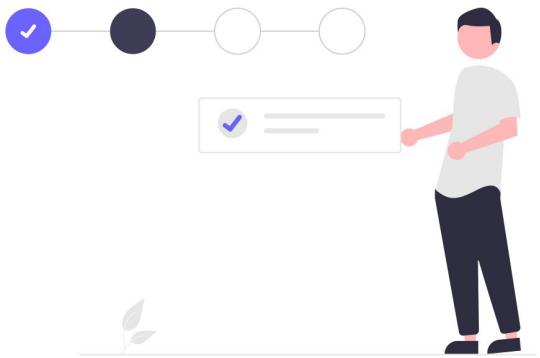
Javascript

Objectifs

1. **Découvrir** les types de données
2. **Vérifier** le type d'une expression
3. **Manipuler** les types

Découvrir les types de données



Les types de données

JavaScript est un langage dont le **typage est faible et dynamique.**

Cela signifie qu'il n'est **pas nécessaire de déclarer le type d'une variable avant de l'utiliser.**

Le type de la variable sera automatiquement déterminé lorsque le programme sera exécuté. Cela signifie également que la même variable pourra avoir différents types au cours de son existence :

```
let toto = 42;      // toto est un nombre
    toto = 'truc'; // toto est désormais une chaîne de caractères
    toto = true;    // toto est désormais un booléen
```

Les types de données

L'ensemble des types disponible en JavaScript se compose des **valeurs primitives** et **valeurs non-primitives**.

Lorsque l'on déclare une variable en JS, notre ordinateur alloue un espace de mémoire à cette variable. Cet espace peut-être alloué de 2 façons :

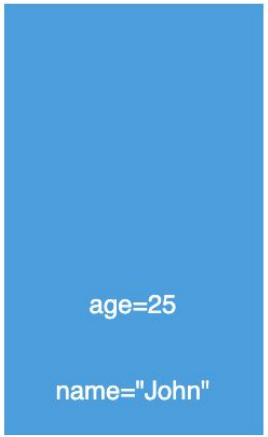
- Par **Valeur**
- Par **Référence**

Les types de données

- **Par valeur**
  - la taille de l'espace de mémoire est **fixe**.
  - La variable est stockée dans une **pile statique**

```
let name = 'John';

let age = 25;
```

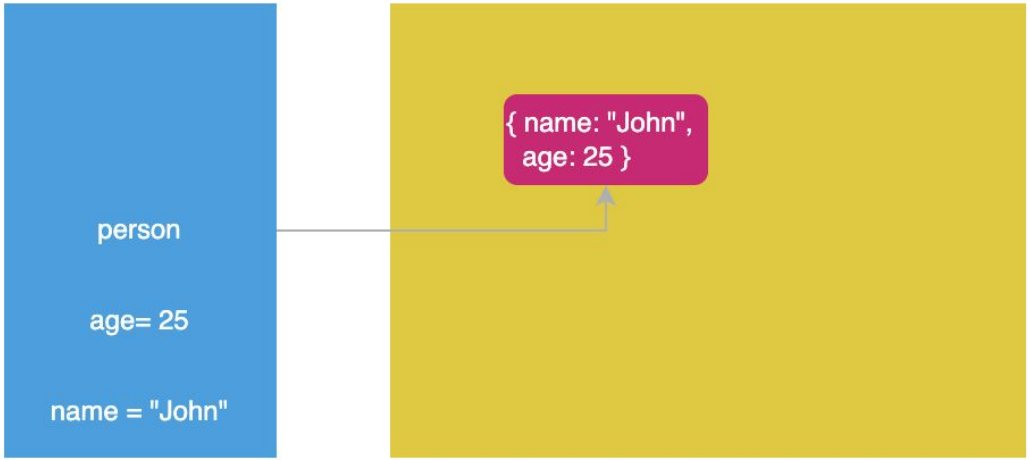


Les types de données

- **Par Référence**
  - La taille de l'espace de mémoire **n'est pas fixée.** L'espace alloué dépend de l'espace que nécessite la valeur. Cette valeur peut évoluer..
  - La variable est stockée dans une **pile dynamique**

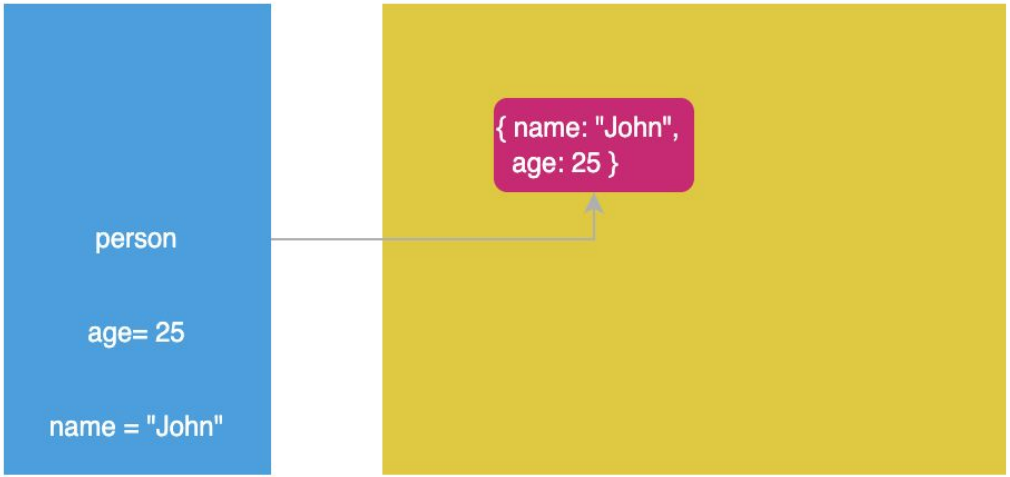
```
let name = 'John';
let age = 25;

let person = {
  name: 'John',
  age: 25,
};
```



```
let name = 'John';
let age = 25;

let person = {
  name: 'John',
  age: 25,
};
```



1. JS alloue 3 emplacements dans la mémoire statique pour 3 éléments : **name**, **age**, **person**
2. JS alloue un espace de mémoire pour **un objet** dans la mémoire dynamique
3. JS lie l'objet dynamique à la variable statique "**person**"

Les types de données

Conséquences

- **Variable stockée par Valeur**
  - N'a pas de propriétés : on ne peut donc pas en ajouter/éditer ou supprimer
  - Si on la copie, JS crée une nouvelle variable statique possédant une valeur identique. Les 2 variables sont **séparées** : si on modifie l'une des variables, cela ne modifiera **pas** l'autre

- **Variable stockée par Référence**
  - Peut être modifiée à tout moment : on peut ajouter, éditer ou supprimer ses propriétés
  - Si on la copie, JS crée une nouvelle variable statique pointant vers le même objet de référence. Les 2 variables sont **liées** : Si on modifie l'une des variables, cela modifiera l'autre

Les types de données

Conséquences

La différenciation Valeur/Référence est surtout importante lorsque l'on souhaite **copier nos variables.**

JS

Exemple : copie d'une variable stockée par valeur

```
let age = 25;
let newAge = age;
```

Variable stockée par Valeur

Les 2 variables sont séparées : modifier la valeur de **age** ne modifiera pas la valeur de **newAge**

Exemple : copie d'une variable stockée par Référence

```
let person = {
  name: 'John',
  age: 25,
};
```

Variable stockée par Référence

1. On déclare une variable **person** et on lui assigne une valeur qui est un objet (qui a pour propriétés **name** et **age**)

JS

Exemple : copie d'une variable stockée par Référence

```
let person = {
  name: 'John',
  age: 25,
};

let member = person;
```

Variable stockée par Référence

1. On déclare une variable **member** et on lui assigne la valeur **person**

Exemple : copie d'une variable stockée par Référence

Variable stockée par Référence

2. Deux variables statiques sont créées, mais **toutes deux pointent vers la même référence** (vers le même objet)

Exemple : copie d'une variable stockée par Référence

```
let person = {
  name: 'John',
  age: 25,
};

let member = person;

member.age = 26;

console.log(person);
console.log(member);
```

Variable stockée par Référence

3. On modifie la valeur de la propriété **age** de la variable **member**

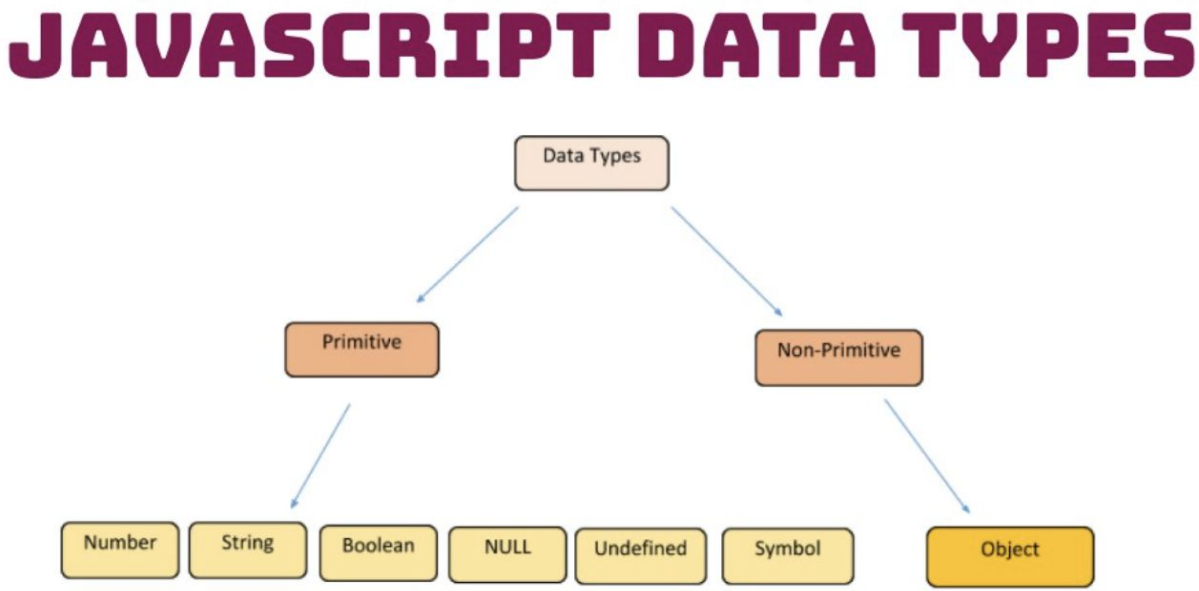
Exemple : copie d'une variable stockée par Référence

Variable stockée par Référence

4. La valeur de la propriété **age** **est modifiée pour les 2 variables**

Les types de données

Il existe 7 types de données en JS, répartis en primitifs/non-primitifs



Les types de données

Les données primitives

- Le type **boolean**

Un booléen représente le résultat d'une assertion logique et peut avoir deux valeurs :

- **true** (pour le vrai logique)
- **false** (pour le faux logique)

```
const isUserLoggedIn = false;
```

Les types de données

Les données primitives

- Le type **null**

Le type null représente une **absence intentionnelle de valeur** et ne possède qu'une valeur :

- **null**

```
const myVar = null;
```

Les types de données

Les données primitives

- Le type **undefined**

Une variable à laquelle on n'a pas affecté de valeur aura par défaut la valeur undefined. On peut aussi assigner la valeur d'une variable à undefined. Ne prend qu'une valeur :

- **undefined**

```
const account = undefined;
let account2;
```

Les types de données

Les données primitives

- Le type **number**

Permet de représenter un nombre. Plus de détails sur les limites stockables du type nombre ici.

- **un nombre sur lequel il est possible d'effectuer des opérations arithmétiques**

```
let age = 123;
```

Les types de données

Les données primitives

- Le type **Bigint**

Permet de représenter et de manipuler un nombre, en particulier un nombre que l'on ne pourrait pas représenter et manipuler avec un number.

- un nombre à la fin duquel on ajoute **n.** Exemple **: 7n**
- un nombre que l'on "entoure" de **Bigint.** Exemple **: BigInt(992001).**

```
let ageWihtBigInt = BigInt(age);
```

Les types de données

Les données primitives

- Le type **string**

Utilisé afin de représenter des données de texte. Chaque élément occupe une position au sein de cette chaîne de caractères. Le premier élément est situé à l'indice 0, le deuxième à l'indice 1 et ainsi de suite.

- “quelque chose”

```
const firstname = "Michel";
```

Les types de données

Les données primitives

- Le type **Symbol**

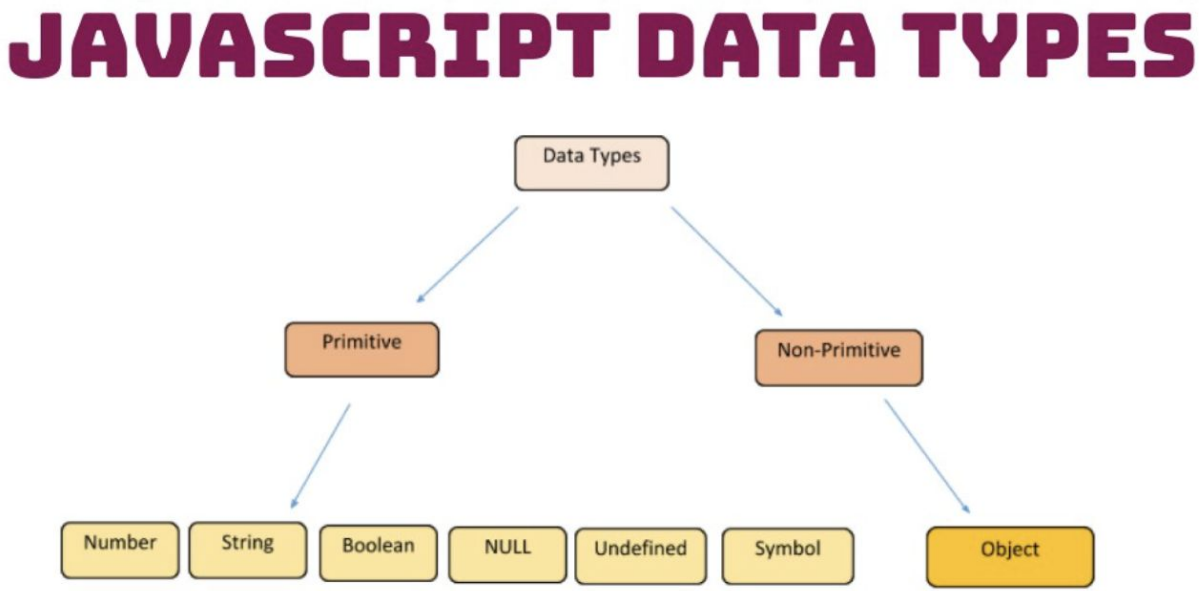
Un symbole est un type représentant une donnée **unique** et **inchangeable.**

- **Symbol()**

```
const aUniqueValue = Symbol(42);
```

Les types de données

Il existe 7 types de données en JS, répartis en primitifs/non-primitifs



Les types de données

Les données primitives

- Le type **Object**

les objets sont en quelque sorte des "boîtes" qui peuvent contenir des *clés* associées à des *valeurs*, un peu comme dans un dictionnaire où on a des mots associés à des définitions. Il y a 3 grands types d'objets :

- Objets littéraux (classiques)

Les types de données

Les données primitives

- Le type **Object**

les objets sont en quelque sorte des "boîtes" qui peuvent contenir des *clés* associées à des *valeurs*, un peu comme dans un dictionnaire où on a des mots associés à des définitions. Il y a 3 grands types d'objets :

- Tableaux

```
const colors = ['red', 'blue', 'green'];
```

Les types de données

Les données primitives

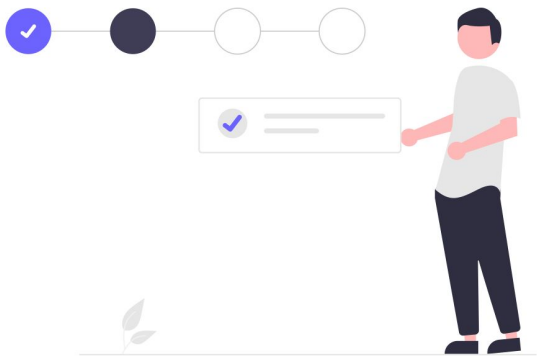
- Le type **Object**

les objets sont en quelque sorte des "boîtes" qui peuvent contenir des *clés* associées à des *valeurs*, un peu comme dans un dictionnaire où on a des mots associés à des définitions. Il y a 3 grands types d'objets :

- Fonctions

```
const onSayHello = function sayHello() {
    return 'Hello';
}
```

Vérifier le type d'une expression



Vérifier le type d'une expression : l'opérateur typeof

En javascript, on peut utiliser une fonction appelée "**typeof**" pour voir le type de données d'une valeur.

Vérifier le type d'une expression : l'opérateur typeof

En javascript, on peut utiliser une fonction appelée "**typeof**" pour voir le type de données d'une valeur.

Pour ce faire, on peut simplement écrire **typeof** suivi de la valeur que nous voulons vérifier.

Vérifier le type d'une expression : l'opérateur typeof

En javascript, on peut utiliser une fonction appelée "**typeof**" pour voir le type de données d'une valeur.

Pour ce faire, on peut simplement écrire **typeof** suivi de la valeur que nous voulons vérifier.

```
console.log(typeof (5*2));
console.log(typeof 'coucou');
console.log(typeof 5*2);
```

Vérifier le type d'une expression : l'opérateur typeof

En javascript, on peut utiliser une fonction appelée "**typeof**" pour voir le type de données d'une valeur.

Pour ce faire, on peut simplement écrire **typeof** suivi de la valeur que nous voulons vérifier.

```
console.log(typeof (5*2));
console.log(typeof 'coucou');
console.log(typeof 5*2);
```

```
number
string
NaN
```

Pratiquons !