# Javascript

## Les objets

# Définition

Un objet est une collection de propriétés.
Une propriété est une association entre un nom et une valeur.
La valeur d'une propriété contient une valeur primitive ou un objet.
Si cet objet est une fonction on parle alors d’une méthode.

# Création

## Constructeur

Peu utilisé

```
let toto = new Object();
toto.nom = "Toto";
toto.age = 25;
```

## Notation littérale

La plus répandue et conseillée.

```
let toto = {
  "nom" : "Toto",
  "age" : 25
};
```

# Notation littérale

## Exemple

```
let toto = {
    "nom" : "Toto",
    "activites" : { //la propriété activites est elle-même un objet
        "jour" : "travail",
        "nuit" : "dodo"
    }
};

let titi = {
    "nom" : "Titi",
    "activites" : null
};

let personnes = { "toto" : toto, "titi" : titi };
```

# Notation littérale

## Rappel des règles des identifiants

- commencent par une lettre, un tiret bas (_) ou un dollar ($)
- les caractères qui suivent peuvent être des chiffres (0 à 9)
- sensibles à la casse
- ne sont pas des mots réservés

# Notation littérale

## Noms de propriétés

Si les noms de propriétés respectent les règles des identifiants en javascript, les guillemets (simples ou doubles) sont optionnels.

```
let toto = {
    "nom" : "Toto",
    'age' : 25,
    genre : "M"
}; //ok
```

Cela peut conduire à des écritures un peu déroutantes :

```
let toto = { nom : "Toto", age : 25 };
let tata = { nom : "Tata", age : 22 };

let personnes = { toto : toto, tata : tata };
```

Depuis ES6, cela peut s'écrire :

```
let toto = { nom : "Toto", age : 25 };
let tata = { nom : "Tata", age : 22 };

let personnes = { toto, tata };
```

# Notation littérale

## Noms de propriétés

Si les noms de propriétés ne respectent pas les règles des identifiants en javascript, les guillemets (simples ou doubles) sont obligatoires.

```
let toto = {
    1-nom : "Toto",
    2-age : 25
};
```

```
let toto = {
    "1-nom" : "Toto",
    '2-age' : 25
};

//ok
```

# Format JSON

## JavaScript Oject Notation

La clarté et la concision de la notation littérale des objets en javascript ont donné naissance au format JSON, devenu le format d'échange de données le plus répandu dans le monde du web.

# Notation littérale vs JSON

Limitation du JSON par rapport à la notation littérale :

- Le nom de la propriété doit être entouré de guillemets doubles.
- Les valeurs ne peuvent être uniquement que des chaînes de caractères, des nombres, des tableaux, true, false, null, ou tout autre objet (JSON).
- Une fonction ne peut pas être affectée comme valeur.
- Les objets Date seront convertis en chaînes de caractères.

# Accès aux propriétés

## Notation avec point

```
let toto = {
    nom : "Toto",
    age : 25
};

toto.age; //25
```

# Accès aux propriétés

## Notation avec crochets

```
let toto = {
    nom : "Toto",
    age : 25
};

toto["age"]; //25
```

Les noms de propriétés peuvent dans ce cas ne pas respecter les règles des identifiants de variables.

```
let toto = {
    "1-nom" : "Toto",
    "2-age" : 25
};

toto["2-age"]; //25
```

# Accès aux propriétés

Une tentative d'accès à une propriété non définie ne renvoie pas d'erreur mais la valeur primitive `undefined` :

```
let toto = { nom : "Toto", age : 25 };

toto.nbEnfants; //undefined
```

On peut également tester l'existence d'une propriété par l'opérateur `in` :

```
let toto = { nom : "Toto", age : 25 };

"nbEnfants" in toto; //false
```

# Affectation par décomposition

```
let toto = {
  nom : "Toto",
  age : 25
};

let { age, nom } = toto;

nom; // "Toto"
age; // 25
```

Equivalent à :

```
let toto = {
  nom : "Toto",
  age : 25
};

let age = toto.age;
let nom = toto.nom;
```

# Affectation par décomposition

## Avec un nom différent

```
let toto = {
  nom : "Toto",
  age : 25
};

let { age : ageToto, nom : nomToto } = toto;

ageToto; //25
nomToto; //"Toto"
```

# Affectation par décomposition

## A plusieurs niveaux

```
let toto = {
  nom : "Toto",
  age : 25,
  telephone : {
    portable : "06 25 32 XX XX",
    pro : "05 61 07 XX XX"
  }
};

let { telephone : { portable } } = toto;

portable; //"06 25 32 XX XX";
telephone; // ReferenceError: telephone is not defined
```

# Affectation par décomposition

## Affecter le reste

```
let toto = {
  nom : "Toto",
  age : 25,
  adresse : "24 rue Mozart",
  profession : "informaticien"
};

let { nom, age, ...autres } = toto;

autres; //{ adresse : "24 rue Mozart", profession : "informaticien" };
```

# Noms de propriétés calculés

```
let prop = "nom";

let monObjet = {
  [prop] : "Toto"
};

monObjet.nom; // Toto
```

# Enumération des propriétés

## boucle for...in

```
let toto = {
  nom : "Toto",
  age : 25,
  genre : "M"
};

for (let key in toto) {
  console.log(key, toto[key]);
}
```

# Enumération des propriétés

## Object.keys

### Renvoie un tableau des noms de propriétés

```
let toto = {
    nom : "Toto",
    age : 25,
    genre : "M"
};

let tab = Object.keys(toto);

console.log(tab); //"nom,age,genre"
```

# Enumération des propriétés

## Object.values

### Renvoie un tableau des valeurs de propriétés

```
let toto = {
      nom : "Toto",
      age : 25,
      genre : "M"
  };

  let tab = Object.values(toto);

  console.log(tab); //"Toto,25,M"
```

Les tableaux font l'objet d'un chapitre dédié.

# Suppression d'une propriété

## Opérateur `delete`

```
let toto = {
  nom : "Toto",
  age : 25,
  genre : "M"
};

delete toto.genre;
delete toto.prenom; // ne génère pas d'erreur

console.log("genre" in toto); //false
```

# Objets intégrés

Valables dans tous les environnements d'exécution :

- `Object` : constructeur d'objets
- `Function` : constructeur de fonctions
- `Array` : constructeur de tableaux
- `String` : constructeur d'objets chaînes de caractères
- `Boolean` : constructeur d'objets booléens
- `Number` : constructeur d'objets nombres
- `Math` : objet natif pour fonctions mathématiques
- `Date` : constructeur d'objets dates
- `RegExp` : constructeur d'expressions rationnelles
- `JSON` : objet natif pour manipulation du JSON
- `Error (Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError)` : constructeurs d'erreurs

# Objets intégrés

## Ajouts dans la norme ECMAScript 6

- `ArrayBuffer`
- `DataView`
- `Float32Array`
- `Float64Array`
- `Int8Array`
- `Int16Array`
- `Int32Array`
- `Map`
- `Proxy`
- `Promise`
- `Set`
- `Symbol`
- `Uint8Array`
- `Uint8ClampedArray`
- `Uint16Array`
- `Uint32Array`
- `WeakMap`
- `WeakSet`
- `Reflect`

# Objet global

Objet intrinsèque qui représente l'espace global dans un environnement javascript. Il est créé lors de l'initialisation du moteur de script.

Définir une variable globale revient à définir une propriété de l'objet global.

Dans l'environnement d'un navigateur web, cet objet global est **`window`**.

```
typeof window; //"object"

window.toto = "tata";

toto; //"tata"

function add(a,b) { return a+b; }

add(2,1); //3
window.add(2,1); //3
```

# Objet global

## globalThis

Référence vers l'objet global de l'environnement, et donc indépendamment de celui-ci.

Dans l'environnement d'un navigateur web, **`globalThis`** et **`window`** sont le même objet.

# Références

Contrairement aux valeurs primitives, une variable à laquelle on affecte un objet contient une référence vers cet objet

```
let toto = {  //toto contient une référence vers l'objet
  nom:"Toto",
  age:26
};

let tata = toto; //tata contient une copie de la référence vers l'objet

tata.nom = "Tata";
toto.nom; //Tata

tata = null;
toto; //{"nom"":"Tata",age:26}
```

# Cloner un objet

## Clonage superficiel (shallow cloning)

```
let toto = {
  nom:"Toto",
  age:26
};

let tata = { ...toto }; //tata contient une copie de la référence vers l'objet

tata.nom = "Tata";
toto.nom; //Toto
```

# Cloner un objet

## Problème

```
let toto = {
  nom:"Toto",
  age:26,
  activites : {
    jour : "travail",
    nuit : "dodo"
  }
};

let tata = { ...toto };

tata.activites.jour = "sport";
toto.activites.jour; // sport
```

# Cloner un objet

## Clonage profond (deep cloning)

```
let toto = {
  nom:"Toto",
  age:26,
  activites : {
    jour : "travail",
    nuit : "dodo"
  }
};

let tata = structuredClone(toto);

tata.activites.jour = "sport";
toto.activites.jour; // travail
```

Remarque : les fonctions ne sont pas clonables.