Les Objets

=> Objects are used to represent real-world entities or abstract concepts in code.

- Un objet est un conteneur pour des valeurs sous la forme de **propriétés** et des fonctionnalités sour la forme de **méthodes**

- Exemples

```
1  const h1 = document.getElementById("header");
2  console.log(h1.innerHTML);
3  h1.innerHTML = "Javascript is awesome";
```

  - **document**, **console** et **h1** sont des objets
  - **getElementById** et **log** sont des méthodes
  - **innerHTML** est une propriété (ici de l'objet **h1**)

Example : const h1 = document.getElementById("header");

document is an object representing the DOM of the HTML document,
getElementById is a method of the document object used to retrieve an element from the document by its ID, and
h1 becomes an object representing the HTML element with the ID "header."

Les Objets

- Possèdent des fonctions applicables appelées des **méthodes**. Les méthodes peuvent ou non retourner des valeurs;
- Possèdent des valeurs ou données, appelées des **propriétés**;
- Le nom d'une propriété est une **clé**;
- Le contenu d'une propriété est une **valeur**;

Rappel : les Objets littéraux

- Peuvent être utilisé pour stocker des données
- Peuvent être utilisé pour stocker l'état d'un objet
- Les clés sont des chaînes de caractères

```
1  let person = {
2      name : Lauren,
3      student : true,
4  }
5  person.name;
6  person['student'];
```


Ajouter des méthodes

- Exemple : fonction permettant de tirer un dé aléatoirement

```
1  function diceRoll(){
2      let sides = 6;
3      let rndNumber = Math.floor(Math.random() * sides) + 1;
4      console.log(rndNumber);
5  }
```

=> the roll method simulates rolling a six-sided die.

Ajouter des méthodes

- Exemple : fonction permettant de tirer un dé aléatoirement (randomly)

```
1  function diceRoll(){
2      let sides = 6;
3      let rndNumber = Math.floor(Math.random() * sides) + 1;
4      console.log(rndNumber);
5  }
```

- Encapsuler la fonction dans un objet permet de la transformer en méthode

```
1  let dice = {
2      roll: function (){
3          let sides = 6;
4          let rndNumber = Math.floor(Math.random()*sides)+1;
5          console.log(rndNumber);
6      }
7  }
```

Aller un peu plus loin

- Comment passer le nombre de faces comme une propriété du dé et non une variable de la méthode roll?

```
1  let dice = {
2      sides: 6,
3      roll: function (){
4          let rndNumber = Math.floor(Math.random()*dice.sides
               )+1;
5          console.log(rndNumber);
6      }
7  }
```

=> How to pass the number of faces as a property of the die and not a variable of the roll method?

=> This approach allows for more flexibility, as the number of sides can be easily changed without modifying the method itself.

Aller un peu plus loin

- Comment passer le nombre de faces comme une propriété du dé et non une variable de la méthode roll?

```
1  let dice = {
2      sides: 6,
3      roll: function (){
4          let rndNumber = Math.floor(Math.random()*dice.sides
               )+1;
5          console.log(rndNumber);
6      }
7  }
```

- Le mot clé **this**

```
1  let dice = {
2      sides: 6,
3      roll: function (){
4          let rndNumber = Math.floor(Math.random()*this.sides
               )+1;
5          console.log(rndNumber);
6      }
7  }
```

=> directly references the sides property of the dice object using this.sides.
=> This enhances the flexibility and reusability of the code.

Les Constructeurs

Constructors are used to create multiple objects of the same type with shared methods and properties.

- Limite des objets littéraux : déclarer plusieurs objets ayant les même méthodes
- Les constructeurs
  - Décrire comment un objet devrait être créer
  - Créer des objets similaires
  - Chaque objet crée est une instance de ce type d'objet

```
1  function Dice(sides){
2      this.sides = sides;
3      this.roll = function(){
4          let rndNumber = Math.floor(Math.random()*this.sides
               )+1;
5          return rndNumber;
6      }
7  }
8  let dice1 = new Dice(6);
9  let dice4 = new Dice(4);
```

=> to manage multiple instances with common characteristics.

Méthodes and Prototypes

- Que renvoie la ligne de code suivante ?

```
1  console.log(dice1.roll === dice4.roll);
```

Méthodes and Prototypes

- Que renvoie la ligne de code suivante ?

```
1  console.log(dice1.roll === dice4.roll);
```

→ **false**

- **Solution** : les prototypes !

Concept of prototypes in JS: it allows sharing methods among instances of objects efficiently

```
1  function Dice(sides) {
2      this.sides = sides;}
3
4  Dice.prototype.roll = function(){
5      let rndNumber = Math.floor(Math.random()*this.sides)+1;
6      return rndNumber;
7  }
```

- Chaque nouvel objet créé hérite des prototype du constructeur

=> Each new object created inherits the prototype from the constructor

Qu’est ce qu’un prototype

- Un prototype est une liste de propriétés attachée à un constructeur
- Toutes les fonctions déclarées en Javascript possèdent nativement un prototype
- L’ajout d’une propriété sur le prototype du constructeur devient disponible sur l’ensemble des instances

Notion d’héritage et chaîne de prototype

Concept of inheritance and prototype chain

- MDN : Héritage et chaînes de prototype

https://developer.mozilla.org/fr/docs/Web/JavaScript/Inheritance_and_the_prototype_chain