

# L'objet String

```
var unitPrice = 45;  
var qt = 3;  
  
console.log(` ${qt} produits à ${unitPrice}  
pièce font ${qt * unitPrice}`);
```

plus besoin de fermer les guillemets et d'utiliser  
l'opérateur de concaténation

Il est même possible d'évaluer des expressions  
et de faire des calculs

Fonction	Description
str.length	nombre de caractères dans la chaîne str
str.charAt(pos)	retourne le caractère à l'index pos. Sur les navigateurs récents on peut également utiliser str[pos]
str.charCodeAt(pos)	retourne le code du caractère à l'index pos
str.fromCharCode(code)	retourne le caractère correspondant au code passé en argument
str.trim()	retourne la chaîne str sans les espaces éventuels aux extrémités
str.toLowerCase()	convertit la chaîne str en minuscule
str.toUpperCase()	convertit la chaîne str en majuscule
str.repeat(n)	rèpte la chaîne str n fois

# Javascript Extraction de caractères

Fonction	Description
<code>str.slice(start,end)</code> <code>str.substring(start, end)</code>	retourne une sous chaîne de str depuis l'index <code>start</code> jusqu'à l'index <code>end</code> (non inclus)
<code>str.substr(start, length)</code>	retourne <code>length</code> caractères de str depuis l'index <code>start</code>
<code>str.replace(search, value)</code>	remplace la première occurrence de <code>search</code> au sein de str par la chaîne <code>value</code>
<code>str.replaceAll(search, value)</code>	remplace toutes les occurrences de <code>search</code> au sein de str par la chaîne <code>value</code>

# Javascript Majuscule sur la première lettre d'un mot

```
var word = "hello";  
  
var capitalized = word[0].toUpperCase() + word.substr(1).toLowerCase();  
  
console.log(capitalized);
```

En javascript les chaînes de caractères ont des tableaux de caractères, il est même possible d'accéder à chaque caractère comme on le fait avec un tableau

Cependant on ne peut modifier une chaîne de caractère de cette façon, les caractères sont en lecture seule

```
// ne fonctionne pas affiche toujours hello
word[1] = "o";
word[4] = "a"
console.log(word);
```

```
word = word.replace("o", "a").replace("e", "o");  
console.log(word);  
  
var phrase = "Ce chien est un gentil chien";  
  
// Ne remplace que la première occurrence  
console.log(phrase.replace("chien", "chat"));  
  
// Remplace toutes les occurrences  
console.log(phrase.replaceAll("chien", "chat"));
```

# Javascript Boucle sur les chaînes de caractères

```
// Boucle sur les indices
for (let index in word){
    console.log(index, " : ", word[index]);
}

// Boucle sur les caractères
for ( let letter of word){
    console.log(letter);
}
```

Eliminer les lettres consécutives  
en doublon dans une chaîne de caractère  
hello deviendra helo

Fonction	Description
<code>str.indexOf(search)</code>	retourne la position de la première occurrence de la chaîne search au sein de str
<code>str.lastIndexOf(search)</code>	retourne la position de la dernière occurrence de la chaîne search au sein de str
<code>str.includes(search)</code>	retourne true si la chaîne search est inclus dans str
<code>str.startsWith(search)</code>	retourne true si la chaîne str commence par la chaîne search
<code>str.endsWith(search)</code>	retourne true si la chaîne str se termine par la chaîne search

Fonction	Description
<code>str.split(separator)</code>	découpe la chaîne str à chaque occurrence de <b>separator</b> et retourne un tableau de chaînes de caractères
<code>array.join(separator)</code>	contraire de split, constitue une chaîne de caractère en concaténant toutes les valeurs d'un tableau et en ajoutant la chaîne <b>separator</b> devant chaque élément

Ecrire un programme qui met en gras toutes les occurrences d'une chaîne recherchée

Ecrire une chaîne de caractère contenant des gros mots  
Ecrire un programme qui remplace  
ces gros mots par des étoiles

Vous avez un chemin et un nom de fichier,  
par exemple : /var/html/fichier.site.html

Extraire de cette chaîne le nom du fichier sans  
l'extension ni le chemin (juste fichier.site)

# L'objet Date

Une date est définie sous la forme d'un entier représentant le nombre de millisecondes écoulées depuis le premier janvier 1970 à minuit

```
//Date et heure en cours
var d1 = new Date();
//Nombre de millisecondes
var d2 = new Date(86400000);

//année, mois, jour (janvier = 0)
var d3 = new Date(2016,11,5);

//ajout des heures, minutes, secondes, millisecondes
var d4 = new Date(2016,11,5, 15,5,20,0);

var d5 = new Date("October 13, 2016 12:15:00");
```

d.toDateString();	Mon Dec 05 2016
d.toLocaleDateString();	05/12/2016
d.toLocaleString();	05/12/2016 à 15:05:20
d.toLocaleTimeString();	15:05:20
d.toISOString();	2016-12-05T14:05:20.000Z
d.toJSON();	2016-12-05T14:05:20.000Z
d.toUTCString();	Mon, 05 Dec 2016 14:05:20 GMT
d.getTimeString();	15:05:20 GMT+0100 (CET)

d.getFullYear();	L'année sur 4 chiffres
d.getMonth();	Le mois de 0 à 11
d.getDate();	Le jour du mois de 1 à 31
d.getDay();	Le jour de la semaine de 0 à 6 (0 = dimanche)
d.getHours();	L'heure de 0 à 59
d.getMinutes();	La minute de 0 à 59
d.getMilliseconds();	Les millisecondes de 0 à 999
d.getTime()	Nombre de millisecondes depuis le 01/01/1970 minuit
d.getTimezoneOffset();	Décalage en minutes entre la date locale et la date UTC

d.setFullYear(year);	L'année sur 4 chiffres
d.setMonth(month);	Le mois de 0 à 11
d.setDate(date);	Le jour du mois de 1 à 31
d.setDay(day);	Le jour de la semaine de 0 à 6 0 = dimanche
d.setHours(hours);	L'heure de 0 à 59
d.setMinutes(min);	La minute de 0 à 59
d.setMilliseconds(ms);	Les millisecondes de 0 à 999

```
var d = new Date(2016, 11, 10);

d.setDate(d.getDate() + 25);

//affiche 04/01/2017
console.log(d.toLocaleDateString());
```

Calculer le nombre de secondes écoulées entre le début et la fin d'un script

Ajouter 6 mois et 5 jours à la date du jour

Afficher la date du premier lundi du mois prochain

Calculer le nombre de mois écoulés entre deux dates (en considérant qu'un mois commencé est écoulé)

# Les timers

- ✓ Exécuter un code au terme d'un délai ou de façon répétée avec une temporisation
- ✓ Une fonction de rappel (callback) est déclarée puis exécutée par le timer

```
let time = 0;

function startChrono(){
    time++;
    console.log(time);
}

function stopChrono(){
    // clearInterval à besoin d'un argument
    // pour savoir que intervalle il doit supprimer
    clearInterval(interval);
}

// Lance le chrono qui s'exécute toutes les secondes
let interval = setInterval(startChrono, 1000);

// Arrête le chrono au bout de 15 secondes
let timer = setTimeout(stopChrono, 15* 1000);
```

Créer une horloge qui affiche l'heure  
dans la page web ou la console

- ✓ Masquer une alerte au bout d'un délai
- ✓ Faire une animation (déplacement dans l'espace et le temps)
- ✓ Tester quelque chose régulièrement
  - ✓ Jeu
  - ✓ chat
  - ✓ compte à rebours

# Les fonctions

```
const sophie = {  
    firstName: "Sophie",  
    lastName: "Martin",  
    age: 34,  
    getFullName: function(){  
        return `${this.firstName} ${this.lastName.toUpperCase()}`;  
    }  
}  
  
console.log(sophie.getFullName());
```

Ajouter une fonction canDrinkAlcohol qui retourne vrai si la personne à 21 ans ou plus

# Javascript What is this ?

```
const sophie = {  
    firstName: "Sophie",  
    lastName: "Martin",  
    age: 34,  
    getFullName: function(){  
        // affiche l'objet sophie  
        console.log(this);  
        return `${this.firstName} ${this.lastName.toUpperCase()}`;  
    }  
}  
// Affiche Window  
console.log(this);  
console.log(sophie.getFullName());
```

This donne le contexte, l'objet sur lequel est exécuté la fonction. Par défaut c'est Window

```
//Syntaxe basique
const add = (n1, n2) => {
    console.log(n1 + n2);
};

//Si une seule instruction on peut omettre les accolades
const add2 = (n1, n2) => console.log(n1+ n2);

//Si un seul argument on peut omettre les parenthèses
const sayHello = name => console.log("Hello " + name);

//return implicite pour les fonction qui n'ont qu'une seule ligne
const square = x => x * x;
```

# Javascript This n'est plus contextuel

```
const point = {
  x: 0,
  y: 0,
  setCoordinates: function(x,y){
    // affiche point
    console.log(this);
    this.x = x;
    this.y = y;
  },
  calcDistance: (x,y) => {
    // affiche window
    console.log(this);
    return Math.sqrt(
      (this.x - x)**2 +
      (this.y - y)**2
    );
}
};
```

Avec les arrow function, this fait toujours référence au contexte global.

Ici c'est gênant, mais il est des cas où c'est intéressant notamment lorsqu'une fonction callback doit conserver son contexte de déclaration (très utile dans React)

## L'objet Array

```
const fruits = ["poires", "oranges", "poires"];  
const search = "poires";  
  
//Retourne l'indice de la première occurrence  
//de la valeur recherchée  
//ou -1 si la valeur est absente  
console.log(fruits.indexOf(search));  
  
//indice de la dernière occurrence  
console.log(fruits.lastIndexOf(search));
```

```
const fruits = ["figues", "oranges", "poires", "mangues"];
var subset;
//Extrait de la position 1 à la position 3 (non inclus)
//["oranges", "poires"]
subset = fruits.slice(1,3);

//Extrait de la position 1 à la fin du tableau
//["oranges", "poires", "mangues"]
subset = fruits.slice(1);

//Extrait les deux derniers éléments
//["poires", "mangues"]
subset = fruits.slice(-2);
```

```
const fruits = ["figues", "oranges", "poires", "mangues"];

//supprime 2 éléments à partir de la position 1
//et retourne un tableau des éléments supprimés
//["oranges", "poires"]
const deleted = fruits.splice(1, 2);
```

```
const fruits = ["figues", "oranges", "poires", "mangues"];  
  
//insère 2 éléments à partir de la position 1  
fruits.splice(1, 0, "kiwis", "citrons");  
  
//supprime 1 élément à partir de la position 2  
//et en insère 1  
fruits.splice(2, 1, "coings");
```

```
const fruits = ["figues", "oranges", "poires", "mangues"];
const legumes = ["courges", "carottes", "aubergine"];
const legumineuses = ["soja", "pois chiche", "fèves"];

// Concaténation des tableaux
const aliments = fruits.concat(legumes, legumineuses);

// Autre syntaxe
const aliments2 = [...fruits, ...legumes, ...legumineuses];
```

```
// Demander la saisie d'une phrases
const phrase = prompt("Votre phrase");

// découper la phrase en tableau
// avec une case par mot
const words = phrase.split(" ");

// Mélanger les mots dans le tableau
words.sort( (item) => Math.random() - .5);

// Convertir le tableau en chaîne de caractère
const yodaSpeaking = words.join(" ");

console.log(yodaSpeaking);
```

# Les fonctions avancées de l'objet Array

# Javascript Trier un tableau

```
const fruits = ["pommes", "poires", "bananes"];
//["bananes", "poires", "pommes"]
fruits.sort();

//La fonction sort traite les éléments
//comme des chaînes de caractères
const list = [50, 111, 9, 200, 38];
//[111, 200, 38, 50, 9]
list.sort();
```

# Javascript Tri amélioré

```
const list = [50, 111, 9, 200, 38];
// [200, 111, 38, 50, 9]
list.sort(function(a, b){
    return b-a;
});
```

La fonction anonyme de comparaison  
doit retourner un entier  
positif, négatif ou égal à zéro

```
const list = [50, 111, 9, 200, 38];
// avec une arrow function
list.sort((a, b) => b-a);
```

```
const persons = [
    {nom: 'toto', age: 50},
    {nom: 'titi', age: 12},
    {nom: 'tata', age: 28}
];
persons.sort( (a, b) => b.age - a.age);
```

Le tri porte sur les propriétés des  
objets dans le tableau ordinal

# Javascript Mélangé un tableau

```
const list = [1,2,3,4,5];
persons.sort( (a, b) => Math.random() - .5);

console.log(list);
```

```
var persons = [
    {nom: 'toto', age: 50},
    {nom: 'titi', age: 12},
    {nom: 'tata', age: 28},
    {nom: 'tutu', age: 17}
];

// Retourne la première personne qui répond true
// au test dans la fonction de callback
const p = persons.find(person => person.age < 20);

console.log(p);
```

1	currentValue	Obligatoire : la valeur de l'élément en cours
2	index	Facultatif : l'index de l'élément en cours
3	array	Facultatif : le tableau source

valable pour les fonctions  
find, findIndex, forEach, every, some, filter et map

# Javascript Tester ce code

```
var persons = [
    {nom: 'toto', age: 50},
    {nom: 'titi', age: 12},
    {nom: 'tata', age: 28},
    {nom: 'tutu', age: 17}
];

const p = persons.find( (person, index, list) => {
    console.log(index);
    console.log(person);
    console.log(list);
    return person.age < 20
} );

console.log(p);
```

<code>array.find(fn)</code> <code>array.findIndex(fn)</code>	Retourne la première valeur passant un test définit dans la fonction fn. <code>findIndex</code> retourne l'index au lieu de la valeur
<code>array.every(fn)</code>	Vérifie si tous les éléments du tableau passent le test définit dans la fonction fn
<code>array.some(fn)</code>	Vérifie si au moins un élément du tableau passe le test définit dans la fonction fn

# Javascript Exemple

```
var persons = [  
    {nom: 'toto', age: 50},  
    {nom: 'titi', age: 12},  
    {nom: 'tata', age: 28},  
    {nom: 'tutu', age: 17}  
];  
  
// Retourne vrai si toutes les personnes ont plus de 17 ans  
const allAdults = persons.every(person => person.age >17);  
  
console.log(allAdults);
```

Rechercher s'il y a des teenagers dans la liste  
des personnes

# Javascript Rechercher un index

```
var persons = [
    {nom: 'toto', age: 28},
    {nom: 'titi', age: 13},
    {nom: 'tata', age: 5},
    {nom: 'tutu', age: 17}
];

// Trouve l'index du premier teenager
const index = persons.findIndex(person => {
    return person.age < 20 && person.age > 12;
});

// supprime ce teenager
persons.splice(index, 1);

console.log(persons);
```

array.filter(fn)	Retourne un nouveau tableau contenant les éléments du tableau original qui passent le test défini dans la fonction fn
array.forEach(fn)	Appelle la fonction fn pour chaque élément du tableau
array.map(fn)	Retourne un tableau résultant de l'exécution de la fonction fn sur chaque élément du tableau
array.reduce(fn)	Réduit un tableau à une seule valeur (agrégation). Parcourt le tableau du premier indice au dernier.
array.reduceRight(fn)	Réduit un tableau à une seule valeur (agrégation). Parcourt le tableau du dernier indice au premier.

# Javascript Filtrer des données

```
var persons = [
    {nom: 'toto', age: 28},
    {nom: 'titi', age: 13},
    {nom: 'tata', age: 5},
    {nom: 'tutu', age: 17}
];

// Uniquement les teenagers
persons = persons.filter( person => {
    return person.age < 20 && person.age > 12;
});

console.log(persons);
```

# Javascript Foreach, une autre façon de boucler

```
const counter = {  
    odd: 0,  
    even: 0  
};  
  
const list = [3, 2, 8, 7, 5, 0];  
  
// compter les nombres pairs et impairs  
list.forEach( (n) => {  
    if(n% 2 == 0){  
        counter.even ++;  
    } else {  
        counter.odd ++;  
    }  
})  
  
console.log(counter);
```

```
const list = [ "5", "7", "aaa", "4.9" ];

newList = list.map( (item) => {
    // tente de convertir en entier
    // 0 si impossible
    return parseInt(item) || 0;
} );

console.log(newList);
```

Quelles sont les différences  
entre map et forEach ?

# Javascript Reduce

```
const notes = [5, 20, 7, 18, 13, 16, 14, 9, 20];

const total = notes.reduce(
    // La fonction de callback admet deux arguments
    // l'accumulateur qui stock le total
    // la valeur de chaque itération
    function (accumulator, value) {
        return accumulator + value;
    }
);

// Équivalent avec une boucle classique
let total2 = 0;
for(let n of notes){
    total2 += n;
}

console.log(total);
console.log(total2);
```

Réduire une liste de données à une seule valeur

## reduce

Itérations	1	2	3	4	5
Eléments	5	4	2	3	1
Accumulateur	0	5	9	11	14
				15	

## reduceRight

Itérations	5	4	3	2	1
Eléments	5	4	2	3	1
Accumulateur	10	6	4	1	0
				15	

```
const list = [
  5,
  4,
  2,
  3,
  1
];
```

# Javascript Les arguments de la fonction reduce

1	accumulator	Obligatoire : le résultat de la précédente itération
2	current item	Obligatoire : l'élément en cours
3	index	Facultatif : l'index de l'élément en cours
4	array	Facultatif : le tableau source

# Javascript Chainage des méthodes

```
const students = [
  {
    name: "Patricia", grades: {
      maths: 12,
      physics: 10,
      english: 13
    }
  },
  {
    name: "Paul", grades: {
      maths: 7,
      physics: 8,
      english: 15
    }
  }
];

const mathAverage = students
  .map(function (item) {
    return item.grades.maths;
  })
  .reduce(function (acc, val) {
    return acc + val;
  }) / students.length;

console.log(mathAverage);
```

Ici map transforme le tableau d'objet en un tableau de valeurs numériques sur lequel on exécute la fonction reduce

```
const articles = [
    {title: "Article1", author: "Paul", rating: 3},
    {title: "Article1", author: "Sophie", rating: 5},
    {title: "Article1", author: "Paul", rating: 4},
    {title: "Article1", author: "Jean", rating: 2},
    {title: "Article1", author: "Paul", rating: 2},
    {title: "Article1", author: "Valérie", rating: 5},
];
```

Calculer la note moyenne  
des articles de Paul

# La gestion des erreurs

- ✓ Syntaxe
- ✓ Exécution (runtime)
- ✓ Logique

## source

Mon code n'est pas syntaxiquement correct

- ✓ J'ai oublié de fermer une parenthèse, une accolade
- ✓ J'utilise une fonction inconnue de mon interpréteur

# Javascript Erreurs de syntaxe solutions

- ✓ Être plus attentif
- ✓ Utiliser un bon IDE

# Javascript Erreurs d'exécution

## source

La valeur d'une variable à un instant t provoque une erreur

- ✓ La saisie de l'utilisateur est incorrecte
- ✓ Le type d'une variable a changé

# Javascript Erreurs d'exécution solutions

- ✓ Tester les cas marginaux
- ✓ Valider les données avant de les utiliser (saisie utilisateur, arguments fonctions, autres données externes)
- ✓ Éviter le transtypage

Mon code est correct mais ne fais pas ce que je veux

- ✓ Mon algorithme n'est pas approprié
- ✓ J'ai mal compris le problème

# Javascript Erreurs de logique solutions

- ✓ Spécifier le code (comportement attendu)
- ✓ Tester le code avec des valeurs d'entrées et de sortie connues

```
try {  
  
    instructions;  
  
} catch(error){  
  
    gestion de l'erreur  
}
```

Capturer des erreurs exceptionnelles (erreur d'exécution) lorsque l'on ne peut être certain de la validité du code car :

- ✓ Le code dépend de données externes
- ✓ Le code fait appel à des librairies ou fonctions externes

```
var jsonData = "test";
var data;
try
{
    data = JSON.parse(jsonData);
}
catch(error)
{
    console.log("Exception");
    console.log(error.name);
    console.log(error.message);
}
```

- ✓ Signaler une erreur sans la traiter immédiatement
- ✓ Très utile pour créer des bibliothèques de fonctions

# Javascript Exemple

```
function calcul(n, divisor){  
    if(! divisor){  
        throw {  
            message:"Division pas zéro interdite",  
            name:"InvalidArgumentException"};  
    }  
    return n / divisor;  
}
```

```
try  
{  
    calcul(5,0)  
}  
catch(error)  
{  
    console.log(error);  
}
```

# LocalStorage

- ✓ Conserver des données lorsque l'utilisateur ferme son navigateur
- ✓ On ne veut ou ne peut pas utiliser de base de données

- ✓ Un espace de stockage dans le navigateur de l'utilisateur
- ✓ On associe des clefs à des valeurs
- ✓ On ne stocke que des chaînes de caractères
- ✓ Limité à 20 Mo par domaine (site)

# Javascript Exemple simple

```
const storage = window.localStorage;

// Récupération depuis le localStorage
let userName = storage.getItem("userName");

// Lorsque la clef n'existe pas
// getItem retourne la chaîne de caractère
// "null"
if(! userName || userName === "null"){
    userName = prompt("Votre nom");
    // Enregistrement dans le localStorage
    storage.setItem("userName", userName);
}

alert("Bonjour " + userName);
```

```
const storage = window.localStorage;

// Récupération depuis le localStorage
// On transforme la chaîne de caractère en objet
// avec JSON.parse
// si le résultat est null on affecte un tableau vide
let data = JSON.parse(storage.getItem("list")) || [];

// Ajout d'un élément au tableau
data.push("item");

// Sauvegarde dans le storage,
// il faut convertir le tableau en chaîne de caractère
// avec JSON.stringify
storage.setItem("list", JSON.stringify(data));

console.log(data);
```

# Les classes

- ✓ Nouvelle syntaxe introduite avec ES6
- ✓ Syntaxiquement similaire à Java

# Javascript Un exemple

La classe

Une méthode

Instanciation

Le constructeur

```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        console.log(`Bonjour, je m'appelle ${this.name}.`);  
    }  
  
    const charlie = new Person("Charlie", 28);  
    charlie.greet();
```

Exécution d'une méthode  
sur l'instance

- ✓ Encapsulation
  - ✓ Les données et les traitements sont regroupés dans un objet
- ✓ Instanciation
  - ✓ La classe est un modèle qui permet de fabriquer des objets
- ✓ Modélisation
  - ✓ La classe est une représentation d'une partie de la réalité

Modéliser un panier de commerce électronique

Lister les propriétés et les méthodes qui vous paraissent judicieuses

```
sophie = {  
    name: 'Sophie',  
    grades: [12, 8, 9, 15]  
}  
  
function addGrade(grade, student) {  
    student.grades.push(grade);  
}  
  
function calculateAverage(student) {  
    return student.grades.reduce(  
        (sum, grade) => { sum += grade; }, 0  
    ) / student.grades.length;  
}  
  
function hasPassed(student) {  
    return calculateAverage(student) >= 10;  
}  
  
addGrade(16, sophie);  
addGrade(2, sophie);  
  
console.log(`${sophie.name} a son diplôme :`, hasPassed(sophie));
```

Reprendre le code suivant  
en utilisant une classe  
Student

Faire une variante en  
utilisant les méthodes des  
objets littéraux

Appel du constructeur  
du parent

Classe parente

```
class Employee extends Person {
    constructor(name, age, jobTitle) {
        super(name, age);
        this.jobTitle = jobTitle;
    }

    work() {
        console.log(`#${this.name} is working as a ${this.jobTitle}.`);
    }
}

const employee1 = new Employee('Bob', 25, 'Developer');
employee1.greet();
employee1.work();
```

Reprendre la classe  
Student et hériter  
de la classe Person

- ✓ L'héritage est utile quand on peut répondre positivement à la question "est un", c'est une spécialisation
- ✓ Un employé est une personne
- ✓ Javascript ne supporte pas l'héritage multiple
- ✓ Quand la question est "possède un" alors on utilise la composition

# Javascript Exemple de composition

```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        console.log(`bonjour je m'appelle ${this.name}`);  
    }  
}  
  
class Student extends Person {  
    constructor(name, age) {  
        super(name, age);  
        this.grades = [];  
    }  
  
    addGrade(grade){  
        this.grades.push(grade);  
    }  
  
    calculateAverage() {  
        return this.grades.reduce(  
            (sum, grade) => { sum += grade; }, 0  
        ) / this.grades.length;  
    }  
  
    hasPassed() {  
        return this.calculateAverage() >= 10;  
    }  
}
```

```
class Classroom {  
    constructor(teacher) {  
        this.students = [];  
        this.teacher = teacher  
    }  
  
    addStudent(student) {  
        this.students.push(student);  
    }  
  
    getClassResults(){  
        return this.students.map(  
            (student)=> {  
                return {  
                    name : student.name,  
                    result: student.hasPassed()? 'pass' : 'fail'  
                }  
            }  
        );  
    }  
}  
  
const joeTeacher = new Person('Joe', 42);  
const classroom = new Classroom(joeTeacher);  
  
const sophie = new Student('Sophie', 20);  
sophie.addGrade(10);  
sophie.addGrade(12);  
classroom.addStudent(sophie);  
  
const martin = new Student('Martin', 20);  
sophie.addGrade(8);  
sophie.addGrade(9);  
classroom.addStudent(martin);  
  
console.log(classroom.getClassResults());
```

# Javascript Méthodes et propriétés privées

```
class BankAccount {  
    #balance;  
  
    constructor(balance) {  
        this.#balance = balance;  
    }  
  
    deposit(amount) {  
        this.#balance += amount;  
    }  
  
    getBalance() {  
        return this.#balance;  
    }  
}  
  
const account = new BankAccount(1000);  
account.deposit(500);  
console.log(account.getBalance());  
  
// Provoque une erreur  
console.log(account.#balance);
```

Reprendre la classe  
Classroom et rendre la  
propriété students privée