Cours JavaScript

Part 1

PLAN

JS

Présentations

Les Variables

Les Structure de contrôles

Les Boucles

Les Fonctions

TP

Conclusion

Jérémy Young

JS

Diplôme d'ingénieur SIGL (Ingésup Aix)

Divers projets perso en Node.js

Labo en Node.js

Mémoire basé sur Node.js

Développeur FullStack Web (spécialisé Node.JS + MongoDB)

A votre tour

Installation de votre Env de Dev

JS

Télécharger et installer Visual Code Studio

Télécharger et installer Node.js

https://danielarancibia.wordpress.com/2017/03/28/install-or-upgrade-nodejs-with-nvm-for-windows/

Créer un compte GitHub

https://github.com/

Créer un compte Repl.it

https://repl.it/

Les Variables

```
1 let maVariable = 15;
```

camelCase / lowerCamelCase

(Utilisé pour les Variables & Paramètres)

```
1 let ma_variable = 'Hello World';
```

snake_case / underscore_case

(Utilisé pour les Variables en Python, PHP, Ruby)

```
1 const MA_CONSTANTE_PI = 3.14159265359;
```

SCREAMING_SNAKE_CASE

(Utilisé pour les constantes)

```
1 var maOldVariable = "Ancien moyen de déclarer une variable";
```

Déclarer une Variable

```
1 let maVariableDeclare;
```

Permet de réserver une adresse en mémoire pour stocker plus tard une valeur

Initialiser une Variable

```
1 let maVariableDeclare;
2 maVariableDeclare = 'Hey';
```

Permet d’initialiser (d'affecter) une valeur dans l'adresse mémoire de "maVariableDeclare"

Déclarer et Initialiser une Variable

```
1 let maVariableDeclareEtInitialise = 'Hey';
```

Permet de déclarer et d'affecter une valeur dans l'adresse mémoire en une seule instruction

Il est aussi possible de changer de type de donné dynamiquement

```
1 let maVariableDeclareEtInitialise = 'Hey';
2 console.log(typeof maVariableDeclareEtInitialise); // => string
3
4 maVariableDeclareEtInitialise = 521.2;
5 console.log(typeof maVariableDeclareEtInitialise); // => number
```

Typage en JS

```
1 typeof "Best cours"; // => string
```

```
1 typeof 42; // => number
```

```
1 typeof Symbol('foo'); // => symbol
```

```
1 typeof 42n; // => bigint
```

```
1 typeof true; // => boolean
```

```
1 typeof -42.58; // => number
```

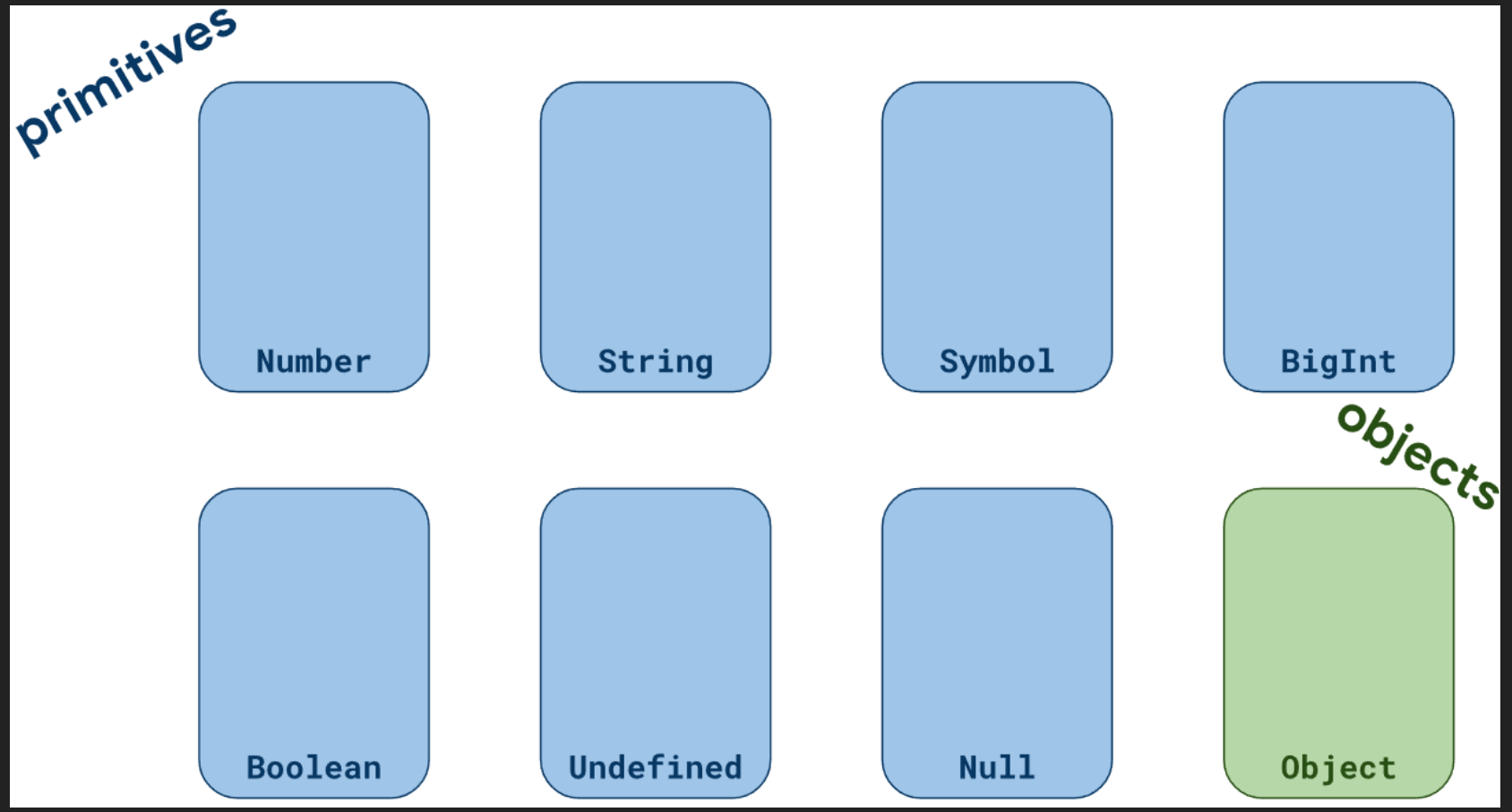
```
1 typeof undefined; // => undefined
```

```
1 typeof null; // => object
```

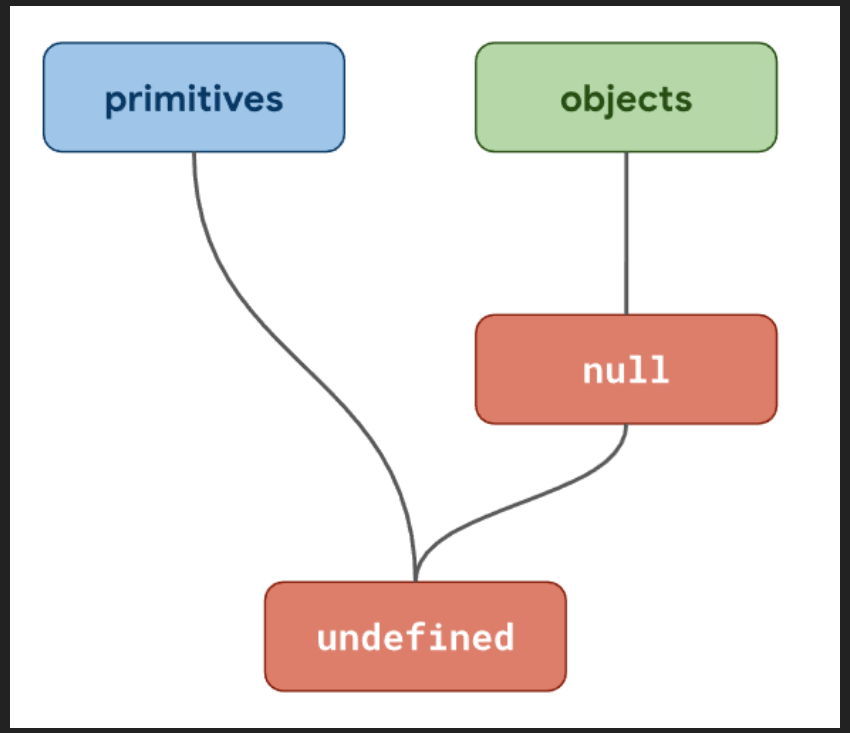
```
1 typeof { x: 42 }; // => object
```

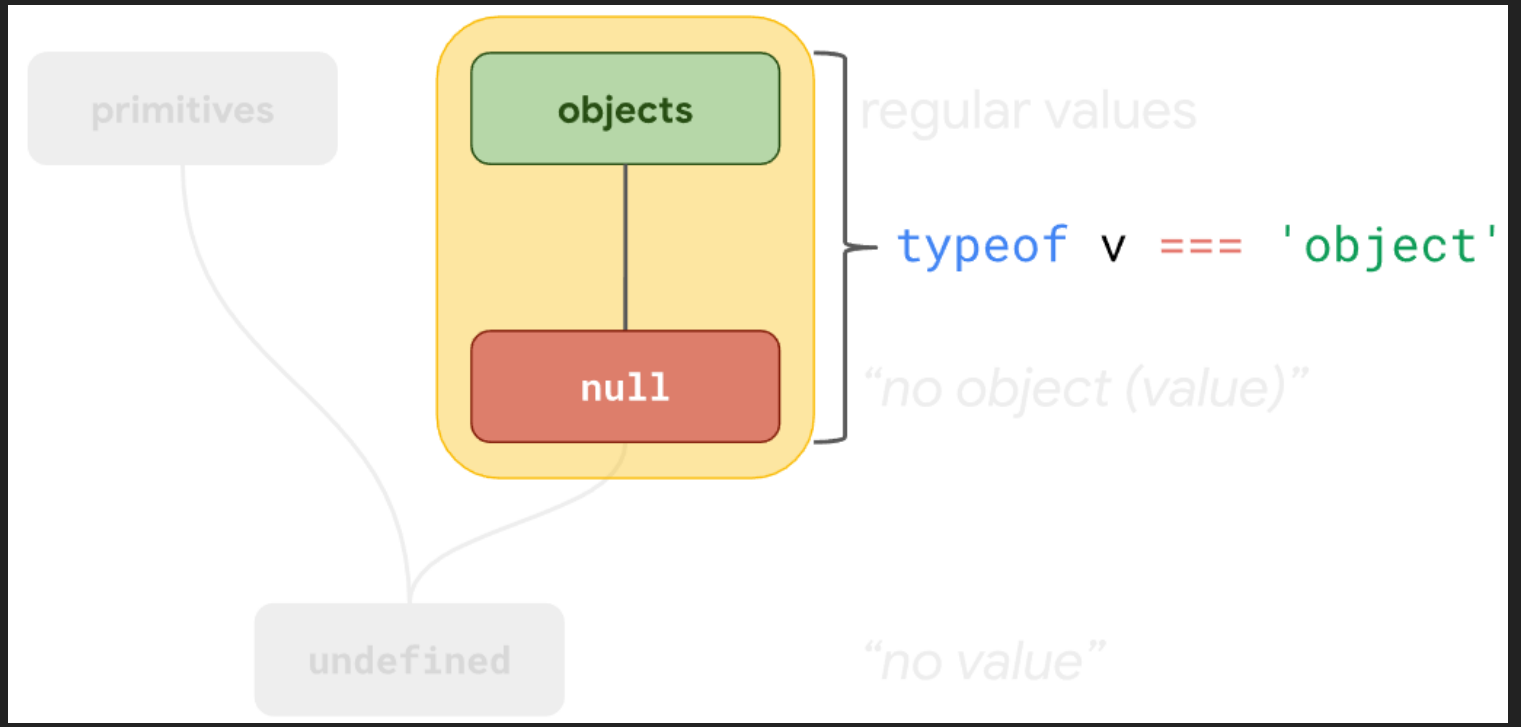
```
1 typeof [42]; // => object
```

Typage en JS


primitives
Number
String
Symbol
BigInt
objects
Boolean
Undefined
Null
Object

Typage en JS


primitives
objects
null
undefined


primitives
objects
null
undefined
regular values
typeof v === 'object'
“no object (value)”
“no value”

Les références

Primitives :

```
let greeting = "Hello World";

let greeting2 = greeting;

greeting2 = "Hello Everyone";

console.log(greeting); // => Hello World
console.log(greeting2); // => Hello Everyone
```

```
let a = 10;
let b = 40;

let result = a + b;
let alteredResult = result;

alteredResult = 999;

console.log(a); // => 10
console.log(b); // => 20
console.log(result); // => 50
console.log(alteredResult); // => 999
```

Objects :

```
let greeting = [];

greeting.push("Hello", "World");

let greeting2 = greeting;

greeting2.push("Everyone");

console.log(greeting[0]); // Hello
console.log(greeting[12]); // undefined
console.log(greeting); // => Hello World Everyone
console.log(greeting2); // => Hello World Everyone
```

```
let originalValues = { a: 10, b: 40 };

let data = { result: originalValues.a + originalValues.b }
let alteredData = data;

alteredData.result = 999;

console.log(originalValues); // => { a: 10, b: 40 }
console.log(data); // => { result: 999 }
console.log(alteredData); // => { result: 999 }
```

Le scope d'une variable

En JS, il y a deux type de scope :

- Global scope
- Local scope

Chaque fonction créer son propre scope

```
1 function sayHello (pseudo) {
2   console.log('Hello ' + pseudo);
3 }
```

Le scope détermine l'accéssibilité de la variable

```
1 if (true) {
2   let a = 150;
3   console.log(a); // => 150
4 }
5
6 console.log(a) // => ReferenceError:
7 // a is not defined
```

Toute variable contenant une valeur de type primitive est supprimé par le Garbage collector dès que le scope ne contient plus d'instruction à exectuer

Le scope d'une variable

Differences entre le mot réservé *var* et [*let* / *const*] :

var (function scoped) :

```
function sayHello (pseudo) {
  var b = 42;

  console.log('Hello ' + pseudo);
  console.log(b); // => 42
}

sayHello('John'); // => Hello John

console.log(b); // => ReferenceError: a is not defined
```

il est possible de re-déclarer une variable avec le même nom :

```
var a = 10;
console.log(a); // => 10
var a = 20;
console.log(a); // => 20
```

```
if (true) {
  var a = 150;
  console.log(a); // => 150
}

console.log(a) // => 150
```

Si une variable déclaré avec *var* n'est pas déclaré dans une fonction, elle sera globale, sinon elle sera locale

Le scope d'une variable

Differences entre le mot réservé *var* et [*let / const*] :

let (scoped at all block) :

```
1 let a = 10;
2 console.log(a); // => 10
3 let a = 20; // => SyntaxError: Identifier 'a'
4 // has already been declared
5 console.log(a);
```

```
1  function sayHello (pseudo) {
2    let b = 42;
3
4    console.log('Hello ' + pseudo);
5    console.log(b); // => 42
6  }
7
8  sayHello('John'); // => Hello John
9
10 console.log(b); // => ReferenceError: a is not defined
```

```
1 if (true) {
2   let a = 150;
3   console.log(a); // => 150
4 }
5
6 console.log(a) // => ReferenceError: a is not defined
```

Une variable déclaré avec *let* sera forcément locale

Le scope d'une variable

Differences entre le mot réservé *var* et [*let / const*] :

const (scoped at all block & can't be assigne anymore) :

```
1 const a = 10;
2 console.log(a); // => 10
3 a = 20; // => TypeError: Assignment to
4 // constant variable.
5 console.log(a);
```

```
1  function sayHello (pseudo) {
2    const b = 42;
3
4    console.log('Hello ' + pseudo);
5    console.log(b); // => 42
6  }
7
8  sayHello('John'); // => Hello John
9
10 console.log(b); // => ReferenceError: a is not defined
```

```
1 if (true) {
2   const a = 150;
3   console.log(a); // => 150
4 }
5
6 console.log(a) // => ReferenceError: a is not defined
```

Une constante déclaré avec *const* sera forcément locale

Les structure de contrôles

```
if (true) {
   // this_block_is_executed ?
  console.log("Hello dude");
}
else if (true) {
  // this_block_is_executed ?
  console.log("Hello bro");
}
else {
  // this_block_is_executed ?
  console.log("Hello Everyone");
}
```

If / else if / else

```
if (true && false) {
   // this_block_is_executed ?
  console.log("Hello dude");
}
else if (false || false) {
  // this_block_is_executed ?
  console.log("Hello bro");
}
else if (false || true) {
  // this_block_is_executed ?
  console.log("Hello Boss");
}
else {
  // this_block_is_executed ?
  console.lgo("Hello Everyone");
}
```

Les structure de contrôles

```
const expression = 15;

switch (expression) {
  case 5:
    // this_block_is_executed ?
    console.log("Hello dude");
    break
  case 10:
    // this_block_is_executed ?
    console.log("Hello world");
    break
  default:
    // this_block_is_executed ?
    console.log("Bye bye");
    break
}
```

Switch case

```
const expression = 15;

switch (expression) {
  case 5:
    // this_block_is_executed ?
    console.log("Hello dude");
    break
  case 10:
    // this_block_is_executed ?
    console.log("Hello world");
    break
  case 15:
    // this_block_is_executed ?
    console.log("Hello world");
  case 20:
    // this_block_is_executed ?
    console.log("And everyone !");
    break
  default:
    // this_block_is_executed ?
    console.log("Bye bye");
    break
}
```

Les structure de contrôles

Le ternaire

```
1 const age = 29;
2
3 // condition ? <expression si vrai> : <expression si faux>
4 console.log("Je suis " + (age >= 18 ? "majeur" : "mineur"));
```

Les Boucles

```
let i = 0;

while (i < 3) {
  console.log("Je compte " + i);

  if (i == 1) {
    break;
  }

  i++;
}
```

La boucle While

```
const limit = 2;
let i = 10;

while (i >= limit) {
  console.log("Je compte " + i--);
}
```

Les Boucles

```
1 const eleves = ['Jean', 'Marc', 'Marie'];
2
3 for (let i = 0; i < eleves.length; i++) {
4     console.log(eleve[i]);
5 }
```

La boucle For

Les Boucles

La boucle For...in

La boucle For...of

La boucle Array.prototype.forEach()

Les Fonctions

```
1 function sayHello (pseudo) {
2   console.log('Hello ' + pseudo);
3 }
4
5 sayHello("Jean"); // Hello Jean
```

Fonction nommé

Fonction annonyme

```
1 const sayHello = function (pseudo) {
2   console.log('Hello ' + pseudo);
3 }
4
5 sayHello("Jean"); // Hello Jean
```

Les Fonctions

Fonction fléché (*arrow function*)

```
1 const sayHello = pseudo => {
2   console.log('Hello ' + pseudo);
3 }
4
5 sayHello("Jean"); // Hello Jean
```

```
1 const sayHello = pseudo => console.log('Hello ' + pseudo);
2
3 sayHello("Jean"); // Hello Jean
```

```
1 const sayHello = pseudo => {
2   return 'Hello ' + pseudo;
3 }
4
5 const result = sayHello("Jean");
6 console.log(result); // Hello Jean
```

```
1 const sayHello = pseudo => 'Hello ' + pseudo;
2
3 const result = sayHello("Paul");
4 console.log(result); // Hello Paul
```

Les Fonctions

Fonction fléché (*arrow function*)

```
1  const getUserInfos = (last, first) => `${first} ${last}`;
2
3  const sayHello = user => {
4    const lastName = user.lastName;
5    const firstName = user.lastName;
6
7    return `Hello ${getUserInfos(lastName, firstName)}, welcome to the JS class`;
8  }
9
10 sayHello({ lastName: "Pierre", firstName: "Stella" }); // Hello Stella Pierre, welcome to the JS cl
```

TP

JS

Rejoignez la classroom suivante :

https://repl.it/classroom/invite/HfBZBNm

Conclusion

Nous avons vu :

- Les variables
- Les de structure de contrôles
- Les boucles
- Les fonctions

Les bases classiques de n'importe quel languages