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()
```

I- 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);
```

II- 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();
```

2- 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

3- 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);
```

III- 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();
```

PROJET : YOGA-APP

- Nous remarquons déjà que nos fichiers images sont numérotés… Nous comprendrons plus tard pourquoi c'est ainsi

Faut déjà savoir que dans ce projet en réalité la page ne change jamais… C'est juste nous qui injectons des choses dans la page (Ce sont des vues)

```
//On pointe d'abord le main (Vu que c'est lui qui va contenir nos cartes)
const main = document.querySelector('main');

//On crée une variable qui va stocker tous nos exercies
//En sachant que notre carte elle a une image, un nombre de minute qui leur soit adosser

let exerciceArray = [

]
```

Donc on va créer un tableau d'objets

```
let exerciceArray = [

  {
    pic: 1,
    min: 1
  },
  {
    pic: 2,
    min: 1
  },
  {
    pic: 3,
    min: 1
  },
  {
    pic: 4,
    min: 1
  },
  {
    pic: 5,
```

```
    min: 1
  },
  {
    pic: 6,
    min: 1
  },
  {
    pic: 7,
    min: 1
  },
  {
    pic: 8,
    min: 1
  },
  {
    pic: 9,
    min: 1
  },
  {
    pic: 10,
    min: 1
  },
]
```

On stocke nos éléments dans un tableau car avec celui-ci ils seront plus facile à manipuler (Suppression, déplacement, etc…) en faite tout ce qu'on va faire sera basé sur ce tableau

- **On se crée une classe (Qui va être le générateur d'exercices) C'est cette classe qui va gérer le passage d'un exercice à l'autre**

```
class Exercice  {}
```

- **On crée un objet 'utils' dans lequel on va mettre toute nos fonction qui vont être utiles dans notre projet**

```
const utils = {

}
```

- **On crée un objet 'page' dans lequel on va mettre toutes nos pages (Vues)**

```
const page = {


}
```

❖ **Petit exemple d'utilisation de la fonction Lobby (Pour comprendre un peu la logique de notre code)**

```
 //On crée un objet 'page' dans lequel on va mettre toutes nos pages (Vues)
  const page = {

      lobby: function(){
        document.querySelector('h1').innerHTML = "Paramétrage <i id='reboot' class='fas
fa-undo'></i>"
      }
  }

  page.lobby();
```

- **On peut ensuite s'ajouter le bouton ''commencer''**

```
  const page = {

      lobby: function(){
        document.querySelector('h1').innerHTML = "Paramétrage <i id='reboot' class='fas
fa-undo'></i>"

        main.innerHTML = "Exercices"
        document.querySelector(".btn-container").innerHTML = "<button
id='start'>Commencer<i class='far fa-play-circle'></i></button>"
      }
  }

  page.lobby();
```

- **Si on veut éviter de se répéter tout le code précédent pour chaque vue, on peut se créer une fonction et mettre ce code-là à l'intérieur**

NB : Ceci va nous permettre de changer les informations affichées à l'écran sans avoir besoin de réécrire à chaque fois le code mais on pourra le faire juste en appelant la fonction 'pageContent'

```
const utils = {
  pageContent: function (title, content, btn) {
    document.querySelector("h1").innerHTML = title;
    main.innerHTML = content;
    document.querySelector(".btn-container").innerHTML = btn;
  },
};
```

- **Utilisation de la méthode pageContent dans les pages :**

```
const page = {
  lobby: function () {

    utils.pageContent(
      "Paramétrage <i id='reboot' class='fas fa-undo'></i>",
```

```
      "Exercices",
      "<button id='start'>Commencer<i class='far fa-play-circle'></i></button>"

    )},

  routine: function () {
    utils.pageContent(
      "Routine", "Exercice avec chrono", null
    )
  },

  finish: function () {
    utils.pageContent(
      "C'est terminé !",
      "<button id='start'>Recommencer</button>",
      "<button id='reboot' class='btn-reboot'>Rénitialiser<i class='fas fa-times-
cicle'></i></button>"
    )
  },
};

page.finish();
```

1- **Maintenant qu'on sait faire ça, on va se coder la logique d'une carte**

A- **Dans la fonction** lobby(), on va se créer un map(Qu'on va stocker dans une variable et qu'on va donner en paramètre à notre fonction pageContent

```
 lobby: function () {

    let mapArray = exerciceArray
    .map(
      (exo)=>
      `
      <li>

      <div class="card-header">
        <input type="number" id="${exo.pic} min="1" max="10" value=${exo.min}>
        <span>min</span>
        </div>

      </li>

      `

    ).join('');
```

B- On passe ensuite la variable **mapArray** comme second paramètre de la fonction **pageContent** dans le **lobby()** pour signifier que le contenu de cette page là est le résultat du map **(Qui a été stocker dan le mapArray)**

```
const page = {
  lobby: function () {

    let mapArray = exerciceArray
    .map(
      (exo)=>
      `
      <li>

      <div class="card-header">
        <input type="number" id="${exo.pic} min="1" max="10" value=${exo.min}>
        <span>min</span>
        </div>

      </li>

      `

    ).join('');

    utils.pageContent(
      "Paramétrage <i id='reboot' class='fas fa-undo'></i>",
      mapArray,
      "<button id='start'>Commencer<i class='far fa-play-circle'></i></button>"

    )},
```

C- AJOUT DES IMAGES À NOS CARTE

Nous devons nous assurez que l'ajout des images soit dynamique (Donc le src va varier en fonction de l'image sur laquelle nous sommes)

```
lobby: function () {

    let mapArray = exerciceArray
    .map(
      (exo)=>
      `
      <li>

      <div class="card-header">
        <input type="number" id="${exo.pic} min="1" max="10" value=${exo.min}>
        <span>min</span>
        </div>
        <img src="./img/${exo.pic}.png" />

      </li>
```

```
    `
```

D- AJOUT DES DEUX BOUTONS DU BAS (Pour déplacer les cartes et le bouton pour supprimer une carte)

```
lobby: function () {

   let mapArray = exerciceArray
   .map(
     (exo)=>
     `
     <li>

     <div class="card-header">
       <input type="number" id="${exo.pic} min="1" max="10" value=${exo.min}>
       <span>min</span>
       </div>
       <img src="./img/${exo.pic}.png" />
       <i class="fas fa-arrow-alt-circle-left arrow" data-pic=${exo.pic}></i>
       <i class="fas fa-times-circle deleteBtn" data-pic=${exo.pic}></i>

     </li>

     `
```

Maintenant qu'on a fini de créer la logique de nos cartes, maintenant on doit pouvoir les manipuler(Ajouter des évènements)

E- Création de la fonction qui va nous permettre de gérer (récupérer) le nombre de minutes choisi par l'utilisateur

a- Création et implémentation de la fonction handleEventMinutes() dans l'objet utils{}

```
handleEventMinutes: function(){
   document.querySelectorAll('input[type="number"]').
   forEach((input)=>{
     input.addEventListener("input", (event)=>{
       console.log(event);
     })
   })
 }
```

NB : Le console.log() que nous avons mis ici nous permet de visualiser (Pour le moment) le résultat de notre fonction

- **Nous allons appeler cette fonction dans le lobby () après le pageContent() (Pour s'assurer que toutes les cartes soient chargées avant de commencer à les utiliser**

- **Nous pouvons maintenant poser une condition dans la fonction pour vérifier s'il va trouver l'élément sur lequel on agit et afficher un message dans la console**

```
handleEventMinutes: function(){
    document.querySelectorAll('input[type="number"]').forEach((input)=>{
      input.addEventListener("input", (e)=>{
        exerciceArray.map((exo)=>{
          console.log("Test");
          if (exo.pic == e.target.id) {
            console.log("Yes");
          }
        })
      })
    })
  }
```

```
handleEventMinutes: function(){
   document.querySelectorAll('input[type="number"]').forEach((input)=>{
     input.addEventListener("input", (e)=>{
       exerciceArray.map((exo)=>{
         if (exo.pic == e.target.id) {
           exo.min = parseInt(e.target.value);
           console.log(exerciceArray)
         }
       })
     })
   })
 }
```

F- Déplacement des éléments

- **On se crée une fonction qui va nous permettre de gérer ce déplacement**
- **On doit pouvoir identifier la position de l'élément sur lequel on a cliqué**

```
handleEventArrow: function(){

   document.querySelectorAll(".arrow").forEach((arrow)=>{
     arrow.addEventListener('click', (e)=>{
       let position = 0; //Variable qui va nous permettre de trouver la positon de l
       exerciceArray.map((exo)=>{
         if (exo.pic == e.target.dataset.pic) {
           [exerciceArray[0], exerciceArray[1]] = [exerciceArray[1], exerciceArray[0]]
            console.log(exerciceArray);

         } else{
           position++;
           console.log(position);
```

```
        }
      })

    })
  })
```

On a réussi ici à intervertir les éléments aux positions 0 et 1 en utilisant la méthode :

```
[exerciceArray[0], exerciceArray[1]] = [exerciceArray[1], exerciceArray[0]]
```

- **Mais ce que nous voulons faire ici c'est de rendre cet échange dynamique**

```
handleEventArrow: function(){

  document.querySelectorAll(".arrow").forEach((arrow)=>{
    arrow.addEventListener('click', (e)=>{
      let position = 0; //Variable qui va nous permettre de trouver la positon de l
      exerciceArray.map((exo)=>{
        if (exo.pic == e.target.dataset.pic) {
          [exerciceArray[position], exerciceArray[position -1 ]] =
[exerciceArray[position -1], exerciceArray[position]]
           console.log(exerciceArray);

        } else{
          position++;
          console.log(position);

        }
      })

    })
  })
```

- **Maintenant on pose une condition pour s'assurer que la position ne soit jamais inférieure à 0 Et on rappelle la fonction page.lobby() pour reafficher les éléments**

```
handleEventArrow: function(){

  document.querySelectorAll(".arrow").forEach((arrow)=>{
    arrow.addEventListener('click', (e)=>{
      let position = 0; //Variable qui va nous permettre de trouver la positon de l
      exerciceArray.map((exo)=>{
        if (exo.pic == e.target.dataset.pic && position !== 0) {
          [exerciceArray[position], exerciceArray[position -1 ]] =
[exerciceArray[position -1], exerciceArray[position]]
           page.lobby();

        } else{
          position++;

        }
```

```
        })

      })
    })

  }
};
```

G- SUPPRESSION DES EXERCICES

Nous allons créer une fonction qui va nous permettre de gérer cela :

```
 deleteItems: function(){
    document.querySelectorAll(".deleteBtn").forEach((btn)=>{
      btn.addEventListener("click", (event)=>{
        let newArr = [];
        exerciceArray.map((exo)=>{
          if(exo.pic != event.target.dataset.pic){
            newArr.push(exo);
          }
        });
        exerciceArray = newArr;
console.log(exerciceArray);

      })
    })
  }
```

```
 deleteItems: function(){
    document.querySelectorAll(".deleteBtn").forEach((btn)=>{
      btn.addEventListener("click", (event)=>{
        let newArr = [];
        exerciceArray.map((exo)=>{
          if(exo.pic != event.target.dataset.pic){
            newArr.push(exo);
          }
        });
        exerciceArray = newArr;
        page.lobby();

      })
    })
  }
};
```

H- REBOOT (Recharger tous les éléments)