1995 Mosaic/Netscape
- 1997 ECMA (European Computer Manufacturers Association)
 - ECMA-262
- 1998 ActionScript - Flash, Adobe
- 1999 JScript - Microsoft (ECMA-262 3)
- 2011 ECMA-262 5.1
- 2015 ECMA-262 6



Java is to JavaScript as ham is to hamster. Jeremy Keith



Présentation

Histoire | Pourquoi utiliser Javascript



- Language moderne
- Language up-to-date, qui suit les paradigme actuels
- La meilleure manière de rendre une page dynamique
 - Dès que vous voulez ajouter des interaction, actions, événements on va avoir besoin de JS (sans compter les animation CSS ou SVG)



Présentation

Histoire | Pourquoi utiliser Javascript

Apr 2023	Apr 2022	Change	Programming Language		Ratings	Change
1	1			Python	14.51%	+0.59%
2	2			C	14.41%	+1.71%
3	3			Java	13.23%	+2.41%
4	4			C++	12.96%	+4.68%
5	5			C#	8.21%	+1.39%
6	6			Visual Basic	4.40%	-1.00%
7	7			JavaScript	2.10%	-0.31%
8	9	^		SQL	1.68%	-0.61%
9	10	^		PHP	1.36%	-0.28%
10	13	^		Go	1.28%	+0.20%
11	12	^		Delphi/Object Pascal	1.23%	+0.05%
12	8	v		Assembly language	1.03%	-1.31%



Présentation

Histoire | Javascript Meme

⋮

Console


What's New


⊘

top

▼

Filter

> 2 + 2	
< 4	
> "2" + "2"	
< "22"	
> 2 + 2 - 2	
< 2	
> "2" + "2" - "2"	
< 20	

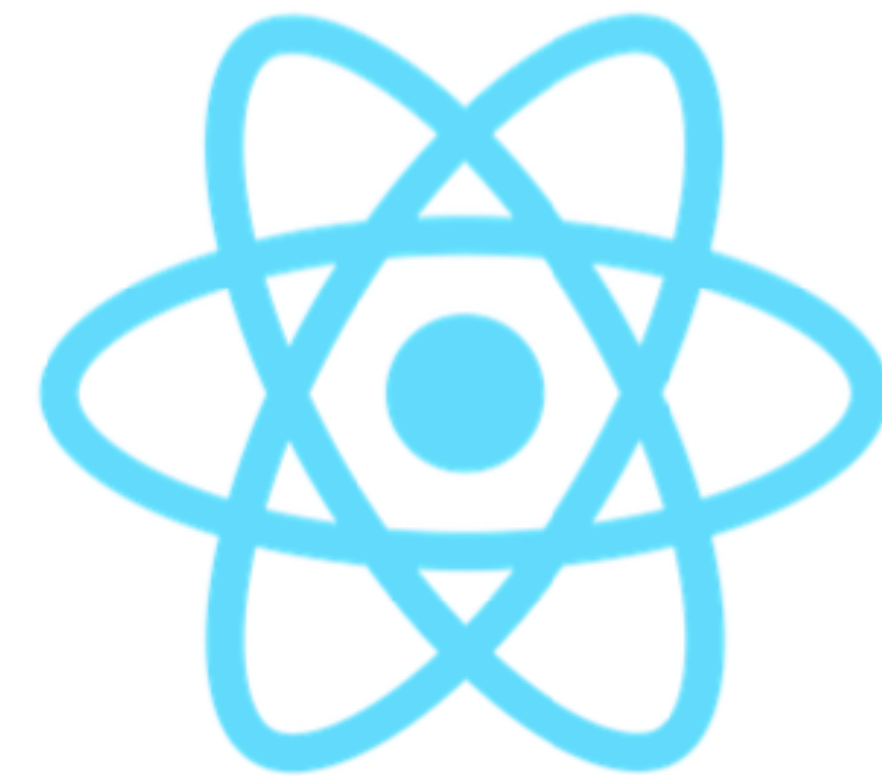
> typeof NaN	> true==1
< "number"	< true
> 9999999999999999	> true===1
< 10000000000000000	< false
> 0.5+0.1==0.6	> (!+[[]+[]+![])).length
< true	< 9
> 0.1+0.2==0.3	> 9+"1"
< false	< "91"
> Math.max()	> 91-"1"
< -Infinity	< 90
> Math.min()	> []==0
< Infinity	< true
> []+[]	
< ""	
> []+{}	
< "[object Object]"	
> {}+[]	
< 0	
> true+true+true===3	
< true	
> true-true	
< 0	



Planning

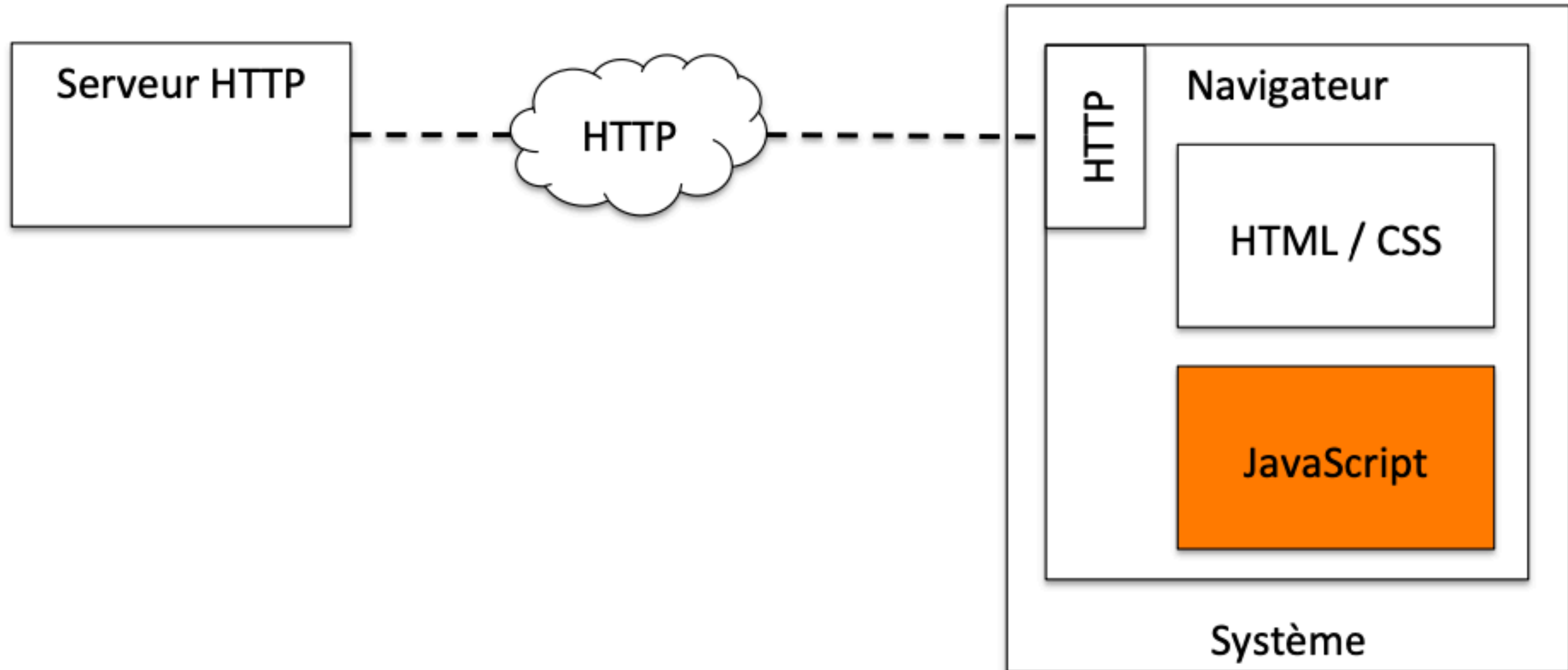
Histoire | JS renaissance

- Google et AJAX (Google maps / hearth)
- NodeJS : création en 2009, open-source, cross-platform, environnement backend Javascript
- ECMAScript 6 ou Javascript 2015: plus propre, plus sain et simplification du langage
- Framework JS
- Utilisé par des compagnies leader tels que : Google, Meta, LinkedIn, Netflix, Mozilla, etc.



Objectifs

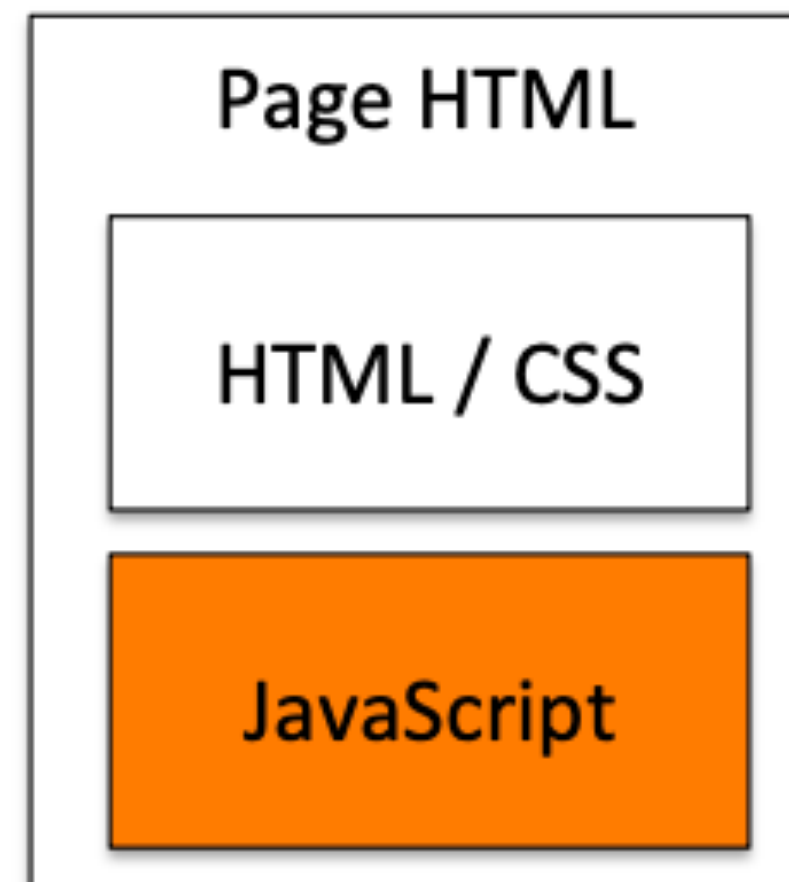
Environnement | Vue d'ensemble



Planning

Environment | Vue d'ensemble

- Le JavaScript est un langage interprété
- On l'intègre (généralement) dans une page web à l'aide de la balise `<script>...</script>`



Planning

Environment | Intégration du code

- Balise `<script>...</script>`
- HTML 1.0 :
 - `<script language="javascript">...</script>`
- HTML 4:
 - `<script type="text/javascript">...</script>`
- HTML 5:
 - `<script>...</script>`



Planning

Environment | Intégration du code

- Le contenu de la Balise ``<script>...</script>`` est interprété comme du JavaScript

```
<script language="javascript">
```

```
// partie interprétée comme du code
```

```
</script>
```



Planning

Programmer | Sommaire

- Syntaxe
 - Commentaires
 - Variables, constantes
 - Opérateurs
 - Fonctions
 - Traitement - Structures de contrôle
 - Structure de données
 - Objets
- Classes fournies



Planning

Programmer | Les références

- **Références officielles**

- <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- <http://www.w3.org/DOM/>

- **Fournisseurs**

- <https://developer.mozilla.org/fr/docs/JavaScript>
- [http://msdn.microsoft.com/fr-fr/library/ie/yek4tbz0\(v=vs.94\).aspx](http://msdn.microsoft.com/fr-fr/library/ie/yek4tbz0(v=vs.94).aspx)
- <https://developers.google.com/v8/>

- **Tiers**

- <http://www.w3schools.com>
- <http://www.javascriptkit.com/jsref/>
- <http://www.xul.fr/ecmascript/>



Planning

Programmer | Commentaires

- **Commentaires de bloc**

`/* Ceci est un commentaire`

`Sur plusieurs lignes`

`*/`

- **Commentaire Inline**

`// Ceci est un commentaire sur une ligne`

- **Error**

`/* Par contre il n'est /* pas possible d'avoir des commentaires imbriqués */ SyntaxError */`



Planning

Programmer | Variables

var x Déclare une variable, assignation de valeur optionnelle (Vielle façon de faire, obsolète)

let x Declare une variable de block, variable locale, assignation de valeur optionnelle

const x = 1 Declare une constante, une constante est locale au block comme **let**

N'utilisez jamais de **var** commencez toujours avec **const** et si vous avez réellement besoin utilisé **let**

Le nom d'une variable en javascript doit commencer par une lettre un underscore (_) ou un signe dollar (\$)



Planning

Programmer | Variables

Javascript avant ES6 n'avait pas de variables de block. Une variable de block est locale au bloc dans lequel elle a été déclarée

Variable de block **let** ou **const**
Fonctionnement comme en C

```
if (true) {  
  let y = 5  
}  
  
console.log(y) // ReferenceError: y is not defined
```

```
if (true) {  
  var x = 5  
}  
  
console.log(x) // x = 5
```



Planning

Programmer | Opérateurs

- Affectation
- Concaténation
- Arithmétique
- Logique
- Comparaison
- Spéciaux



Planning

Programmer | Opérateurs

- Affectation `[=, +=, -=, *=, /=, %/]`
- Concaténation `+`
- Arithmétique `[+, -, *, /, ++, --, %]`
- Logique `[&&, ||, !]`
- Comparaison `[==, !=, <, >, <=, >=, ===, !==]`
- Spéciaux
 - `this, new, instanceof, typeof, in, ? :, ...`



Planning

Programmer | Conditions I



```
// instructions  
if (test) {  
    // instructions si test est vrai  
}  
// instructions
```



Planning

Programmer | Conditions II



```
// instructions
if (test) {
    // instructions si test est vrai
} else {
    // instruction si test est faux
}
// instructions
```



Planning

Programmer | Conditions III



```
// instructions
if (test1) {
    // instructions si test est vrai
} else if (test2) {
    // instruction si test1 est faux mais que tes2 est vrai
} else {
    // instruction si les deux tests sont faux
}
// instructions
```



Planning

Programmer | Conditions IV

```
//instructions
switch(boisson) {
  case "cafe":
    // instructions
    break

  case "the":
    //instructions
    break

  default:
    // instruction
}
```



Planning

Programmer | Conditions V

Opérateur Ternaire; if-then-else



```
const majorite = age >= 18 ? "Utilisateur est majeur" : "utilisateur est mineur"
```



Planning

Programmer | Boucles I

Tant que le test est vrai on boucle



```
//instructions  
while(test){  
    // instructions  
}  
// instructions
```



Planning

Programmer | Boucles II

Utilisé quand on sait à l'avance combien de fois on va boucler



```
//instructions  
for(let i=0; i<10; i++){  
    // instructions  
}  
// instructions
```



Planning

Programmer | Boucles III

Boucle a travers les clés (objet) et les index (array)



```
//instructions  
for(let val in structure){  
    // instructions  
}  
// instructions
```



Planning

Programmer | Boucles IV

Boucle a travers les valeurs (array)



```
//instructions  
for(let val of structure){  
    // instructions  
}  
// instructions
```



Planning

Programmer | Fonctions

Si une fonction est appelée avec plus de paramètres qu'attendu JS va simplement ignorer les extra paramètres

Si une fonction est appelée avec avec moins de paramètres que prévu JS vas donner la valeur de `undefined` aux paramètres manquant



```
function foo(a, b) {  
  return a+b  
}
```



```
const result = foo(1,2,3)  
console.log(result) // 3
```



Planning

Programmer | Fonctions anonymes



```
const y = function () {  
  // instructions  
}  
y.name // "y"  
y() // execute la fonction y
```



```
function foo(f) {  
  f() // execute la fonction passée en paramètre  
}  
  
foo(function () {  
  // instructions  
})
```



Planning

Programmer | Fonctions fléchée



```
([parametre]) => {  
  // instruction  
}
```



```
function foo(f) {  
  f() // execute la fonction passée en paramètres  
}  
  
foo(() => {  
  // instructions  
})
```



Planning

Programmer | Fonctions fléchée avec paramètres



```
function foo(f) {  
  f() // execute la fonction passée en paramètres  
}  
  
foo((param1, param2) => {  
  // instructions  
})
```



Planning

Programmer | Fonctions fléchée avec UN paramètre



```
function foo(f) {  
  f() // execute la fonction passée en paramètres  
}  
  
foo((param) => {  
  // instructions  
})
```



```
function foo(f) {  
  f() // execute la fonction passée en paramètres  
}  
  
foo(param => {  
  // instructions  
})
```



Planning

Programmer | Fonctions fléchée sans paramètres



```
function foo(f) {  
  f() // execute la fonction passée en paramètres  
}  
  
foo(() => {  
  // instructions  
})
```

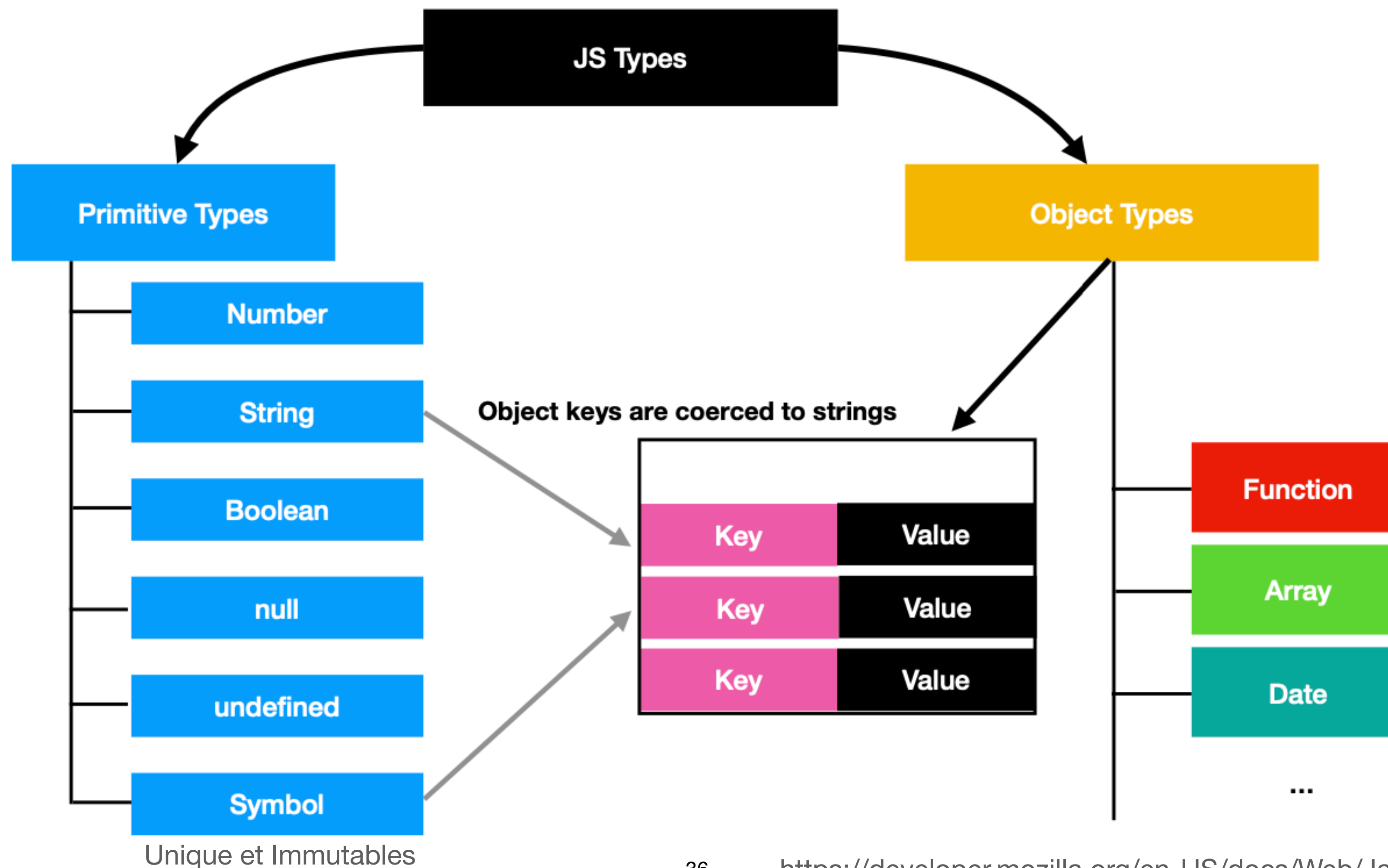


```
function foo(f) {  
  f() // execute la fonction passée en paramètres  
}  
  
foo(_ => {  
  // instructions  
})
```



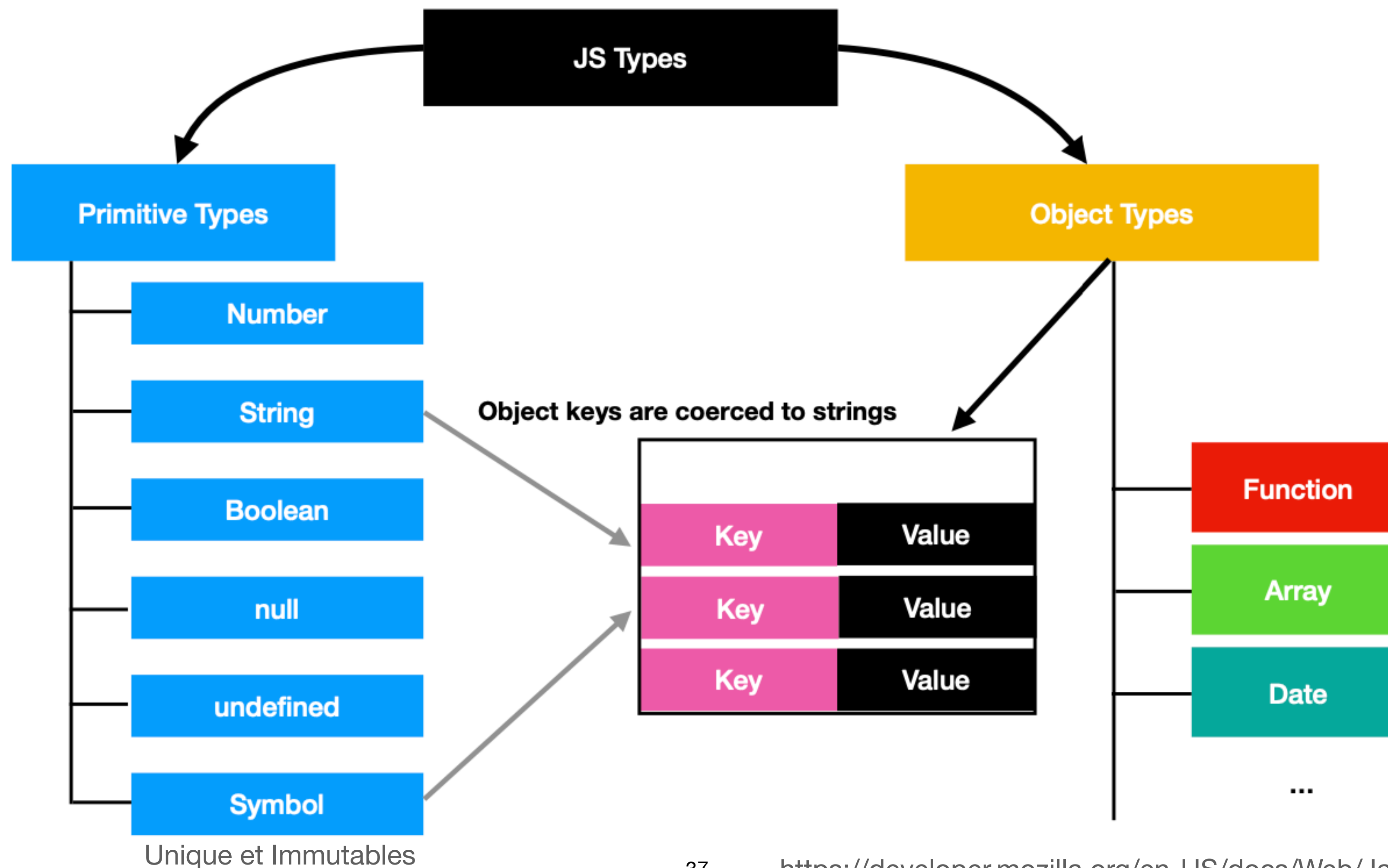
Planning

Programmer | Structure de données



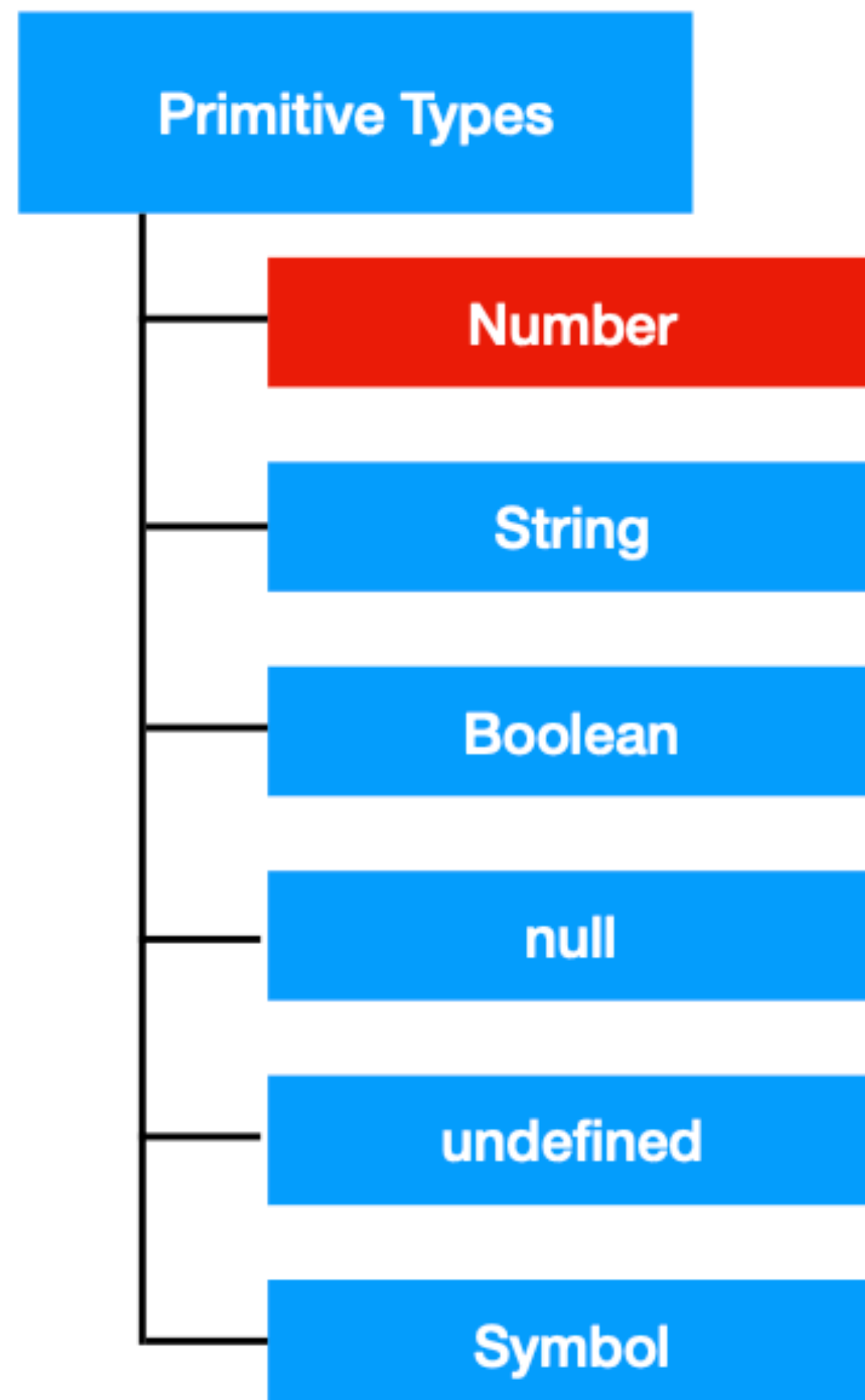
Planning

Programmer | Structure de données



Planning

Programmer | Structure de données Number



Double précision 64bit float (Pas d'Integer)

Fonction arithmétique de base (+, -, *, /, %)

Math.round(), math.floor(), Math.cos(), etc.

parseInt(), parseFloat() to parse strings

Nombres Spéciaux : **Nan, Infinity**



Planning

Programmer | Structure de données



null Signifie qu'il n'y a pas de valeur

undefined Signifie qu'il n'y a pas de variables



Planning

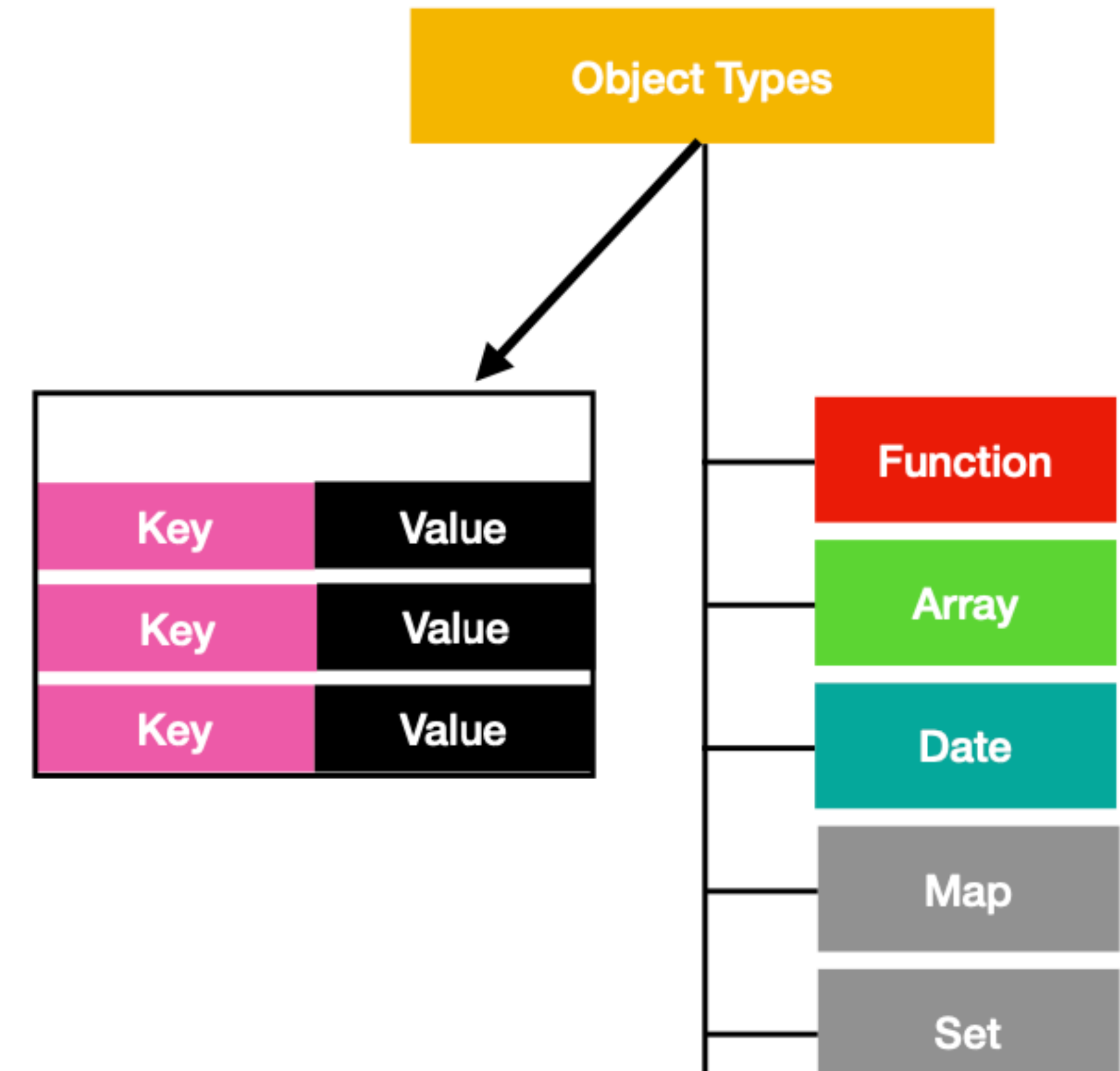
Programmer | Structure de données Objet

Le type le plus important en JS.

Tout est objet à part les types primitifs

Les objets sont des collections de propriétés
dictionnaires / hash map / tableaux associatifs

Les fonctions sont des objets avec la particularité
de pouvoir être appelée



Planning

Programmer | Structure de données Objet

```
const obj = {  
  "a": 12,  
  "b": "Salut"  
  c: _ => {console.log("Hello")}  
}  
  
console.log(obj["a"]) // affiche 12  
console.log(obj.a) // affiche 12  
obj["a"] = 21 // re-assignement  
obj.d = 3.14 // ajout d'une valeur dynamiquement  
obj.c() // affiche "Hello"
```



Planning

Programmer | Structure de données Array

```
const arr = [1,2,3]
console.log(arr[0]) // 1
let emptyArray = []
emptyArray.push(2)
console.log(emptyArray) // [2]
console.log(arr.length) // 3
```

```
let liste = ['Pomme' , 'Banane']
let liste2 = new Array('Pomme', 'Banane')

liste[1] = 'citrons'
```



Planning

Programmer | Structure de données Array exemple de méthodes



```
let liste = ['Pomme' , 'Banane']
```

```
liste.push('Melon') // ajoute
```

```
liste.length // Nb d'éléments
```

```
liste.sort() // trie
```

```
liste.reverse() // L'autre sense
```

```
liste.pop() // retir et retourne le dernier élément
```

```
...
```



Questions ?

