**La POO**

On va d’abord faire un rappel des objets…

De base en JS les objets ils peuvent être de toute sorte

const obj = {

    pseudo: "Papiico",

    ville: "Thies"

}

console.**log**(typeof obj);//ça affiche object dans la console

let array = [1, 2, 3]

console.**log**(typeof array); //ça affiche object

console.**log**(document.body);

console.**log**(typeof document.body)

console.**log**(typeof null)

Mais le plus souvent les éléments considérés comme étant un objet sont ceux qui ont des accolages

* On peut ajouter un attribut à un objet en mettant le nom de l’objet.leNomDuNouvelAttribut
* const obj = {
* pseudo: "Papiico",
* ville: "Thies"
* }
* //Ajouter un paramètre
* obj.adresse = "Lalale"
* On peut supprimer un paramètre en utilisant **delete**

//Supprimer un élément en utilisant (delete)

delete obj.adresse

* Nous pouvons aussi interroger (Demander à JavaScript si une certaine propriété existe
* console.**log**("pseudo" in obj)//Renvoie true
* console.**log**("adresse" in obj)//Affiche false
* Parcourir un objet pour accéder à ses éléments

//Interroger pour savoir si une certaine propriété existe

console.**log** ("pseudo" in obj)//Renvoie true

console.**log** ("adresse" in obj)//Affiche false

//Parcourir un objet

for (const key in obj) {

   console.**log** (key)

}

* Jusque-là, nous avons travaillé avec des objets basique… Mais nous pouvons par exemple lui ajouter des méthodes par exemple
* const newObj = {
* pseudo: "Mohamed",
* age: 25,
* **direBonjour**(){
* console.**log**("Salut !")
* }
* }
* newObj.**direBonjour**()

La console va nous afficher **« Salut ! »**

* **LE THIS**

Le **this** fait référence à l’objet… Disons qu’il est utilisé pour faire référence à l’objet courant, il est très pratique quand nous voulons accéder aux paramètres de la fonction dans ses méthodes

const newObj = {

    pseudo: "Mohamed",

    age: 25,

**direBonjour**: function () {

        console.**log**(`Salut je m'appelle ${this.pseudo}`)

    }

}

newObj.**direBonjour**()

1. Les méthodes natives du JS pour nous aider à manipuler les objets

* **Object keys()** nous permet de récupérer les différentes clés de l’objet

//Obtenir les clés d'un objet

const keys = Object.**keys**(newObj)

console.**log**(keys);

* **Object.values ()** permet de récupérer les différentes valeurs des clés

//Obtenir les valeurs de l'objet

const values = Object.**values**(newObj);

console.**log**(values)

const restArray = Object.**entries**(obj);

console.**log**(restArray)

//On peut fusionner les objets

* **Object.assign ()** nous permet de fusionner deux objets

//On peut fusionner les objets

const fusion = Object.**assign**({}, obj, newObj);

console.**log**(fusion);

NB : Si nos deux objets ont un attribut en commun, c’est le dernier annoncé qui l’emporte

* **Object.seal()** permet d’empêcher les modifications, on peut bien changer les valeurs des paramètres mais on ne peut pas en ajouter
* const otherObject = Object.**seal**(obj);

1. **CONSTRUIRE DES OBJETS**

Nous allons voir 3 constructeurs

1. **Fonction constructeur**

C’est à la base une fonction

function **User**(pseudo, ville){

    this.pseudo = pseudo;

    this.ville = ville;

}

const user1 = new **User**("Joe", "Thiès"); //Ceci est une instance de l'objet User

console.**log**(user1);

const user2 = new **User**("Papiico", "Dakar");//On créer une nouvelle instance

console.**log**(user2);

* Nous pouvons aussi ajouter des méthodes (Ici nous verrons mieux comment on utilise le **this**

function **User** (pseudo, ville) {

    this.pseudo = pseudo;

    this.ville = ville;

    this.**getCity** = function () {

        console.**log**(`${this.pseudo} habite à ${this.ville}`);

    }

}

const user1 = new **User**("Joe", "Thiès"); //Ceci est une instance de l'objet User

const user2 = new **User**("Papiico", "Dakar");//On créer une nouvelle instance

user1.**getCity**();

user2.**getCity**();

1. **Factory functions**

C’est la méthode la plus récente qui a été créé et qui nous permet de créer des objets

//------------------------Factory Functions----------------------

function **User3** (pseudo, ville) {

    return {

        pseudo: pseudo,

        ville: ville

    }

}

//Ici on n’a pas besoin de faire un NEW

const user4 = **User3**('Christine', "Dakar");

console.**log**(user4);

On a pas besoin d’utilise le New ici pour instancier un objet

1. Class(Méthode la plus utilisée)

//---------------------CLASS (Méthode la plus utilisée)-----------------------

class Utilisateur {

    constructor(pseudo, ville){

        this.pseudo = pseudo;

        this.ville = ville;

    }

}

const user5 = new **Utilisateur**("Kitty", "Bamako");

console.**log**(user5);

* On peut également ajouter des méthodes :

//Ajout d'une méthode

class Utilisateur {

    constructor(pseudo, ville){

        this.pseudo = pseudo;

        this.ville = ville;

    }

**sayMyName** = function () {

        console.**log**(`Bonjour je suis ${this.pseudo}`);

    }

}

const user5 = new **Utilisateur**("Kitty", "Bamako");

user5.**sayMyName**()

console.**log**(user5);

* On peut ajouter une méthode en passant par le prototype

//On peut ajouter une méthode en passant par le prototype

Utilisateur.prototype.**sayCity** = function () {

    console.**log**(`J'habite à ${this.ville}`);

}

user5.**sayCity**();

* On peut même se créer plusieurs prototypes d’un coup

//On peut ajouter pleins de prototypes d'un coup

Object.**assign**(Utilisateur.prototype, {

**methode1**(){

        //Ma méthode

    },

**methode2**(){

        //Ma méthode

    }

})

console.**log**(user5);

1. L’HERITAGE

L’héritage nous permet de faire hériter des propriétés et attribut d’une classe à d’autres

class Animal {

    constructor(name, age) {

        this.name = name;

        this.age = age

    }

**saySomething**(text){

        console.**log**(this.name + "dit : "+ text);

    }

}

class Dog extends Animal {

**run**(){

        console.**log**("Le chien court !");

    }

}

const rintintin = new **Dog**("Rintintin", 9)

console.**log**(rintintin);

rintintin.**run**();