Javascript : Bases sur les objets

1. Introduction

Tout ce qui n’est pas primitive.

C’est un ensemble de clef / valeur.

Clef : string ou symbol

Valeur : primitive ou objet

Il y a les objets littéraux, les fonctions, Dates, Tableau.

* Déclaration :

Notion litterale :

const a = {}

const a = Object() ; // donne la meme chose

const a = new Object(); // donne la meme chose

1. Propriétés des objets

Avec une déclaration litérale :

Accès à une propriete :

obj.property

obj.[‘property’]

obj.[variable]

Ne pas oublier que c’est des ref :

const a = {a : 5} ;

const b = a ;

b.a = 4 ;

**modifie la meme reference**

Une valeur peut etre une fonction (qui est un objet), c’est une méthode.

Construction :

{

a:4,

b :{}, // on peut imbriquer des objets en valeur

maMethode : function(){},

maMethode2(){

}, // ecriture qui fonctionne aussi pour les fonctions, plus courte

}

1. Raccourcis syntaxiques

Depuis ES6 :

Cela permet de raccourcir la déclaration de propriétés pour les objets littéraux.

On souhaite que le nom d’une propriété correspond au nom d’une variable.

V1 :

const a = 1;

const b = 42;

const c = {a: 2};

const monObjet = {

a: a,

b: b,

c: c

}; // {a: 1, b: 42, c: {a: 2}}

Raccourci syntaxique V2 :

const a = 1;

const b = 42;

const c = {a: 2};

const monObjet = {

a,

b,

c,

}; // {a: 1, b: 42, c: {a: 2}}

Autre exemple : On peut simplifier cette écriture :

const prop1 = "Une chaîne de caractères";

const monObjet = {};

monObjet[prop1] = "test";

console.log(monObjet); // {Une chaîne de caractères: "test"}

Par :

const prop1 = "Une chaîne de caractères";

const monObjet = {

[prop1]: "test",

};

console.log(monObjet); // {Une chaîne de caractères: "test"}

1. Décomposition d’objet

L’affectation par décomposition permet d’extraire des données d’un objet.

Ici il est assez simple à partir d’un objet d’en retourner des variables :

* **Cas simple (nom des variables sont identiques):**

const unObjet = { toto: 1, tutu: 2};

const { toto:toto, tutu:tutu } = unObjet;

// Équivaut à :

const { toto, tutu } = unObjet;

console.log(toto); // 1

console.log (tutu); // 2

* **Si on utilise des noms différents :**

//on constate juste que le nom des variables se trouvent à doite.

const unObjet = { toto: 1, tutu: 2};

const { toto:a, tutu:b } = unObjet;

console.log(a); // 1

console.log (b); // 2

* **Cas des valeurs par défaut si on ne trouve pas une propriété à déstructurer :**

const monObjet = {a: 42};

const {a = 10, b = 1} = monObjet;

console.log(a); // 42

console.log(b); // 1

* **Cas complexe d’un objet imbriqués :**

const personne = {

nom: 'Jean',

famille: {

pere: {

prenom: 'Bob',

nom: 'Dylan',

}

}

};

const { famille: { pere } } = personne;

console.log(pere); // Object { nom: 'Dylan', prenom: 'Bob' }

console.log(famille); // undefined

* **Le rest operateur !**

Utile pour récupérer le reste des données.

const {a, b, ...monReste } = {a: 1, b: 2, c: 3, d: 4};

console.log(a); // 1

console.log(b); // 2

console.log(monReste); // {c: 3, d: 4}

1. Tester l’existence et la valeur d’une propriété

Si une propriété retourne undefined c’est qu’elle n’existe pas.

const monObj = {};

monObj.a === undefined; // true

On peut utiliser aussi l’opérateur in pour vérifier l’existence d’une propriété dans l’objet.

Même si la propriété est initialisée à undefined in renverra true pour indiquer qu’elle a été déclarée.

const monObj = {a: 1};

"a" in monObj; // true

"b" in monObj; // false

La méthode .hasOwnProperty(«property») permet de vérifier l’existance d’une propriété sur un objet.

1. Supprimer ou écarter des propriétés

* Pour indiquer qu’une propriété n’a pas de valeur on la positionne à null.

const monObj = {a: null};

* Pour supprimer une propriété on utilise delete

const monObj = {a: 1, b:2};

delete monObj.a;

delete monObj['b'];

console.log(monObj); // {}

* Il est possible de creer un nouvel objet en écartant tout simplement des propriétés avec le rest operator et le déstructuration.

const {a, b, ...copie } = monObj;

console.log(copie); // {c: 3, d: 4}

copie ne contiendra que les propriétés en rest.

1. Fusionner des objets :
2. introduction

La méthode Object.assign() permet de copier toutes les propriétés d’un ou plusieurs objets sur une autre objet cible.

Object.assign(objetCible, objetACopier1, objetACopier2...)

const objet1 = {a: 1, b: 2};

const objet2 = {c: 3};

const résultat = Object.assign({}, objet1, objet2);

console.log(résultat); // {a: 1, b: 2, c: 3}

Si on a plusieurs objets avec la meme propriete, la derniere a raison !

1. utilisation du spread operator.

Cet opérateur a la même syntaxe que le rest operator.

La différence entre les deux deux est le contexte d’utilisation.

Le spread operator permet d’étendre un objet, c’est-à-dire de lister ses propriétés.

* Exemple :

const monObj = {a: 1};

const copie = {...monObj};

copie.a = 2;

console.log(monObj); // {a: 1}

console.log(copie); // {a: 2}

* on peut fusionner plusieurs objets ainsi.

const monObj = {a: 1};

const monObj2 = {b: 2};

const copie = {...monObj, ...monObj2};

copie.a = 2;

console.log(monObj); // {a: 1}

console.log(copie); // {a: 2, b: 2}

1. Comparer des objets :

Attention les variables contiennent des références.

Quand on compare on compare … les références !

Les objets sont donc comparés par référence :

const obj = {a: 1};

const obj2 = obj;

console.log(obj === obj2); // true car même référence

const obj3 = {a: 1};

console.log(obj === obj3); // false

// Même chose ici :

console.log( {} === {}); // false

1. Itération sur des objets

Grâce à for…in on peut itérer sur les objets.

const obj = {a: 1, b: 2, c: 4};

* Accéder aux properties :

for (const prop in obj) {

console.log(prop);

}

// a

// b

// c

* Accéder à leur valeur :

const obj = {a: 1, b: 2, c: 4};

for (const prop in obj) {

console.log(`${prop}: ${obj[prop]}`);

}

// a: 1

// b: 2

// c: 4

* Des méthodes utiles avec Object.

Object.values(obj) // retourne un array des valeurs

Object.keys(obj) // retourne un array des keys

Objetc.entries(obj) // retourne un array d’array key/value.

1. Format JSON

Formation JSON (Javascript Object Notation)

Utilisation de l’API JSON.

* Transformer un objet en chaine de caracteres.

Attention une propriété undefined sera toujours omise.

* JSON.stringify(obj) ;

Cette méthode peut prendre 3 arguments.

* Le deuxieme est l’argument de remplacement, très peu voir jamais utilisé, il est préférable de faire des traitements sur l’objet avant de passer par le stringify.
* Le troisième est utile pour la mise en forme, c’est l’option d’espae qui permet de déterminer l’espace pour l’affichage. (sous forme d’entier ou de chaine d’espace)

JSON.stringify({ a: 1, b: 2 }, null, 3);

// {

// "a": 1,

// "b": 2

// }

* JSON.parse() permet d’effectuer l’inverse, à partir d’un string revenir sur un objet.

1. Copier un objet.

Il existe deux manières de copier un objet.

* La superficielle est la profonde.

**CAS SUPERFICIEL :**

* Ceci permet d’avoir deux variables avec la meme reference :

const obj = {};

const obj2 = obj;

* Ici une copie avec assign() ou le spread operator :

const obj = {a: 1};

const obj2 = {...obj};

Mais attention ! cela est une copie superficielle, ca va copier toutes les propriétés de premier niveau sur une nouvelle référence obj2.

Cependant si on a des propriétés qui contiennent des objets… on aura ces propriétés qui pointeront vers le meme objet mémoire… ce qui peut être dramatique.

Exemple :

const obj = {a: {}};

const obj2 = {...obj};

obj2.a.test = 42;

console.log(obj); // {a: {test: 42}}

**CAS PROFONDEUR :**

On pourra créer des fonctions, mais cela serait particulièrement long et chronophage.

Soyons malin en utilisant l’API JSON.

const obj = {a: {}};

const obj2 = JSON.parse(JSON.stringify(obj));

obj2.a.test = 42;

console.log(obj); // {a: {}}

1. Chainage optionnel
2. Utilisation du chainage optionnel

L’opérateur ? permettra de continuer que si la propriété précédente est existante :

let street = user?.details?.address?.street;

console.log(street); // "Rue de Rivoli" si toutes les propriétés existent, sinon `undefined`

Cela fonctionne aussi avec les fonctions :

const utilisateur = {

prenom: "Jean"

};

const utilisateur2 = null;

console.log( utilisateur1?.[prenom] ); // Jean

console.log( utilisateur2?.[prenom] ); // undefined

* Attention restriction du chainage optionnel :

Le chainage ne convient pas à cette écriture :

const utilisateur = null;

utilisateur?.prenom = "Jean"; // Erreur, ne fonctionne pas car cela revient à faire undefined = "Jean"