

# JavaScript

Hedi Rivas - 11/2022

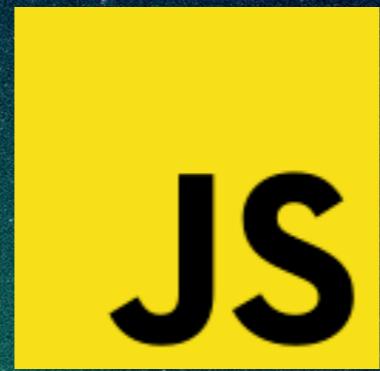
# Programme

Les Fondamentaux - Partie 1  
Les Fondamentaux - Partie 2  
JavaScript dans le navigateur - DOM & Events  
Data structures, Modern Operators and Strings  
Working With Arrays  
Numbers, Dates, Intl and Timers



# Historique

JavaScript est un langage de programmation créé en 1995.  
Une grande majorité des sites web l'utilisent,  
et la majorité des navigateurs web disposent d'un moteur  
JavaScript pour l'interpréter.



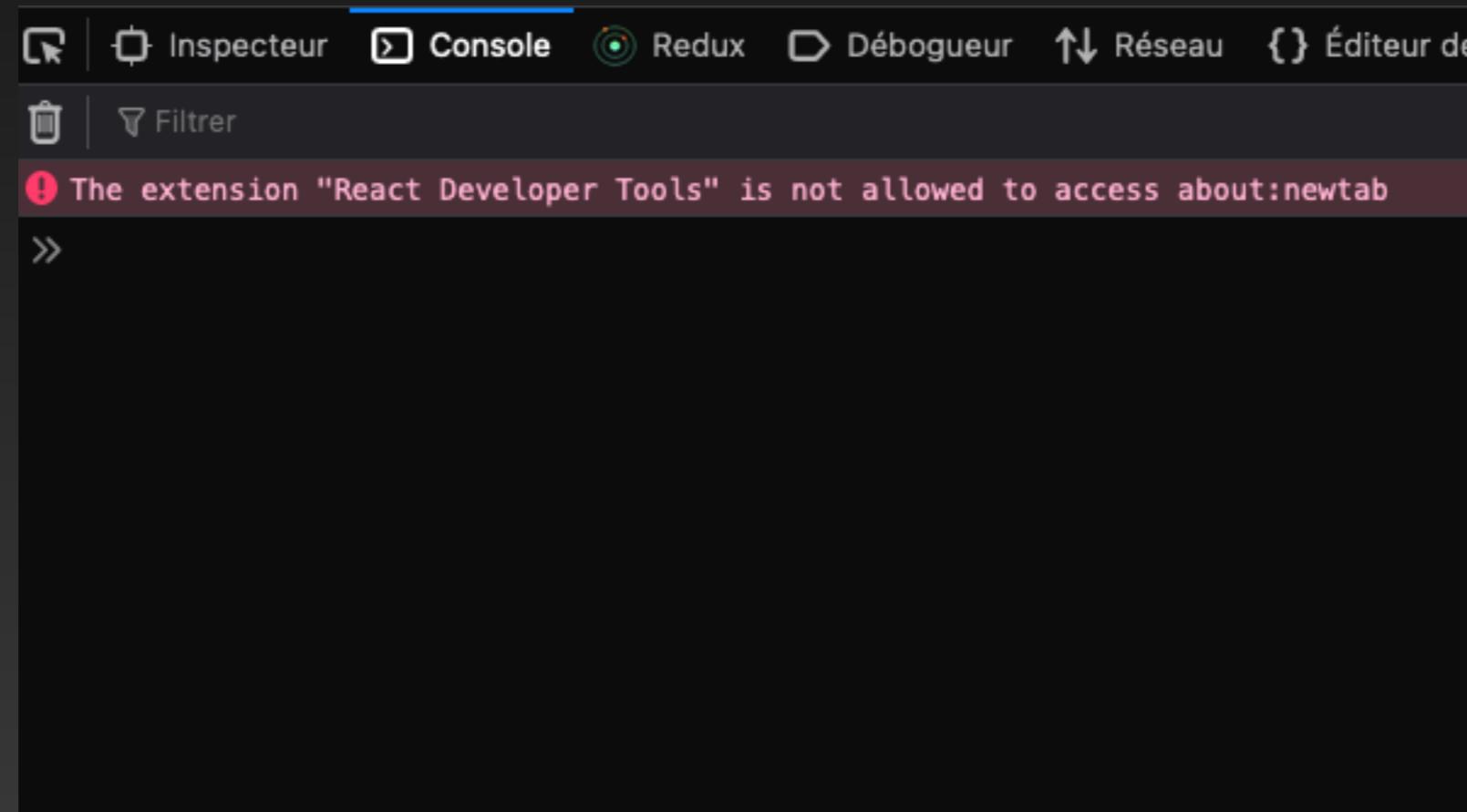
# Les fondamentaux

## Partie 1

# Les Fondamentaux - Partie 1

Hello world

Le premier endroit où nous allons écrire du JavaScript... la **console** du navigateur !



# Les Fondamentaux - Partie 1

## Lié un fichier JavaScript

Écrire du code dans la console du navigateur, c'est sympa mais on peut quand même faire mieux avec du JavaScript!

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>JavaScript Course</title>
  </head>
  <body>
    <script src="script.js"></script>
  </body>
</html>
```

# Les Fondamentaux - Partie 1

## Variables and values

```
//Variables and values

let js = 'amazing';
console.log(40 + 8 + 23 - 10); 61

console.log('Hedi'); Hedi
console.log(23); 23

let firstName1 = 'Hedi';

console.log(firstName1); Hedi
console.log(firstName1); Hedi
console.log(firstName1); Hedi

//Variable name conventions
let hedi_teacher = 'HT';
let $function = 27;

let person = 'hedi';
let PI = 3.1415;

let myFirstJob = 'Coder';
let myCurrentJob = 'Teacher';

let job1 = 'programmer';
let job2 = 'teacher';

console.log(myFirstJob); Coder
```

# Pratique



Variables and values



# Pratique



- \* Déclarez des variables nommées « country », « continent » et « population » et assignez les valeurs correspondantes à votre pays.
- \* Loggez les valeurs dans la console

# Pratique



```
let country = 'Portugal';
let continent = 'Europe';
let population = 10;
console.log(country);
console.log(continent);
console.log(population);
```

# Les Fondamentaux - Partie 1

## Data types

Il y a 7 types de données en JS :

- String
- Number
- Boolean
- Undefined
- Null
- BigInt
- Symbol

# Pratique



Data Types



# Pratique



- \* Déclarez une variable nommée « `isIsland` » et y ajouter une valeur boolean : `true` si votre pays est une île ou `false` si non. Déclarez également une variable « `language` » et la laisser vide pour le moment.
- \* Loggez les types des variables (avec `typeof`) précédemment créées dans la console.

# Pratique



```
let isIsland = false;
let language;
console.log(typeof isIsland);
console.log(typeof population);
console.log(typeof country);
console.log(typeof language);
```

# Les Fondamentaux - Partie 1

## Let, const et var

**let** permet de stocker et modifier une valeur.

```
let score = 0;  
score = 50;
```

**var** correspond à une "ancienne version" de **let**.

```
var score = 0;  
score = 50;
```

**const** permet de stocker une valeur et d'interdire une future modification.

```
const birthYear = 1980;  
birthYear = 1990;
```

# Pratique



Let, const et var



# Pratique



- ✳ Ajoutez votre langue à la variable « language »
- ✳ Réfléchissez à quelle variable peut-être modifiée en const (quelle valeur ne changera jamais ?)
- ✳ Essayez de changer la valeur de la variable const à présent, que remarquez-vous ?

# Pratique



```
language = 'portuguese';
const country = 'Portugal';
const continent = 'Europe';
const isIsland = false;
isIsland = true;
```

# Les Fondamentaux - Partie 1

## Basic operators

```
// Basic operators

const now = 2037;
const ageHedi = now - 1988;
const ageSarah = now - 2018;
console.log(ageHedi, ageSarah); 49 19

console.log(ageHedi * 2, ageHedi / 10, 2 ** 3); 98 4.9 8
// 2 ** 3 means 2 to the power of 3 = 2 * 2 * 2

const firstName = 'Hedi';
const lastName = 'Rivas';
console.log(firstName + ' ' + lastName); Hedi Rivas


// Assignment operators
let x = 10 + 5; // 15
x += 10; // x = x + 10 = 25
x *= 4; // x = x * 4 = 100
x++; // x = x + 1 = 101
x--; // x = x - 1 = 100
x--; // x = x - 1 = 99
console.log(x); 99


// Comparison operators
console.log(ageHedi > ageSarah); // >, <, >=, <= true
console.log(ageSarah >= 18); true

const isFullAge = ageSarah >= 18;

console.log(now - 1991 > now - 2018); true
```

# Pratique



Basic operators



# Pratique



- \* Si la population de votre pays était divisé en 2, combien resteraient-ils de personnes dans une moitié ?
- \* Augmentez la population de votre pays de 1 et loggez le résultat dans la console.
- \* La Finlande a une population de 6 millions de personnes. Est-ce que votre pays a une plus grande population ?
- \* La population moyenne d'un pays est 33 millions de personnes. Est-ce que votre pays a moins de personnes que la moyenne ?
- \* En partant des variables que vous avez précédemment créées, créez une nouvelle variable « description » qui contient une string avec le format : « Le Portugal est en Europe, et ses 11 millions d'habitants parlent le portugais. »

# Pratique



```
console.log(population / 2);
population++;
console.log(population);
console.log(population > 6);
console.log(population < 33);
const description1 =
  country +
  ' is in ' +
  continent +
  ', and its ' +
  population +
  ' million people speak ' +
  language;
console.log(description1);
```

# Les Fondamentaux - Partie 1

## Priorité entre les opérateurs

```
// Operator precedence

let a, b;
a = b = 25 - 10 - 5; // a = b = 10, a = 10
console.log(a, b); 10 10

const averageAge = (ageHedi + ageSarah) / 2;
console.log(ageHedi, ageSarah, averageAge); 49 19 34
```



# Part 1

# Challenge #1



[LIEN DU CHALLENGE](#) 

# Les Fondamentaux - Partie 1

## Template literals

```
//template literal
const firstName = 'Hedi';
const job = 'teacher';
const birthYear = 1988;

const hedi =
| "I'm " + firstName + ', a ' + (2021 - birthYear) + ' years old ' + job + '!';
console.log(hedi); I'm Hedi, a 33 years old teacher!

const hediNew = `I'm ${firstName}, a ${2021 - birthYear} years old ${job}!`;
console.log(hediNew); I'm Hedi, a 33 years old teacher!

console.log(`Just a regular string...`); Just a regular string...

console.log( String with multiple lines
| 'String with \n\
multiple \n\
lines'
);

console.log(`String String multiple lines
multiple
lines`);
```

# Pratique



Template literals



# Pratique



- ✿ Recréez la variable « description » en utilisant les template literals.

# Pratique

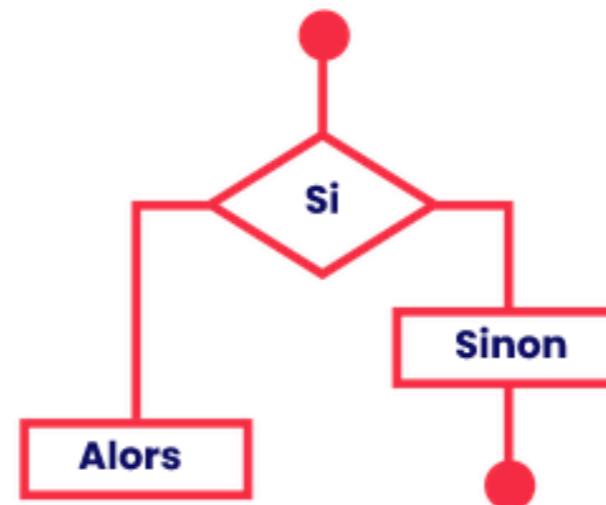


```
const description = `${country} is in ${continent}, and its  
${population} million people speak ${language}`;
```

# Les Fondamentaux - Partie 1

## If / else statements

Même si notre code est automatisé grâce aux boucles, l'exécution du code se fait linéairement, sans s'adapter à différents scénarios.  
Il est dépourvu de **logique** et n'est pas capable de **prendre des décisions**.



# Les Fondamentaux - Partie 1

## If / else statements

```
//if / else statements
const birthYear = 1991;
let century;

if (birthYear <= 2000) {
    century = 20;
} else {
    century = 21;
}
console.log(century); 20
```

# Pratique



If/else statements



# Pratique



- \* Si la population de votre pays est plus grande que 33 millions, logez dans la console une string comme celle-ci « La population du Portugal est plus grande que la moyenne ». Dans le cas contraire, logez une string comme « La population du Portugal est 22 millions en dessous de la moyenne» (Le nombre 22 est trouvé en faisant la différence entre la moyenne et la population du Portugal).
- \* Après avoir essayé votre condition, recommencez en changeant la valeur de la population, essayé avec 13 et 130. Voyez les changements qui se passent et remettez la valeur de base de la population.

# Pratique



```
if (population > 33) {  
    console.log(`${country}'s population is above average`);  
} else {  
    console.log(  
        `${country}'s population is ${33 - population} million  
        below average`,  
    );  
}
```



# Part 1 Challenge #2



[LIEN DU CHALLENGE](#) 

# Les Fondamentaux - Partie 1

## Type conversion

```
//type conversion

const inputYear = '1991';
console.log(Number(inputYear), inputYear); 1991 '1991'
console.log(Number(inputYear) + 18); 2009

console.log(Number('Hedi')); NaN
console.log(typeof NaN); number

console.log(String(23), 23); 23 23
```

# Les Fondamentaux - Partie 1

## Type coercion

```
//type coercion

console.log('I am ' + 30 + ' years old.');// I am 30 years old.
console.log("23" - "10" - 3); // 10
console.log(('23' / "2")); // 11.5
console.log("10" * "2"); // 20
```

```
// Essayez de deviner la valeur de :

let n = '1' + 1;
n = n - 1;
```

# Pratique



Type conversion et coercion



# Pratique



- \* Essayez de prédire le résultat de ces 5 opérations sans les exécuter dans votre code :
  - « 9 » - « 5 »;
  - « 19 » - « 13 » + « 17 »;
  - « 19 » - « 13 » + 17;
  - « 123 » < 57;
  - 5 + 6 + « 4 » + 9 - 4 - 2;
- \* Exécutez maintenant les opérations et loggez les dans la console.

# Pratique



```
console.log('9' - '5'); // -> 4
console.log('19' - '13' + '17'); // -> '617'
console.log('19' - '13' + 17); // -> 23
console.log('123' < 57); // -> false
console.log(5 + 6 + '4' + 9 - 4 - 2); // -> 1143
```

# Les Fondamentaux - Partie 1

## Truthy and False values

```
// il existe 5 valeurs falsy : 0, "", undefined, null, NaN

console.log(Boolean(0));  false
console.log(Boolean(undefined));  false
console.log(Boolean('Hedi'));  true
console.log(Boolean({}));  true

const money = 0;

if (money) {
    console.log("Don't spend it all");
} else {
    console.log("You should get a job!");  You should get a job!
}

let height;
if (height) {
    console.log("Yay! Height is defined");
} else {
    console.log("Height is UNDEFINED");  Height is UNDEFINED
}
```

# Les Fondamentaux - Partie 1

## Equality operators == vs ===

```
//Equality operators

const age = '18';

//Strict operator
if (age === 18) console.log('Tu es majeur');

//Loose operator
if (age == 18) console.log('Tu es majeur');
```

⚠ Je vous ai montré les deux cas pour l'exemple mais pour éviter des bugs très compliqué à résoudre je vous conseille de ne pas utiliser le loose operator ==

# Les Fondamentaux - Partie 1

## Equality operators == vs ===

```
const favourite = prompt("What's your favorite number ?");
console.log(favourite);
console.log(typeof favourite);

if (favourite === 23) {
  console.log('23 is a cool number');
} else if (favourite === 9) {
  console.log('9 is also a cool number');
} else if (favourite === 7) {
  console.log('7 is also a cool number');
} else {
  console.log('Number is not 23, 7 , 9');
}

if (favourite !== 23) console.log('Why not 23?');
```

# Pratique



Equality operators



# Pratique



- \* Déclarez une variable « numNeighbours » basé sur un prompt comme celui ci : prompt(« Combien de pays frontaliers a votre pays ? »).
- \* Dans un bloc if, s'il y a un seul pays frontalier, loggez dans la console « un seul pays frontalier! ». (Utilisez la loose equality == pour le moment).
- \* Utilisez un bloc else if pour loggez « Plus d'un pays frontalier » dans le cas où « numNeighbour » est plus grand que 1.
- \* Dans le bloc else loggez « Pas de frontières » si « numNeighbour » est égale à 0 ou s'il n'a pas de valeur.
- \* Testez différentes valeurs de « numNeighbour » incluant 1 et 0.
- \* Changez les == par des === et testez le code à nouveau. Notez ce qu'il se passe quand la valeur de « numNeighbour » est 1. Pourquoi se passe t'il cela ?
- \* Finalement, convertissez « numNeighbour » en nombre, et regardez ce qu'il se passe quand la valeur entrée est 1.

# Pratique



```
const numNeighbours = prompt(  
    'How many neighbour countries does your country have?',  
);  
  
// LATER : This helps us prevent bugs  
const numNeighbours = Number(  
    prompt('How many neighbour countries does your country  
have?'),  
);  
  
if (numNeighbours === 1) {  
    console.log('Only 1 border!');  
} else if (numNeighbours > 1) {  
    console.log('More than 1 border');  
} else {  
    console.log('No borders');  
}
```

# Les Fondamentaux - Partie 1

Boolean logic : AND operator

Prenons l'exemple suivant :

A : Sarah a un permis de conduire

B : Sarah a une bonne vue

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

L'opérateur AND renverra TRUE si toutes les conditions sont TRUE, peu importe le nombre de variables !

# Les Fondamentaux - Partie 1

Boolean logic : OR operator

Prenons l'exemple suivant :

A : Sarah a un permis de conduire

B : Sarah a une bonne vue

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

L'opérateur OR renverra TRUE si au moins une des conditions est TRUE !

# Les Fondamentaux - Partie 1

Boolean logic : NOT operator

L'opérateur NOT inverse la valeur de la variable sur laquelle il est utilisé.

# Les fondamentaux - partie 1

Boolean logic : exemple

```
let age = 16
```

Boolean variables

👉 A: **age** est plus grand ou égal à 20

false

👉 B: **age** est plus petit que 30

true

Utilisons les opérateurs

👉 NOT A

true

👉 A AND B

false

👉 A OR B

true

👉 NOT A AND B

true

👉 A OR NOT B

false

# Les fondamentaux - partie 1

## Logical operators

```
const hasDriversLicence = true; //A
const hasGoodVision = false; //B

console.log(hasDriversLicence && hasGoodVision); //AND &&
console.log(hasDriversLicence || hasGoodVision); //OR ||
console.log(!hasDriversLicence); //NOT !

if (hasDriversLicence && hasGoodVision) {
  console.log('Sarah peut conduire !');
} else {
  console.log("Quelqu'un d'autre devrait conduire...");
}

const isTired = false; //C
console.log(hasDriversLicence && hasGoodVision && isTired);

if (hasDriversLicence && hasGoodVision && !isTired) {
  console.log('Sarah peut conduire !');
} else {
  console.log("Quelqu'un d'autre devrait conduire...");
}
```

# Pratique



Logical operators



# Pratique



- \* Commentez le code précédent afin que le prompt ne soit plus pris en compte.
- \* Imaginons que Sarah souhaite changer de pays. Elle veut vivre dans un pays anglophone, avec moins de 50 millions de personnes et qui n'est pas une île.
- \* Écrivez une condition if/else pour aider Sarah à voir si votre pays entre dans ses critères.
- \* Si toutes les conditions sont remplies, vous devrez loggez une string « You should live in Portugal » (Tu devrais vivre au Portugal). Sinon loggez « Portugal does not meet your criteria » (Le Portugal ne coche pas tout tes critères)
- \* Votre pays ne va probablement pas correspondre aux critères de Sarah. Changez certaines variables pour que la condition soit vraie.

# Pratique



```
if (language === 'english' && population < 50 && !isIsland)
{
  console.log(`You should live in ${country} :)`);
} else {
  console.log(`${country} does not meet your criteria :(`);
}
```



# Part 1

# Challenge #3



[LIEN DU CHALLENGE](#)



# Les Fondamentaux - Partie 1

## The Switch statement

```
//SWITCH
const day = 'monday';

switch (day) {
  case 'monday': // day === 'monday'
    console.log('Préparer un cours');
    console.log('Assister à une réunion');
    break;
  case 'tuesday':
    console.log('Préparer une vidéo');
    break;
  case 'wednesday':
  case 'thursday':
    console.log('Ecrire du code');
    break;
  case 'friday':
    console.log('Enregister une vidéo');
    break;
  case 'saturday':
  case 'sunday':
    console.log('Profiter du week-end');
    break;
  default:
    console.log('Pas un jour valide');
}
```

Attention à ne pas oublier d'écrire **break**, sinon le code continuera de s'exécuter jusqu'au prochain cas.



Pour vous entraîner, réécrivez ce code en utilisant des else/if

# Pratique



The Switch



# Pratique



- \* Utilisez un switch pour log la string suivante selon le « language »
  - chinese : « MOST number of native speakers! »
  - spanish : « 2nd place in number of native speakers »
  - english : « 3rd place »
  - hindi : « Number 4 »
  - arabic : « 5th most spoken language »
  - Pour les autres log simplement « Great language too :D »

# Pratique



```
switch (language) {  
    case 'chinese':  
    case 'mandarin':  
        console.log('MOST number of native speakers!');  
        break;  
    case 'spanish':  
        console.log('2nd place in number of native speakers');  
        break;  
    case 'english':  
        console.log('3rd place');  
        break;  
    case 'hindi':  
        console.log('Number 4');  
        break;  
    case 'arabic':  
        console.log('5th most spoken language');  
        break;  
    default:  
        console.log('Great language too :D');  
}
```

# Les Fondamentaux - Partie 1

## Statements & expressions

**Expression JavaScript** : Une expression JavaScript est un morceau de code qui produit une valeur. Par exemple, `2 + 3` est une expression qui produit la valeur 5. Les expressions peuvent être utilisées dans des instructions pour effectuer des calculs ou pour obtenir des résultats.

```
3 + 4;  
1991;  
true && false && !false;
```

# Les Fondamentaux - Partie 1

## Statements & expressions

**Statement JavaScript** : Un statement JavaScript est une instruction qui effectue une action. Les statements incluent des choses comme les boucles (`for`, `while`), les conditions (`if`, `else`), et d'autres instructions qui contrôlent le flux du programme. Les statements ne renvoient pas de valeur comme le font les expressions, mais ils sont essentiels pour la logique et le contrôle du programme.

```
if (23 > 10) {  
  const str = '23 is bigger';  
}
```

# Les fondamentaux - partie 1

## Statements & expressions

**Attention**, on ne peut pas utiliser de statements directement dans un template literals !

```
console.log(`I am ${if(23 > 10) {      Expression expected.  
    const str = '23 is bigger';  
}} years old`));` Declaration or statement expected.
```

# Les fondamentaux - partie 1

## Ternary operator

```
//Ternary operator
const age = 23;
age >= 18
  ? console.log('I like to drink wine')
  : console.log('I like to drink water');

const drink = age >= 18 ? 'wine' : 'water';
console.log(drink);

let drink2;
if (age >= 18) {
  drink2 = 'wine';
} else {
  drink2 = 'water';
}
console.log(drink2);

console.log(`I like to drink ${age >= 18 ? 'wine' : 'water'}`);
```

# Pratique



Ternary operator



# Pratique



- \* Si la population de votre pays est plus grande que 33 millions, utilisez une ternary operator pour log une string comme celle-ci « Portugal's population is above average. » (La population du Portugal est au dessus de la moyenne.) Sinon loggez « Portugal's population is below average. » (La population du Portugal est en dessous de la moyenne.) (Notez qu'un seul mot change dans cette phrase).
- \* Après avoir regardé le résultat, change la population à 13 millions et ensuite à 130 et observez le résultat. Remettez ensuite les valeur de base à la variable.

# Pratique



```
console.log(  
  `${country}'s population is ${population > 33 ? 'above' :  
    'below'} average`,  
);
```



# Part 1

# Challenge #4



[LIEN DU CHALLENGE](#) 

# Les Fondamentaux - Partie 1

Les versions de JavaScript ES5, ES6+ et ESNext



# Les fondamentaux

## Partie 2

# Les Fondamentaux - Partie 2

## Activer le Strict mode

```
//strict mode

'use strict';

let hasDriversLicence = false;
const passTest = true;

if (passTest) hasDriverLicence = true;
if (hasDriversLicence) console.log('I can drive');

// const interface = 'Audio'
// const private = 534
```

# Les Fondamentaux - Partie 2

## Functions

```
//Functions

function logger() {
| console.log('My name is Hedi');
}

//calling / running / invoking function
logger();
logger();
logger();

function fruitProcessor(apples, oranges) {
//console.log(apples, oranges);
const juice = `Juice with ${apples} apples and ${oranges} oranges.`;
return juice;
}

const appleJuice = fruitProcessor(5, 0);
console.log(appleJuice);
console.log(fruitProcessor(5, 0));

const appleOrangeJuice = fruitProcessor(2, 4);
console.log(appleOrangeJuice);
```

# Pratique



Functions



# Pratique



- \* Écrivez une fonction nommée « describeCountry » qui prends 3 paramètres « country », « population » et « capitalCity ». La fonction doit retourné une string à ce format : « Finland has 6 million people and its capital city is Helsinki ». (La Finlande a 6 millions d'habitants et sa capitale est Helsinki)
- \* Appelez la fonction 3 fois avec 3 arguments différents pour chaque appel. Stockez les valeurs retournées dans 3 variables différentes et loggez les dans la console.

# Pratique



```
function describeCountry(country, population, capitalCity) {  
  return `${country} has ${population} million people and  
  its capital city is ${capitalCity}`;  
}  
  
const descPortugal = describeCountry('Portugal', 10,  
  'Lisbon');  
const descGermany = describeCountry('Germany', 83,  
  'Berlin');  
const descFinland = describeCountry('Finland', 6,  
  'Helsinki');  
console.log(descPortugal, descGermany, descFinland);
```

# Les Fondamentaux - Partie 2

## Functions declarations vs. expressions

```
//Function declaration
function calcAge1(birthYear) {
  return 2037 - birthYear;
}

const age1 = calcAge1(1991);
console.log(age1);

//Function expression
const calcAge2 = function (birthYear) {
  return 2037 - birthYear;
};

const age2 = calcAge2(1991);
console.log(age1, age2);
```

💡 La différence majeure entre les deux styles de fonctions est qu'avec la **function declaration** l'appel de la fonction peut avoir lieu l'initialisation de la fonction !

# Pratique



Functions declarations vs.  
Expressions



# Pratique



- \* La population mondiale est de 7900 millions de personnes. Créez une fonction declaration nommée « percentageOfWorld1 » qui recevra en paramètre « population » et qui retournera la valeur en pourcentage de la valeur entrée dans population par rapport à la population mondiale. Par exemple, China has 1441 million people, so it's about 18.2% of the world population (la Chine a 1441 millions d'habitants donc c'est 18.2% de la population mondiale).
- \* Pour calculer le pourcentage, divisez la valeur « population » par 7900 et multipliez par 100.
- \* Appelez « percentageOfWorld1 » pour 3 valeurs de population de pays de votre choix, stockez les résultats dans des variables et loggez-les dans la console.
- \* Créez une function expression qui fait la même chose et qui sera nommée « percentageOfWorld2 »,appelez également cette fonction avec 3 valeurs de population de pays de votre choix (Vous pouvez prendre les même que pour la fonction précédente).

# Pratique



```
function percentageOfWorld1(population) {
    return (population / 7900) * 100;
}

const percentageOfWorld2 = function (population) {
    return (population / 7900) * 100;
};

const percPortugal1 = percentageOfWorld1(10);
const percChina1 = percentageOfWorld1(1441);
const percUSA1 = percentageOfWorld1(332);
console.log(percPortugal1, percChina1, percUSA1);
```

# Les Fondamentaux - Partie 2

## Arrow functions

```
//Arrow function

const calcAge3 = (birthYear) => 2037 - birthYear;
const age3 = calcAge3(1991);
console.log(age3);

const yearsUntilRetirement = (birthYear, firstName) => {
    const age = 2037 - birthYear;
    const retirement = 65 - age;
    //return retirement;
    return `${firstName} retires in ${retirement} years`;
}

console.log(yearsUntilRetirement(1991, 'Hedi'));
console.log(yearsUntilRetirement(1980, 'Bob'));
```

# Pratique



Arrow Functions



# Pratique



\* Recréez la dernière fonction « percentageOfWorld2 » mais cette fois-ci en utilisant une fonction fléchée et en la stockant dans une variable nommée « percentageOfWorld3 ».

# Pratique



```
const percentageOfWorld3 = population => (population / 7900)
* 100;

const percPortugal3 = percentageOfWorld3(10);
const percChina3 = percentageOfWorld3(1441);
const percUSA3 = percentageOfWorld3(332);
console.log(percPortugal3, percChina3, percUSA3);
```

# Les fondamentaux - partie 2

## Functions calling other functions

```
//Functions calling other functions

function cutFruitPieces(fruit) {
    return fruit * 4;
}

function fruitProcessor(apples, oranges) {
    const applePieces = cutFruitPieces(apples);
    const orangePieces = cutFruitPieces(oranges);

    const juice = `Juice with ${applePieces} pieces of apple and ${orangePieces} pieces of orange.`;
    return juice;
}

console.log(fruitProcessor(2, 3));
```

# Pratique



Functions calling  
other functions



# Pratique



- \* Créez une fonction nommée « describePopulation ». Utilisez la fonction que vous préférerez. Cette fonction a 2 paramètres « country » et « population », et elle retourne une string comme ceci : « China has 1441 million people, so it's about 18.2% of the world population. »
- \* Pour calculer le pourcentage utilisez la fonction « percentageOfWorld1 » créée auparavant.
- \* Appelez « describePopulation » avec les données de 3 pays de votre choix.

# Pratique



```
const describePopulation = function (country, population) {  
  const percentage = percentageOfWorld1(population);  
  const description = `${country} has ${population} million  
  people, which is about ${percentage}% of the world.`;  
  console.log(description);  
};  
  
describePopulation('Portugal', 10);  
describePopulation('China', 1441);  
describePopulation('USA', 332);
```

# Les Fondamentaux - Partie 2

## Récapitulatif fonctions



A photograph of the Aurora Borealis (Northern Lights) in a dark sky above a snowy, mountainous landscape.

# Part 2

# Challenge #1



[LIEN DU CHALLENGE](#) 

# Les Fondamentaux - Partie 2

## Introduction to Arrays

```
//Arrays

const friend1 = 'Michael';
const friend2 = 'Steven';
const friend3 = 'Peter';
const friends = ['Michael', 'Steven', 'Peter'];

console.log(friends);

const years = new Array(1991, 1984, 2008, 2020);

console.log(friends[0]);
console.log(friends[2]);

console.log(friends.length);
console.log(friends[friends.length - 1]);

friends[2] = 'Jay';
console.log(friends);

const firstName = 'Hedi';
const hedi = [firstName, 'Rivas', 2037 - 1991, 'teacher', friends];
console.log(hedi);
console.log(hedi.length);
```

# Les Fondamentaux - Partie 2

## Introduction to Arrays

```
//Exercise

const calcAge = function (birthYear) {
    return 2037 - birthYear;
};

const years2 = [1990, 1967, 2002, 2010, 2018];
const age1 = calcAge(years2[0]);
const age2 = calcAge(years2[1]);
const age3 = calcAge(years2[years2.length - 1]);

console.log(age1, age2, age3);

const ages = [
    calcAge(years2[0]),
    calcAge(years2[1]),
    calcAge(years2[years2.length - 1]),
];
console.log(ages);
```

# Pratique



## Introduction To Arrays



# Pratique



- \* Créez un tableau contenant 4 valeurs de population de 4 pays de votre choix. Vous pouvez utiliser les valeurs que vous avez utilisées précédemment. Stockez ce tableau dans une variable appelée « populations »
- \* Loggez dans la console si le tableau a 4 éléments ou non (vrai ou faux).
- \* Créez un tableau appelé "pourcentages" contenant les pourcentages de la population mondiale pour ces 4 valeurs de population. Utilisez la fonction "percentageOfWorld1" que vous avez créée précédemment pour calculer les 4 pourcentages.

# Pratique



```
const populations = [10, 1441, 332, 83];
console.log(populations.length === 4);
const percentages = [
  percentageOfWorld1(populations[0]),
  percentageOfWorld1(populations[1]),
  percentageOfWorld1(populations[2]),
  percentageOfWorld1(populations[3])
];
console.log(percentages);
```

# Les Fondamentaux - Partie 2

## Basic Array Operations (Methods)

```
//Basic array operations (methods)

const friends2 = ['Michael', 'Steven', 'Peter'];
//Add elements
const newLength = friends2.push('Jay');
console.log(friends2);
console.log(newLength);

friends2.unshift('John');
console.log(friends2);

//Remove elements
friends2.pop(); //Last
const popped = friends2.pop();
console.log(popped);
console.log(friends2);

friends2.shift(); //First
console.log(friends2);

console.log(friends2.indexOf('Steven'));
console.log(friends2.indexOf('Bob'));

friends2.push(23);
console.log(friends2.includes('Steven'));
console.log(friends2.includes('Bob'));
console.log(friends2.includes(23));

if (friends2.includes('Steven')) {
  console.log('You have a friend called Steven');
}
```

# Pratique



Basic Array  
Operations



# Pratique



- \* Créez un tableau contenant tous les pays voisins d'un pays de votre choix. Choisissez un pays qui a au moins 2 ou 3 voisins. Stockez le tableau dans une variable appelée « neighbours »
- \* À un moment donné, un nouveau pays appelé "Utopia" est créé dans le voisinage du pays que vous avez choisi. Ajoutez-le donc à la fin du tableau « neighbours ».
- \* Malheureusement, au bout d'un certain temps, le nouveau pays est dissous, il faut donc l'enlever de la fin du tableau.
- \* Si le tableau 'neighbours' n'inclut pas le pays 'Germany', enregistrez sur la console : 'Probably not a central European country :D' (probablement pas un pays d'Europe centrale)
- \* Changez le nom d'une des valeurs du tableau « neighbours ». Pour cela, trouvez l'indice du pays dans le tableau "neighbours", puis utilisez cet indice pour modifier le tableau à la position de cet indice. Par exemple, vous pouvez rechercher "Suède" dans le tableau, puis le remplacer par "République de Suède".

# Pratique



```
const neighbours = ['Norway', 'Sweden', 'Russia'];

neighbours.push('Utopia');
console.log(neighbours);

neighbours.pop();
console.log(neighbours);

if (!neighbours.includes('Germany')) {
  console.log('Probably not a central European country :D');
}

neighbours[neighbours.indexOf('Sweden')] = 'Republic of
Sweden';
console.log(neighbours);
```

A photograph of the Aurora Borealis (Northern Lights) in a dark sky above a snowy, mountainous landscape.

# Part 2

# Challenge #2



[LIEN DU CHALLENGE](#)



# Les Fondamentaux - Partie 2

## Introduction to Objects

```
//Objects

const hediArray = [
  'Hedi',
  'Rivas',
  2037 - 1991,
  'teacher',
  ['Michael', 'Peter', 'Steven'],
];

const hedi = {
  firstName: 'Hedi',
  lastName: 'Rivas',
  age: 2037 - 1991,
  job: 'teacher',
  friends: ['Michael', 'Peter', 'Steven'],
};
```

# Pratique



Introduction to Objects



# Pratique



- \* Créez un objet appelé "myCountry" pour un pays de votre choix, contenant les propriétés "country", "capital", "language", "population" et "neighbours" (un tableau comme celui que nous avons utilisé dans les devoirs précédents).

# Pratique



```
const myCountry = {  
  country: 'Finland',  
  capital: 'Helsinki',  
  language: 'finnish',  
  population: 6,  
  neighbours: ['Norway', 'Sweden', 'Russia']  
};
```

# Les Fondamentaux - Partie 2

## Dot vs. Bracket Notation

```
//Dot vs bracket notation

const hedi = {
    firstName: 'Hedi',
    lastName: 'Rivas',
    age: 2037 - 1991,
    job: 'teacher',
    friends: ['Michael', 'Peter', 'Steven'],
};

console.log(hedi);

console.log(hedi.lastName);
console.log(hedi['lastName']);

const nameKey = 'Name';
console.log(hedi['first' + nameKey]);
console.log(hedi['last' + nameKey]);

//console.log(hedi['last' + nameKey]);

const interestedIn = prompt(
    'What do you want to know about Hedi? Choose between firstName, lastName, age, job and friends'
);

if (hedi[interestedIn]) {
    console.log(hedi[interestedIn]);
} else {
    console.log(
        'Wrong request! Choose between firstName, lastName, age, job and friends'
    );
}

hedi.location = 'Portugal';
hedi['twitter'] = '@hediRivas';
console.log(hedi);
```

# Pratique



Dot vs.  
Bracket Notation



# Pratique



- \* En utilisant l'objet de l'exercice précédent, loggez dans la console une string comme ça : « Finland has 6 million finnish-speaking people, 3 neighbouring countries and a capital called Helsinki » (La Finlande compte 6 millions de finnophones, 3 pays voisins et une capitale appelée Helsinki).
- \* Augmentez la population du pays de deux millions en utilisant la notation en points, puis diminuez-la de deux millions en utilisant la notation entre crochets.

# Pratique



```
console.log(  
  `${myCountry.country} has ${myCountry.population} million  
  ${myCountry.language}-speaking people,  
  ${myCountry.neighbours.length} neighbouring countries and  
  a capital called ${myCountry.capital}.`  
);  
  
myCountry.population += 2;  
console.log(myCountry.population);  
  
myCountry['population'] -= 2;  
console.log(myCountry.population);
```

# Les Fondamentaux - Partie 2

## Dot vs. Bracket Notation

//Challenge 🖥

// Loggez une phrase de ce type dans la console !  
// "Hedi has 3 friends, and his best friend is called Michael"

# Les Fondamentaux - Partie 2

## Object Methods

```
//Object methods

const hedi = {
    firstName: 'Hedi',
    lastName: 'Rivas',
    birthYear: 1991,
    job: 'teacher',
    friends: ['Michael', 'Peter', 'Steven'],
    hasDriversLicence: true,

    // calcAge: function (birthYear) {
    //     return 2037 - birthYear;
    // }

    // calcAge: function () {
    //     //console.log(this);
    //     return 2037 - this.birthYear;
    // }

    calcAge: function () {
        this.age = 2037 - this.birthYear;
        return this.age;
    },
};

//console.log(hedi.calcAge(1991));
//console.log(hedi['calcAge'](1991));

console.log(hedi.calcAge());
```

# Les Fondamentaux - Partie 2

## Object Methods

Challenge 

Créez une nouvelle méthode dans l'objet hedi qui s'appelle getSummary et qui retourne une phrase de ce type :

« Hedi is a 46 years old teacher, and he has a driver's licence. »

# Les Fondamentaux - Partie 2

## Object Methods

```
const hedi = {
    firstName: 'Hedi',
    lastName: 'Rivas',
    birthYear: 1991,
    job: 'teacher',
    friends: ['Michael', 'Peter', 'Steven'],
    hasDriversLicence: true,

    calcAge: function () {
        this.age = 2037 - this.birthYear;
        return this.age;
    },
    getSummary: function () {
        return `${this.firstName} is a ${this.age} years old ${this.job}, and he has a driver's licence`;
    }
};

console.log(hedi.getSummary());
```

# Pratique



Object Methods



# Pratique



- \* 1. Ajoutez une méthode appelée « describe » à l'objet « myCountry ». Cette méthode loggera une chaîne dans la console, de la même manière que la chaîne loggée dans le devoir précédent, mais cette fois-ci en utilisant le mot-clé "this".
- \* 2. Appelez la méthode "describe".
- \* 3. Ajoutez une méthode appelée "checkIsland" à l'objet "myCountry".
- \* Cette méthode définira une nouvelle propriété sur l'objet, appelée "isIsland". "isIsland" sera vrai s'il n'y a pas de pays voisins, et faux s'il y en a. Utilisez l'opérateur ternaire pour définir la propriété.

# Pratique



```
const myCountry = {
    country: 'Finland',
    capital: 'Helsinki',
    language: 'finnish',
    population: 6,
    neighbours: ['Norway', 'Sweden', 'Russia'],

    describe: function () {
        console.log(
            `${this.country} has ${this.population} million
            ${this.language}-speaking people,
            ${this.neighbours.length} neighbouring countries and a
            capital called ${this.capital}.`
        );
    },
};

checkIsland: function () {
    this.isIsland = this.neighbours.length === 0 ? true :
    false;

    // Even simpler version (see why this works...)
    // this.isIsland = !Boolean(this.neighbours.length);
}

myCountry2.describe();
myCountry2.checkIsland();
console.log(myCountry2);
```

A photograph of the Aurora Borealis (Northern Lights) in a dark sky above a snowy, mountainous landscape.

# Part 2

# Challenge #3



[LIEN DU CHALLENGE](#)



# Les Fondamentaux - Partie 2

## Iteration : The for Loop

```
//Iteration: the for loop

// console.log('Lifting weights repetition 1 🤸');
// console.log('Lifting weights repetition 2 🤸');
// console.log('Lifting weights repetition 3 🤸');
// console.log('Lifting weights repetition 4 🤸');
// console.log('Lifting weights repetition 5 🤸');
// console.log('Lifting weights repetition 6 🤸');
// console.log('Lifting weights repetition 7 🤸');
// console.log('Lifting weights repetition 8 🤸');
// console.log('Lifting weights repetition 9 🤸');
// console.log('Lifting weights repetition 10 🤸');

// for loop keeps running while condition is TRUE

for (let rep = 1; rep <= 10; rep++) {
  console.log(`Lifting weights repetition ${rep} 🤸`);
```

# Pratique



Iteration :  
The for Loop



# Pratique



- \* Il y a des élections dans notre pays ! Dans notre petite ville, il n'y a que 50 électeurs.
- \* Utilisez une boucle for pour simuler les 50 personnes qui votent, en loggant une string comme celle-ci dans la console (pour les numéros 1 à 50): : « Voter number 1 is currently voting ». (L'électeur numéro 1 est en train de voter).

# Pratique



```
for (let voter = 1; voter <= 50; voter++)  
  console.log(`Voter number ${voter} is currently voting`);
```

# Les Fondamentaux - Partie 2

## Looping Arrays, Breaking and Continuing

```
//Looping arrays, breaking and continuing

const hediArray = [
  'Hedi',
  'Rivas',
  2037 - 1991,
  'teacher',
  ['Michael', 'Peter', 'Steven'],
  true,
];

const types = [];

for (let i = 0; i < hediArray.length; i++) {
  //Reading from hediArray
  console.log(hediArray[i], typeof hediArray[i]);

  //Filling types array
  //types[i] = typeof hediArray[i];
  types.push(typeof hediArray[i]);
}

console.log(types);

const years = [1991, 2007, 1969, 2020];
const ages = [];

for (let i = 0; i < years.length; i++) {
  ages.push(2037 - years[i]);
}

console.log(ages);
```

# Les Fondamentaux - Partie 2

## Looping Arrays, Breaking and Continuing

```
//continue and break

console.log('--- ONLY STRINGS ---');
for (let i = 0; i < hediArray.length; i++) {
    if (typeof hediArray[i] !== 'string') continue;

    console.log(hediArray[i], typeof hediArray[i]);
}

console.log('--- BREAK WITH NUMBER ---');
for (let i = 0; i < hediArray.length; i++) {
    if (typeof hediArray[i] === 'number') break;

    console.log(hediArray[i], typeof hediArray[i]);
}
```

# Pratique



Looping Arrays, Breaking and  
Continuing



# Pratique



Reprenez le tableau "populations" de l'exercice précédent

- \* Utilisez une boucle for pour calculer un tableau appelé "percentages2" contenant les pourcentages de la population mondiale pour les 4 valeurs de population.
- \* Utilisez la fonction 'percentageOfWorld1' que vous avez créée plus tôt
- \* Confirmez que 'percentages2' contient exactement les mêmes valeurs que le tableau 'percentages' que nous avons créé manuellement dans le devoir précédent et réfléchissez à l'amélioration de cette solution

# Pratique



```
const populations = [10, 1441, 332, 83];
const percentages2 = [];
for (let i = 0; i < populations.length; i++) {
  const perc = percentageOfWorld1(populations[i]);
  percentages2.push(perc);
}
console.log(percentages2);
```

# Les Fondamentaux - Partie 2

## Looping Backwards and Loops in Loops

```
//Looping backwards and loops in loops

const hediArray = [
  'Hedi',
  'Rivas',
  2037 - 1991,
  'teacher',
  ['Michael', 'Peter', 'Steven'],
  true,
];

for (let i = hediArray.length - 1; i >= 0; i--) {
  console.log(i, hediArray[i]);
}

for (let exercise = 1; exercise < 4; exercise++) {
  console.log(`----- Starting exercise ${exercise}`);

  for (let rep = 1; rep < 6; rep++) {
    console.log(`Exercise ${exercise}: Lifting weight repetition ${rep} 🌟`);
  }
}
```

# Pratique



Looping Backwards and Loops in  
Loops



# Pratique



- \* Stockez ce tableau de tableaux dans une variable appelée « listOfNeighbours » [[['Canada', 'Mexico'], ['Spain'], ['Norway', 'Sweden', 'Russia']] ;
- \* Loggez uniquement les pays voisins sur la console, un par un, et non les tableaux entiers. Loggez une string comme ça : « Neighbour : Canada » pour chaque pays.
- \* Vous aurez besoin d'une boucle intérieure pour cela. C'est en fait un peu difficile, ne vous inquiétez pas si c'est trop difficile pour vous ! Mais vous pouvez quand même essayer de le faire 😊

# Pratique



```
const listOfNeighbours = [
    ['Canada', 'Mexico'],
    ['Spain'],
    ['Norway', 'Sweden', 'Russia'],
];

for (let i = 0; i < listOfNeighbours.length; i++)
    for (let y = 0; y < listOfNeighbours[i].length; y++)
        console.log(`Neighbour: ${listOfNeighbours[i][y]}`);
```

# Les Fondamentaux - Partie 2

## The while Loop

```
//While loop

for (let rep = 1; rep <= 10; rep++) {
    console.log(`Lifting weights repetition ${rep} 🤸` );
}

let rep = 1;
while (rep <= 10) {
    console.log(`WHILE: Lifting weights repetition ${rep} 🤸` );
    rep++;
}

let dice = Math.trunc(Math.random() * 6) + 1;
console.log(dice);

while (dice !== 6) {
    console.log(`You rolled a ${dice}`);
    dice = Math.trunc(Math.random() * 6) + 1;
    if (dice === 6) console.log('Loop is about to end...');
}
```

# Pratique



The while Loop



# Pratique



- \* Recréez le défi de la partie « LoopingArrays, Breaking and Continuing », mais cette fois-ci en utilisant une boucle while (appelez le tableau "percentages3").
- \* Réfléchissez à la solution que vous préférez pour cette tâche : l'utilisation d'une boucle **for** ou d'une boucle **while** ?

# Pratique



```
const percentages3 = [];
let i = 0;
while (i < populations.length) {
  const perc = percentageOfWorld1(populations[i]);
  percentages3.push(perc);
  i++;
}
console.log(percentages3);
```

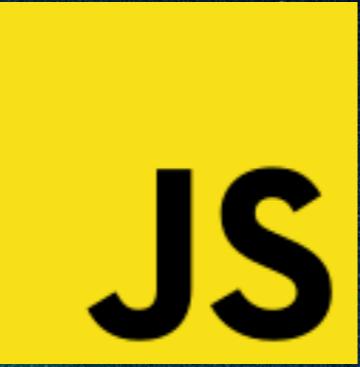


# Part 2

# Challenge #4



[LIEN DU CHALLENGE](#) 



# JavaScript dans le navigateur

## DOM & Events

# Project #1

# Guess My Number!

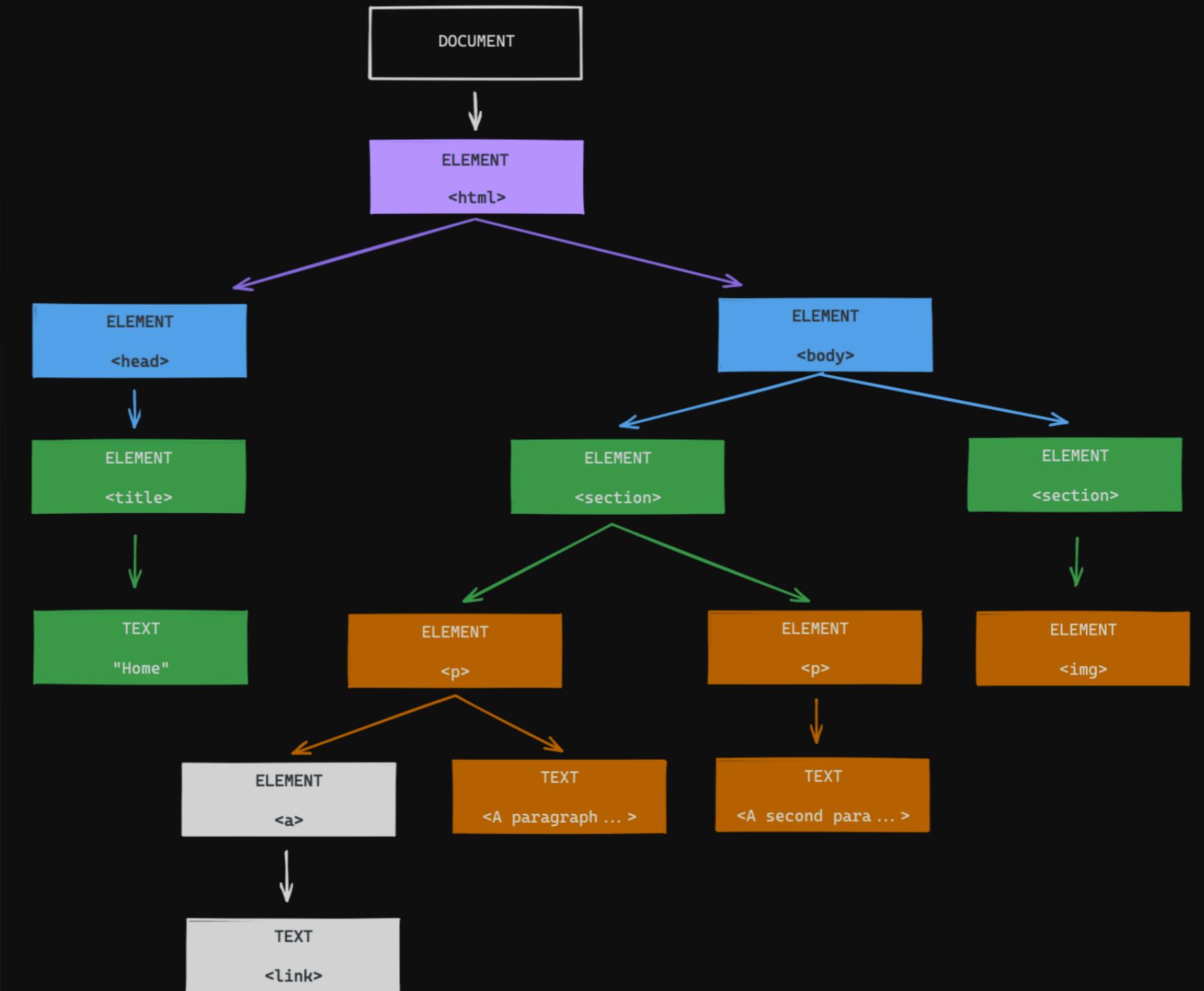
```
git clone https://github.com/HedzDev/guess-my-number.git
```

# JavaScript dans le navigateur

## DOM & Events

### Qu'est-ce que le DOM et la manipulation du DOM

```
<html>
  <head>
    <title>Home</title>
  </head>
  <body>
    <section>
      <p>A paragraph with a <a>link</a></p>
      <p>A second paragraph</p>
    </section>
    <section>
      
    </section>
  </body>
</html>
```



# JavaScript dans le navigateur

## DOM & Events

### Selecting and Manipulating Elements

Pour commencer, nous allons jeter un oeil au HTML. Ensuite nous allons sélectionner et modifier certaines valeurs du DOM et voir ce que cela donne sur notre jeu.

# JavaScript dans le navigateur

## DOM & Events

### Selecting and Manipulating Elements

```
//Selecting and manipulating elements

console.log(document.querySelector('.message').textContent);
document.querySelector('.message').textContent = 'Correct Number! 🎉';

document.querySelector('.number').textContent = 13;
document.querySelector('.score').textContent = 10;

document.querySelector('.guess').value = 23;
console.log(document.querySelector('.guess').value);
```

# JavaScript dans le navigateur

## DOM & Events

### Handle Click Events

Un Event est une action qui se passe sur la page, par exemple une action au clic, une action quand la souris se déplace, etc ...

Nous allons ici créer un listener au clic sur le bouton « check » et logger la valeur de l'input correspondant à la classe « guess ».

# JavaScript dans le navigateur

## DOM & Events

### Handle Click Events

```
//Handling click events

document.querySelector('.check').addEventListener('click', function () {
  console.log(document.querySelector('.guess').value);
});
```

# JavaScript dans le navigateur

## DOM & Events

### Handle Click Events

Nous allons maintenant stocker la valeur de retour de l'input et la stocker dans une constante « guess », puis il faudra logger dans la console la valeur de « guess » et son type. Plus tard nous devrons comparer la valeur de l'input avec un élément de type Number, que faut-il faire pour transformer la valeur de « guess » dans le bon type ?

Faites une première condition dans le block-scope du listener : si la valeur de l'input est inexistant renvoyer un message «  No number! » à la place du message de base correspondant à la classe « message ».

Cliquez sur « check », le message s'affiche-t'il?

# JavaScript dans le navigateur

## DOM & Events

### Handle Click Events

```
//Handling click events

document.querySelector('.check').addEventListener('click', () => {
  const guess = Number(document.querySelector('.guess').value);
  console.log(guess, typeof guess);

  if (!guess) {
    document.querySelector('.message').textContent = '🚫 No number!';
  }
});
```

# JavaScript dans le navigateur DOM & Events

## Implement the Game Logic

Pour faciliter les tests pendant le développement du jeu, affichez le secretNumber dans la page au lieu du « ? » de base.

# JavaScript dans le navigateur

## DOM & Events

### Implement the Game Logic

Ici l'objectif est de mettre en place la logique du jeu. Tout d'abord, tout en haut de votre code, créez une const « secretNumber » qui créera un nombre aléatoire de 1 à 20. Puis, une variable score qui sera égale au nombre 20.

À présent, travaillons sur la logique du jeu, il y aura 3 scénarios :

- le nombre « guess » est égale au nombre « secretNumber », le joueur a donc gagné, le message a affiché sera «  Correct Number ! »
- le nombre « guess » est plus grand que « secretNumber », le message «  Too high ! » doit apparaître, le score doit être réduit de 1 et doit être mis à jour
- le nombre « guess » est plus petit que « secretNumber », le message «  Too low ! » doit apparaître, le score doit être réduit de 1 et doit être mis à jour

# JavaScript dans le navigateur DOM & Events

Implement the Game Logic

Faites un essai, que remarquez-vous ?

# JavaScript dans le navigateur

## DOM & Events

### Implement the Game Logic

Et oui ! Les cas où le nombre « guess » et le nombre « secretNumber » sont plus grand ou plus petit sont en réalité incomplets...

Le score continue de décrémenter quand on passe en dessous de zéro ...

Il faut donc ajouter un if/else statement dans chacun des deux cas pour que le joueur qui arrive à un score de zéro perde la partie. Quand le joueur arrivera à un score de zéro, il faudra afficher le message «  You lost the game ! » et mettre à jour le score du joueur à 0.

À vous de jouer !

# JavaScript dans le navigateur DOM & Events

## Implement the Game Logic

```
//Implementing the Game Logic

const secretNumber = Math.trunc(Math.random() * 20) + 1;
let score = 20;

document.querySelector('.number').textContent = secretNumber;

document.querySelector('.check').addEventListener('click', () => {
  const guess = Number(document.querySelector('.guess').value);
  console.log(guess, typeof guess);

  //When there is no input
  if (!guess) {
    document.querySelector('.message').textContent = '🚫 No number!';

    //When player wins
  } else if (guess === secretNumber) {
    document.querySelector('.message').textContent = 'Correct Number! 🎉';

    //When guess is too high
  } else if (guess > secretNumber) {
    if (score > 1) {
      document.querySelector('.message').textContent = 'Too high! ✘';
      score--;
      document.querySelector('.score').textContent = score;
    } else {
      document.querySelector('.message').textContent = 'You lost the game!';
      document.querySelector('.score').textContent = 0;
    }

    //When guess is too low
  } else if (guess < secretNumber) {
    if (score > 1) {
      document.querySelector('.message').textContent = 'Too low! 🚧';
      score--;
      document.querySelector('.score').textContent = score;
    } else {
      document.querySelector('.message').textContent = 'You lost the game!';
      document.querySelector('.score').textContent = 0;
    }
  }
});
```

# JavaScript dans le navigateur

## DOM & Events

### Manipulating CSS

Nous avons vu comment manipuler des éléments du DOM mais il est aussi possible de manipuler le style des éléments du DOM ! Et ça grâce à la propriété `style` disponible en tapant :

```
document.querySelector(unSélécteur).style
```

# JavaScript dans le navigateur

## DOM & Events

### Manipulating CSS

Nous allons changer, en cas de victoire du joueur, la couleur de fond de la page et augmenter la largeur de la fenêtre contenant le « secretNumber ».

La style du background de l'élément body devra être #60b347 et la largeur de la fenêtre contenant le « secretNumber » devra être 30rem.

# JavaScript dans le navigateur

## DOM & Events

### Manipulating CSS

```
//Manipulating CSS Styles

let secretNumber = Math.trunc(Math.random() * 20) + 1;
document.querySelector('.number').textContent = secretNumber;
let score = 20;

document.querySelector('.check').addEventListener('click', function () {
  const guess = Number(document.querySelector('.guess').value);

  if (!guess) {
    document.querySelector('.message').textContent = '🔴 No number!';
  } else if (guess === secretNumber) {
    document.querySelector('.message').textContent = '🌟 Good number!';
    document.querySelector('body').style.backgroundColor = 'green';
    document.querySelector('.number').style.width = '30rem';
  } else if (guess > secretNumber) {
    if (score > 1) {
      document.querySelector('.message').textContent = '✗ Too High!';
      score--;
      document.querySelector('.score').textContent = score;
    } else {
      document.querySelector('.message').textContent = '❗ Game Over!';
      document.querySelector('body').style.backgroundColor = 'orange';
      document.querySelector('.score').textContent = 0;
    }
  } else if (guess < secretNumber) {
    if (score > 1) {
      document.querySelector('.message').textContent = '↖ Too Low!';
      score--;
      document.querySelector('.score').textContent = score;
    } else {
      document.querySelector('.message').textContent = '❗ Game Over!';
      document.querySelector('body').style.backgroundColor = 'orange';
      document.querySelector('.score').textContent = 0;
    }
  }
});
```

# CodingChallenge #1

Implémentez une nouvelle fonctionnalité : la réinitialisation du jeu, afin que le joueur puisse faire une nouvelle partie !

Vos tâches :

- \* Sélectionnez l'élément avec la classe 'again' et attachez un gestionnaire d'événements de clic
- \* Dans la fonction du gestionnaire, restaurez les valeurs initiales des variables 'score' et 'secretNumber'
- \* Restaurez les conditions initiales des champs de message, de number, de score et de guess
- \* Restaurez également la couleur d'arrière-plan d'origine (#222) et la largeur du number (15rem)

BONNE CHANCE 😊

# JavaScript dans le navigateur

## DOM & Events

### Coding challenge #1

```
//Resetting the Game
document.querySelector('.again').addEventListener('click', () => {
  score = 20;
  secretNumber = Math.trunc(Math.random() * 20) + 1;
  document.querySelector('.message').textContent = 'Start guessing...';
  document.querySelector('.score').textContent = score;
  document.querySelector('.guess').value = '';
  document.querySelector('body').style.backgroundColor = '#222';
  document.querySelector('.number').style.width = '15rem';
  document.querySelector('.number').textContent = '?';
});
```

# JavaScript dans le navigateur

## DOM & Events

### Implement Highscores

Nous allons maintenant créer la logique du HighScores. Créez tout d'abord sous la variable score, une variable « highScore » initialisé à 0. Cachez à présent la valeur du « secretNumber » et affichez-la simplement , en cas de victoire du joueur.

Maintenant, l'objectif est de mettre à jour le meilleur score du joueur dans highScore. Il faudra pour cela créer un if dans le bloc où le jour a gagné. La condition est simple, si le score du joueur est supérieur au highscore alors ...

# JavaScript dans le navigateur DOM & Events

## Implement Highscores



Je vous laisse chercher !

# JavaScript dans le navigateur

## DOM & Events

### Implement Highscores

```
let secretNumber = Math.trunc(Math.random() * 20) + 1;
let score = 20;
let highscore = 0;

document.querySelector('.check').addEventListener('click', function () {
  const guess = Number(document.querySelector('.guess').value);
  if (!guess) {
    document.querySelector('.message').textContent = '❗ No number!';
  } else if (guess === secretNumber) {
    document.querySelector('.message').textContent = '🎉 Good number!';
    document.querySelector('.number').textContent = secretNumber;
    document.querySelector('body').style.backgroundColor = 'green';
    document.querySelector('.number').style.width = '30rem';
    if (score > highscore) {
      highscore = score;
      document.querySelector('.highscore').textContent = highscore;
    }
  } else if (guess > secretNumber) {
    document.querySelector('.message').textContent = '⬆️ Too high!';
  } else {
    document.querySelector('.message').textContent = '⬇️ Too low!';
  }
});
```

# JavaScript dans le navigateur

## DOM & Events

### Refactoring: DRY

DRY pour Don't Repeat Yourself !

C'est un concept très important en programmation ! C'est un des principes de ce que l'on appelle le Clean Code ✨ !

Analysez le code et chercher les éléments qui se répètent et qui pourraient être refactoriser.

# JavaScript dans le navigateur

## DOM & Events

### Refactoring: DRY

C'est fait ? Vous avez trouvé ? Parfait !

Créez maintenant une fonction « `displayMessage` » en dessous de la variable `highScore`, au début de votre code.

Cette fonction prendra un paramètre « **message** » et devra faire passer ce paramètre message dans

```
document.querySelector('.message').textContent = message
```

Essayer à présent par vous-même d'utiliser cette fonction aux bons endroits et de refactoriser le code en utilisant les notions vues dans les cours, les ternaires par exemple 😊

# JavaScript dans le navigateur

## DOM & Events

Refactoring: DRY

```
const displayMessage = function (message) {
  document.querySelector('.message').textContent = message;
};
```

# JavaScript dans le navigateur DOM & Events

## Refactoring: DRY

```
function displayMessage(message) {
  document.querySelector('.message').textContent = message;
}

document.querySelector('.check').addEventListener('click', function () {
  const guess = Number(document.querySelector('.guess').value);
  if (!guess) {
    displayMessage('⚠️ No number!');
  } else if (guess === secretNumber) {
    displayMessage('🎉 Good number!');
    document.querySelector('.number').textContent = secretNumber;
    document.querySelector('body').style.backgroundColor = 'green';
    document.querySelector('.number').style.width = '30rem';
    if (score > highscore) {
      highscore = score;
      document.querySelector('.highscore').textContent = highscore;
    }
  } else if (guess !== secretNumber) {
    if (score > 1) {
      displayMessage(guess > secretNumber ? '📈 Too High!' : '📉 Too Low!');
      score--;
      document.querySelector('.score').textContent = score;
    } else {
      displayMessage('💀 Game Over!');
      document.querySelector('.score').textContent = 0;
    }
  }
});
```

# Project #2

# Modal Window

```
git clone https://github.com/HedzDev/modal-starter.git
```

# JavaScript dans le navigateur

## DOM & Events

### Travailler avec les Class

Après avoir compris le travail demandé et en ayant vu la maquette, mettez en place la logique pour afficher puis fermer les fenêtres modales. Comme montré pendant la présentation, il faudra pouvoir fermer la modal en appuyant sur la croix, en cliquant en dehors de la modal ou bien en utilisant la touche « esc ».

# JavaScript dans le navigateur

## DOM & Events

### Travailler avec les Class

```
// Selecting elements
const modal = document.querySelector('.modal');
const overlay = document.querySelector('.overlay');
const btnCloseModal = document.querySelector('.close-modal');
const btnsOpenModal = document.querySelectorAll('.show-modal');

const openModal = () => {
  modal.classList.remove('hidden');
  overlay.classList.remove('hidden');
};

const closeModal = function () {
  modal.classList.add('hidden');
  overlay.classList.add('hidden');
};

// Looping through the buttons
for (let i = 0; i < btnsOpenModal.length; i++) {
  btnsOpenModal[i].addEventListener('click', openModal);
}

btnCloseModal.addEventListener('click', closeModal);

overlay.addEventListener('click', closeModal);
```

# JavaScript dans le navigateur

## DOM & Events

Utiliser la touche « Esc » du clavier

```
document.addEventListener('keydown', function (e) {  
    console.log(e);  
    if (e.key === 'Escape' && !modal.classList.contains('hidden')) {  
        closeModal();  
    }  
});
```

# Project #2

# Pig Game

```
git clone https://github.com/HedzDev/pig-game-starter.git
```

# JavaScript dans le navigateur

## DOM & Events

### Pig Game Init

Commençons par initialiser le jeu en sélectionnant les éléments du DOM que nous allons utiliser et en mettant les scores à zéro.

Le dé doit être caché en début de partie. Et il devra s'afficher au premier clic sur le bouton « roll dice »

# JavaScript dans le navigateur

## DOM & Events

### Pig Game Init

```
// Selecting elements
const score0El = document.querySelector('#score--0');
const score1El = document.getElementById('score--1');
const diceEl = document.querySelector('.dice');

score0El.textContent = 0;
score1El.textContent = 0;
diceEl.classList.add('hidden');
```

# JavaScript dans le navigateur

## DOM & Events

### Rolling the dice

L'objectif ici est de mettre en place la logique de lancement du dé. On sélectionne tous les éléments que nous allons utiliser du Dom et on les stockent dans des variables.

Au clic sur le bouton 'roll dice', on crée un nombre aléatoire en 1 et 6 et on fait apparaître le dé avec ce nombre.

Si le dé est autre que la valeur 1, on ajoute le score du dé au score courant ('CURRENT') du joueur.

# JavaScript dans le navigateur DOM & Events

## Rolling the dice

```
// Selecting elements
const score0El = document.querySelector('#score--0');
const score1El = document.getElementById('score--1');
const current0El = document.getElementById('current--0');
const current1El = document.getElementById('current--1');
const diceEl = document.querySelector('.dice');
const btnNew = document.querySelector('.btn--new');
const btnRoll = document.querySelector('.btn--roll');
const btnHold = document.querySelector('.btn--hold');
const player0 = document.querySelector('.player--0');
const player1 = document.querySelector('.player--1');

// Starting conditions
score0El.textContent = 0;
score1El.textContent = 0;
diceEl.classList.add('hidden');

let currentScore = 0;

// Rolling dice functionnality

btnRoll.addEventListener('click', () => {
    // 1. Generating a random dice roll
    const dice = Math.trunc(Math.random() * 6) + 1;

    // 2. Display dice
    diceEl.classList.remove('hidden');
    diceEl.src = `dice-${dice}.png`;

    // 3. Check for rolled 1
    if (dice !== 1) {
        // Add dice to current score
        currentScore += dice;
        current0El.textContent = currentScore; //CHANGE LATER
    } else {
        // Switch to next player
    }
});
```

# JavaScript dans le navigateur

## DOM & Events

### Switching the Active Player

À présent, nous allons mettre en place le cas où le joueur fait un jet de dé de valeur 1. Dans ce cas, il faudra switcher de joueur actif.

Ce que nous allons faire pour cela, c'est créer une variable `activePlayer` qui sera initialisé à 0. La méthode sera la suivante : le joueur 0 représente le premier joueur et le nombre 1 représentera le second joueur. Si le premier joueur lance le dé et qu'il tombe sur 1 et bien dans ce cas on set la valeur de la variable `activePlayer` à 1 !

# JavaScript dans le navigateur

## DOM & Events

### Switching the Active Player

Mais avant ça, il faut set, dans le else du cas où le joueur tombe sur 1, le score du joueur qui vient de tomber sur 1 à 0 !

Ensuite, on set également son currentScore à 0. Maintenant nous pouvons switch la valeur de activePlayer de 0 à 1 en utilisant une ternaire !

Et voilà ! Nous avons pesque terminé cette partie ! Il ne nous manque plus qu'a changer le style pour éclairer la partie correspondante au joueur actif ! Analysez le HTML et trouvez la classe qui pourrait nous servir à **toggle** le style de la couleur de fond des joueurs !

# JavaScript dans le navigateur DOM & Events

## Switching the Active Player

```
let currentScore = 0;
let activePlayer = 0;
// Rolling dice functionnality

btnRoll.addEventListener('click', () => {
    // 1. Generating a random dice roll
    const dice = Math.trunc(Math.random() * 6) + 1;

    // 2. Display dice
    diceEl.classList.remove('hidden');
    diceEl.src = `dice-${dice}.png`;

    // 3. Check for rolled 1
    if (dice !== 1) {
        // Add dice to current score
        currentScore += dice;
        document.getElementById(`current--${activePlayer}`).textContent =
            currentScore;
    } else {
        // Switch to next player
        document.getElementById(`current--${activePlayer}`).textContent = 0;
        currentScore = 0;
        activePlayer = activePlayer === 0 ? 1 : 0;
        player0El.classList.toggle('player--active');
        player1El.classList.toggle('player--active');
    }
});
```

# JavaScript dans le navigateur

## DOM & Events

### Holding Current Score

Nous allons à présent mettre en place la fonctionnalité du bouton « hold ».

L'objectif ici est le suivant : lorsque un joueur souhaitera sauvegarder son score il devra cliquer sur 'hold'. À ce moment là, il faudra switcher de joueur et le joueur suivant pourra tirer les dés.

Nous allons créer, pour stocker les scores des deux joueurs, un tableau qui sera initialisé à [0, 0]. On le nommera **scores**.

Nous allons également créer une variable « playing » qui sera initialisé à **true** et qui nous permettra de bloquer le jeu en cas de victoire d'un joueur.

# JavaScript dans le navigateur

## DOM & Events

### Holding Current Score

```
const scores = [0, 0];
let currentScore = 0;
let activePlayer = 0;
let playing = true;
```

# JavaScript dans le navigateur DOM & Events

## Holding Current Score

```
btnHold.addEventListener('click', () => {
    //1. add current score to active player's score
    scores[activePlayer] += currentScore;
    document.querySelector(`#score--${activePlayer}`).textContent =
        scores[activePlayer];

    //2. check if player's score is >= 100
    if (scores[activePlayer] >= 100) {
        playing = false;
        document.querySelector(`#name--${activePlayer}`).textContent = 'Winner!';
        diceEl.classList.add('hidden');
        document
            .querySelector(`.player--${activePlayer}`)
            .classList.add('player--winner');
        document
            .querySelector(`.player--${activePlayer}`)
            .classList.remove('player--active');
    } else {
        //3. switch to next player
        document.querySelector(`#current--${activePlayer}`).textContent = 0;
        currentScore = 0;
        activePlayer = activePlayer === 0 ? 1 : 0;
        player0.classList.toggle('player--active');
        player1.classList.toggle('player--active');
    }
});
```

# JavaScript dans le navigateur

## DOM & Events

### Holding Current Score

En analysant le code vous remarquerez certainement qu'il y a des blocs de codes qui se répètent ! Et ça, pour les développeurs chevronnés et consciencieux que nous sommes 😎, on n'aime pas du tout !

Appliquez la méthode DRY en créant une fonction switchPlayer(), et regarder la slide d'après pour les consignes concernant cette fonction.

# JavaScript dans le navigateur DOM & Events

## Holding Current Score

Je vous ai bien eu !



Pas de consignes ici, faites-le par vous-même c'est facile !

# JavaScript dans le navigateur DOM & Events

## Holding Current Score

```
btnHold.addEventListener('click', () => {
  if (playing) {
    // 1. Add current score to active player's score
    scores[activePlayer] += currentScore;
    document.getElementById(`score--${activePlayer}`).textContent =
      scores[activePlayer];

    // 2. Check if the player's score is >= 100
    if (scores[activePlayer] >= 20) {
      playing = false;
      diceEl.classList.add('hidden');
      document
        .querySelector(`.player--${activePlayer}`)
        .classList.add('player--winner');

      document
        .querySelector(`.player--${activePlayer}`)
        .classList.remove('player--active');
    } else {
      // Switch to next player
      switchPlayer();
    }
  }
});
```

# JavaScript dans le navigateur

## DOM & Events

### Resetting the Game

La dernière étape est de mettre en place le reset du jeu au clic sur le bouton « new game ».

Créez une fonction « init » et mettez à l'intérieur tout les éléments nécessaires pour commencer une partie de zéro.

# JavaScript dans le navigateur

## DOM & Events

### Resetting the Game

```
// Starting conditions
let scores, currentScore, activePlayer, playing;

const init = () => {
  scores = [0, 0];
  currentScore = 0;
  activePlayer = 0;
  playing = true;

  score0El.textContent = 0;
  score1El.textContent = 0;
  current0El.textContent = 0;
  current1El.textContent = 0;

  diceEl.classList.add('hidden');
  player0El.classList.remove('player-winner');
  player1El.classList.remove('player-winner');
  player0El.classList.add('player--active');
  player1El.classList.remove('player--active');
};

init();
```

# JavaScript dans le navigateur

## DOM & Events

### Creating and removing elements

```
const title = document.querySelector('.title');

//creating element
const text = document.createElement('p');

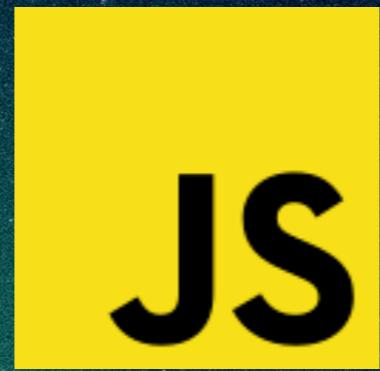
text.innerHTML = 'Un peu de texte...';
text.style.color = 'red';

//adding a class
text.classList.add('text-color');

// adding an Id
text.setAttribute('id', 'text');

// insert element
title.insertAdjacentElement('beforebegin', text);
title.append(text);

// deleting
//text.remove();
```



# Data Structures, Modern Operators and Strings

# Data Structures, Modern Operators and Strings

## Destructuring Arrays

```
// Destructuring Arrays

const restaurant = {
  name: 'Classico Italiano',
  location: 'Via Angelo Tavanti 23, Firenze, Italy',
  categories: ['Italian', 'Pizzeria', 'Vegetarian', 'Organic'],
  starterMenu: ['Focaccia', 'Bruschetta', 'Garlic Bread', 'Caprese Salad'],
  mainMenu: ['Pizza', 'Pasta', 'Risotto'],

  order: function (starterIndex, mainIndex) {
    // return the starter and main course
    return [this.starterMenu[starterIndex], this.mainMenu[mainIndex]];
  },
};

const arr = [2, 3, 4];
const a = arr[0];
const b = arr[1];
const c = arr[2];

const [x, y, z] = arr;
console.log(x, y, z);
console.log(arr);

let [main, , secondary] = restaurant.categories;
console.log(main, secondary);

// Switching variables
// const temp = main;
// main = secondary;
// secondary = temp;
// console.log(main, secondary);

// Switching variables using destructuring
[main, secondary] = [secondary, main];
console.log(main, secondary);

// Recieve 2 return values from a function
const [starter, mainCourse] = restaurant.order(2, 0);
console.log(starter, mainCourse);
```

# Data Structures, Modern Operators and Strings

## Destructuring Arrays

```
// Nested destructuring
const nested = [2, 4, [5, 6]];
// const [i, , j] = nested;
// console.log(i, j);
const [i, , [j, k]] = nested;
console.log(i, j, k);

// Default values
const [p = 1, q = 1, r = 1] = [8, 9];
console.log(p, q, r);
```

# Data Structures, Modern Operators and Strings

## Destructuring Objects

```
// Destructuring Objects
const { name, openingHours, categories } = restaurant;
console.log(name, openingHours, categories);

// Renaming variables
const {
  name: restaurantName,
  openingHours: hours,
  categories: tags,
} = restaurant;
console.log(restaurantName, hours, tags);

// Default values
const { menu = [], starterMenu: starters = [] } = restaurant;
console.log(menu, starters);

// Mutating variables
let a1 = 111;
let b1 = 999;
const obj = { a1: 23, b1: 7, c1: 14 };
({ a1, b1 } = obj);
console.log(a1, b1);

// Nested objects
const {
  fri: { open: o, close: cl },
} = openingHours;
console.log(o, cl);
```

# Data Structures, Modern Operators and Strings

## The Spread Operator (...)

```
// spread operator
const arr1 = [7, 8, 9];
const newArr = [1, 2, ...arr1];
console.log(newArr);
console.log(...newArr);

const newMenu = [...restaurant.mainMenu, 'Gnocci'];
console.log(newMenu);

// Copy array
const mainMenuCopy = [...restaurant.mainMenu];
console.log(mainMenuCopy);

// Join 2 arrays
const menu1 = [...restaurant.starterMenu, ...restaurant.mainMenu];
console.log(menu1);

// Iterables: arrays, strings, maps, sets. NOT objects
const str = 'Jonas';
const letters = [...str, ' ', 'S.'];
console.log(letters);
console.log(...str);

// Real world example
const ingredients = [
  prompt("Let's make pasta! Ingredient 1?"),
  prompt('Ingredient 2?'),
  prompt('Ingredient 3?'),
];

console.log(ingredients);

restaurant.orderPasta(ingredients[0], ingredients[1], ingredients[2]);
restaurant.orderPasta(...ingredients);

// Objects
const newRestaurant = { foundedIn: 1998, ...restaurant, founder: 'Guiseppe' };
console.log(newRestaurant);

const restaurantCopy = { ...restaurant };
restaurantCopy.name = 'Ristorante Roma';
```

# Data Structures, Modern Operators and Strings

## Rest Pattern and Parameters

```
// Rest Pattern and Parameters
// 1) Destructuring
// SPREAD, because on RIGHT side of =
const arr1 = [1, 2, ...[3, 4]];

// REST, because on LEFT side of =
const [a2, b2, ...others] = [1, 2, 3, 4, 5];
console.log(a2, b2, others);

const [pizza, , risotto, ...otherFood] = [
  ...restaurant.mainMenu,
  ...restaurant.starterMenu,
];
console.log(pizza, risotto, otherFood);

// Objects
const { sat, ...weekdays } = restaurant.openingHours;
console.log(weekdays);

// 2) Functions
const add = function (...numbers) {
  // console.log(numbers);
  let sum = 0;
  for (let i = 0; i < numbers.length; i++) {
    sum += numbers[i];
  }
  console.log(sum);
};

add(2, 3);
add(5, 3, 7, 2);
add(8, 2, 5, 3, 2, 1, 4);

const x1 = [23, 5, 7];
add(...x1);

restaurant.orderPizza('mushrooms', 'onion', 'olives', 'spinach');
restaurant.orderPizza('mushrooms');
```

# Data Structures, Modern Operators and Strings

## Récap du **spread operator**

Le spread operator est utilisé pour étendre un tableau ou un objet. Il permet de décomposer les éléments d'un tableau ou les propriétés d'un objet pour les utiliser dans un autre contexte.

```
const tableau1 = [1, 2, 3];
const tableau2 = [4, 5, ...tableau1];

console.log(tableau2); // affiche [4, 5, 1, 2, 3]
```

# Data Structures, Modern Operators and Strings

## Récap du **rest** operator

Le rest operator est utilisé pour rassembler plusieurs arguments en un seul tableau. Il est souvent utilisé dans les fonctions pour gérer un nombre variable d'arguments.

```
function maFonction(...args) {  
    console.log(args);  
}  
  
maFonction(1, 2, 3); // affiche [1, 2, 3]
```

Dans cet exemple, le rest operator a été utilisé pour rassembler les arguments passés à la fonction maFonction dans un tableau appelé args.

# Data Structures, Modern Operators and Strings

## Short circuiting (&& and || operators)

```
// Short Circuiting (&& and ||)
console.log('---- OR ----');
// Use ANY data type, return ANY data type, short-circuiting
console.log(3 || 'Hedi');
console.log('' || 'Hedi');
console.log(true || 0);
console.log(undefined || null);

console.log(undefined || 0 || '' || 'Hello' || 23 || null);

// restaurant.numGuests = 23;
const guests1 = restaurant.numGuests ? restaurant.numGuests : 10;
console.log(guests1);

const guests2 = restaurant.numGuests || 10;
console.log(guests2);

console.log('---- AND ----');
console.log(0 && 'Hedi');
console.log(7 && 'Hedi');
console.log(7 && 'Hedi' && 23 && null && 'Hedi');

// Practical example
if (restaurant.orderPizza) {
  restaurant.orderPizza('mushrooms', 'spinach');
}

restaurant.orderPizza && restaurant.orderPizza('mushrooms', 'spinach');
```

# Data Structures, Modern Operators and Strings

## The Nullish Coalescing Operator (??)

```
// Nullish Coalescing Operator (??)
restaurant.numGuests = 0;
const guests3 = restaurant.numGuests || 10;
console.log(guests3);

// Nullish: null and undefined (NOT 0 or '')
const guestCorrect = restaurant.numGuests ?? 10;
console.log(guestCorrect);
```

# Data Structures, Modern Operators and Strings

## Logical Assignment Operators

```
// Logical Assignment Operators

const rest1 = {
  name: 'Capri',
  // numGuests : 23,
  numGuests: 0,
};
const rest2 = {
  name: 'La Piazza',
  owner: 'Guisepppe Rizzi',
};

// OR assignment operator
// rest1.numGuests = rest1.numGuests || 10;
// rest2.numGuests = rest2.numGuests || 10;
// rest1.numGuests ||= 10;
// rest2.numGuests ||= 10;

// nullish assignment operator (null or undefined)
rest1.numGuests ??= 10;
rest2.numGuests ??= 10;

// AND assignment operator
// rest1.owner = rest1.owner && '<ANONYMOUS>';
// rest2.owner = rest2.owner && '<ANONYMOUS>';

rest1.owner &&= '<ANONYMOUS>';
rest2.owner &&= '<ANONYMOUS>';

console.log(rest1);
console.log(rest2);
```

# Data Structures, Modern Operators and Strings

## CodingChallenge #1

[LIEN DU CHALLENGE](#)



# Data Structures, Modern Operators and Strings

## Looping Arrays : The **for of** Loop

```
// Looping Arrays: The for-of Loop
const menu2 = [...restaurant.starterMenu, ...restaurant.mainMenu];

for (const item of menu2) console.log(item);

// for (const item of menu2.entries()) {
//   console.log(`${item[0] + 1}: ${item[1]}`);
// }

for (const [i, el] of menu2.entries()) {
  console.log(`${i + 1}: ${el}`);
}
```

# Data Structures, Modern Operators and Strings

## Objects Literals Améliorés

```
// Enhanced Object Literals
const weekdays1 = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun'];

const openingHours1 = {
  [weekdays1[3]]: {
    open: 12,
    close: 22,
  },
  [weekdays1[4]]: {
    open: 11,
    close: 23,
  },
  [weekdays1[5]]: {
    open: 0, // Open 24 hours
    close: 24,
  },
};

const restaurant = {
  name: 'Classico Italiano',
  location: 'Via Angelo Tavanti 23, Firenze, Italy',
  categories: ['Italian', 'Pizzeria', 'Vegetarian', 'Organic'],
  starterMenu: ['Focaccia', 'Bruschetta', 'Garlic Bread', 'Caprese Salad'],
  mainMenu: ['Pizza', 'Pasta', 'Risotto'],
  openingHours1,

  order(starterIndex, mainIndex) {
    // return the starter and main course
    return [this.starterMenu[starterIndex], this.mainMenu[mainIndex]];
  },

  orderDelivery({ starterIndex, mainIndex, time, address }) {
    console.log(
      `Order received! ${this.starterMenu[starterIndex]} and ${this.mainMenu[mainIndex]}`)
  },
}

orderPasta(ing1, ing2, ing3) {
  console.log(`Here is your pasta with ${ing1}, ${ing2}, ${ing3}.`);
},
```

# Data Structures, Modern Operators and Strings

## Optional Chaining (?.)

```
// Optional Chaining (?.)

if (restaurant.openingHours && restaurant.openingHours.mon)
  console.log(restaurant.openingHours.mon.open);

// console.log(restaurant.openingHours.mon.open);

// With optional chaining
console.log(restaurant.openingHours.mon?.open);
console.log(restaurant.openingHours?.mon?.open);

// Example

const days = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun'];

for (const day of days) {
  const open = restaurant.openingHours[day]?.open ?? 'closed';
  console.log(`#${day}, we open at ${open}`);
}

// Methods
console.log(restaurant.order?(0, 1) ?? 'Method does not exist');
console.log(restaurant.orderRisotto?(0, 1) ?? 'Method does not exist');

// Arrays
const users = [{ name: 'Hedi', email: 'contact@hedi.io' }];

console.log(users[0]?.name ?? 'User array empty');

if (users.length > 0) console.log(users[0].name);
else console.log('User array empty');
```

# Data Structures, Modern Operators and Strings

## Looping Objects : Objects Keys, Values and Entries

```
// Looping Objects: Object Keys, Values, and Entries

// Property NAMES
const properties = Object.keys(openingHours);
console.log(properties);

let openStr = `We are open on ${properties.length} days: `;
for (const day of properties) {
  openStr += `${day}, `;
}
console.log(openStr);

// Property VALUES
const values = Object.values(openingHours);
console.log(values);

// Entire object
const entries = Object.entries(openingHours);
// console.log(entries);

// [key, value]
for (const [day, { open, close }] of entries) {
  console.log(`On ${day} we open at ${open} and close at ${close}`);
}
```

# Data Structures, Modern Operators and Strings

## CodingChallenge #2

[LIEN DU CHALLENGE](#)



# Data Structures, Modern Operators and Strings

## Sets

```
// Sets
const ordersSet = new Set([
  'Pasta',
  'Pizza',
  'Pizza',
  'Risotto',
  'Pasta',
  'Pizza',
]);
console.log(ordersSet);
console.log(new Set('Hedi'));
console.log(ordersSet.size);
console.log(ordersSet.has('Pizza'));
console.log(ordersSet.has('Bread'));

ordersSet.add('Garlic Bread');
ordersSet.add('Garlic Bread');
console.log(ordersSet);

ordersSet.delete('Risotto');
console.log(ordersSet);

// ordersSet.clear();
// console.log(ordersSet);

for (const order of ordersSet) console.log(order);

// Example
const staff = ['Waiter', 'Chef', 'Waiter', 'Manager', 'Chef', 'Waiter'];
const staffUnique = new Set(staff);
console.log(staffUnique);
```

# Data Structures, Modern Operators and Strings

## Maps : Fundamentals

```
// Maps: Fundamentals
const rest = new Map();
rest.set('name', 'Classico Italiano');
rest.set(1, 'Firenze, Italy');
rest.set(2, 'Lisbon, Portugal');

console.log(rest.set(3, 'San Francisco, USA'));

rest
  .set('categories', ['Italian', 'Pizzeria', 'Vegetarian', 'Organic'])
  .set('open', 11)
  .set('close', 23)
  .set(true, 'We are open')
  .set(false, 'We are closed');

console.log(rest.get('name'));
console.log(rest.get(true));
console.log(rest.get(1));

const time = 21;
console.log(rest.get(time > rest.get('open') && time < rest.get('close')));

console.log(rest.has('categories'));
rest.delete(2);
// rest.clear();
console.log(rest);
```

# Data Structures, Modern Operators and Strings

## Maps : Iteration

```
// Maps: Iteration

const question = new Map([
  ['question', 'What is the best programming language ?'],
  [1, 'C'],
  [2, 'Java'],
  [3, 'JavaScript'],
  ['correct', 3],
  [true, 'Good answer 😊'],
  [false, 'Try again!'],
]);

// Convert object to map
console.log(Object.entries(openingHours));
const hoursMap = new Map(Object.entries(openingHours));
console.log(hoursMap);

for (const [key, value] of question) {
  if (typeof key === 'number') console.log(`Answer ${key} : ${value}`);
}

// const answer = Number(prompt('You answer'));
// console.log(answer);

// console.log(question.get(question.get('correct') === answer));

// Convert map to array
console.log([...question]);
// console.log([...question.entries()]);
console.log([...question.keys()]);
console.log([...question.values()]);
```

# Data Structures, Modern Operators and Strings

## Résumé : Quelles Structures de Data utilisées ?

ARRAYS

VS.

SETS

```
tasks = ['Code', 'Eat', 'Code'];
// ['Code', 'Eat', 'Code']
```

```
tasks = new Set(['Code', 'Eat', 'Code']);
// {'Code', 'Eat'}
```

👉 À utiliser quand vous voulez une liste ordonnée de valeurs (peut contenir des doublons).

👉 À utiliser quand vous devez manipuler de la data

👉 À utiliser quand vous avez besoin de travailler avec des valeurs uniques.

👉 À utiliser quand on cherche plus de performances.

👉 À utiliser pour effacer les doublons des tableaux.

# Data Structures, Modern Operators and Strings

## Résumé : Quelles Structures de Data utilisées ?

Pour résumé, si vous avez besoin d'accéder fréquemment aux éléments par leur indice ou si vous devez conserver l'ordre des éléments, les tableaux peuvent être plus appropriés.

Les sets sont particulièrement utiles lorsque vous devez effectuer des opérations de recherche, de suppression et de vérification d'unicité d'éléments de manière efficace.

# Data Structures, Modern Operators and Strings

## Résumé : Quelles Structures de Data utilisées ?

OBJECTS

```
task = {  
    task: 'Code',  
    date: 'today',  
    repeat: true,  
};
```

VS.

MAPS

```
task = new Map([  
    ['task', 'Code'],  
    ['date', 'today'],  
    [false, 'Start coding!'],  
]);
```

- 👉 La plus traditionnelle key/value façon de stocker.
- 👉 Facile à écrire et à accéder aux valeurs avec la dot et bracket notation.
- 👉 À utiliser quand vous avez besoin d'inclure des fonctions dans l'objet (methods).
- 👉 À utiliser quand vous travaillez en JSON.

- 👉 Meilleures performances pour les utilisations avec de nombreuses additions/suppressions de keys/values.
- 👉 Les keys peuvent avoir n'importe quel type de données.
- 👉 Simple à itérer.
- 👉 Taille facile à calculer (size).
- 👉 À utiliser quand vous avez besoin d'une key autre qu'une string.

# Data Structures, Modern Operators and Strings

## Résumé : Quelles Structures de Data utilisées ?

Les objets sont des collections de données non ordonnées, utilisant des chaînes de caractères comme clés, et qui peuvent stocker des valeurs de tout type.

Les Maps sont également des collections de données, mais elles maintiennent l'ordre d'insertion, acceptent n'importe quel type de données en tant que clé, et peuvent stocker n'importe quel type de valeur.

# Data Structures, Modern Operators and Strings

## CodingChallenge #3

[LIEN DU CHALLENGE](#)



# Data Structures, Modern Operators and Strings

## Working with Strings - Part 1

```
// Working with Strings - Part 1
const airline = 'TAP Air Portugal';
const plane = 'A320';

console.log(plane[0]);
console.log(plane[1]);
console.log(plane[2]);
console.log('B737'[0]);

console.log(airline.length);
console.log('B737'.length);

console.log(airline.indexOf('r'));
console.log(airline.lastIndexOf('r'));
console.log(airline.indexOf('Portugal'));

console.log(airline.slice(4));
console.log(airline.slice(4, 7));

console.log(airline.slice(0, airline.indexOf(' ')));
console.log(airline.slice(airline.lastIndexOf(' ') + 1));

console.log(airline.slice(-2));
console.log(airline.slice(1, -1));

const checkMiddleSeat = function (seat) {
  // B and E are middle seats
  const s = seat.slice(-1);
  if (s === 'B' || s === 'E') console.log('You got the middle seat 😍');
  else console.log('You got lucky 😊');
};

checkMiddleSeat('11B');
checkMiddleSeat('23C');
checkMiddleSeat('3E');
```

# Data Structures, Modern Operators and Strings

## Working with Strings - Part 2

```
// Working with Strings - Part 2

console.log(airline.toLowerCase());
console.log(airline.toUpperCase());

// Fix capitalization in name
const passenger = 'hEdI';
const passengerLower = passenger.toLowerCase();
const passengerCorrect =
| passengerLower[0].toUpperCase() + passengerLower.slice(1);
console.log(passengerCorrect);

// Comparing emails
const email = 'hedi@io';
const loginEmail = ' Hedi@IO \n';

// const lowerEmail = loginEmail.toLowerCase();
// const trimmedEmail = lowerEmail.trim();
// console.log(trimmedEmail);

const normalizedEmail = loginEmail.toLowerCase().trim();
console.log(normalizedEmail);

console.log(email === normalizedEmail);

// Replacing
const priceGB = '288,97£';
const priceUS = priceGB.replace('£', '$').replace(',', '.');
console.log(priceUS);

const announcement =
| 'All passengers come to boarding door 23. Boarding door 23!';
console.log(announcement.replace('door', 'gate'));
console.log(announcement.replaceAll('door', 'gate'));

// Regular Expressions
console.log(announcement.replace(/door/g, 'gate'));
```

# Data Structures, Modern Operators and Strings

## Working with Strings - Part 2

```
// Booleans
const plane1 = 'Airbus A320neo';
console.log(plane1.includes('A320'));
console.log(plane1.includes('Boeing'));
console.log(plane1.startsWith('Air'));

if (plane1.startsWith('Airbus') && plane1.endsWith('neo'))
  console.log('Part of the NEW Airbus family');

// Practice exercise
const checkBaggage = function (items) {
  const baggage = items.toLowerCase();
  if (baggage.includes('knife') || baggage.includes('gun'))
    console.log('You are NOT allowed on board');
  else console.log('Welcome aboard!');
};

checkBaggage('I have a laptop, some Food and a pocket Knife');
checkBaggage('Socks and camera');
checkBaggage('Got some snacks and a gun for protection');
```

# Data Structures, Modern Operators and Strings

## Working with Strings - Part 3

```
// Working with Strings - Part 3

// Split and Join
console.log('a+very+nice+string'.split('+'));
console.log('Hedi Rivas'.split(' '));

const [firstName, lastName] = 'Hedi Rivas'.split(' ');

const newName = ['Mr.', firstName, lastName.toUpperCase()].join(' ');
console.log(newName);

const capitalizeName = function (name) {
  const names = name.split(' ');
  const namesUpper = [];
  for (let n of names) {
    // namesUpper.push(n[0].toUpperCase() + n.slice(1));
    namesUpper.push(n.replace(n[0], n[0].toUpperCase()));
  }
  console.log(namesUpper.join(' '));
};

capitalizeName('jessica ann smith davis');
capitalizeName('hedi rivas');
```

# Data Structures, Modern Operators and Strings

## Working with Strings - Part 3

```
// Padding
const message = 'Go to gate 23!';
console.log(message.padStart(25, '+').padEnd(35, '+'));
console.log('Hedi'.padStart(25, '+'));

const maskCreditCard = function (number) {
  const str = number + '';
  const last = str.slice(-4);
  return last.padStart(str.length, '*');
};
console.log(maskCreditCard(53718463));
console.log(maskCreditCard(537281287473882839));
console.log(maskCreditCard('7472819198374647288292'));

// Repeat

const message2 = 'Bad weather... All Departures Delayed... ';
console.log(message2.repeat(5));

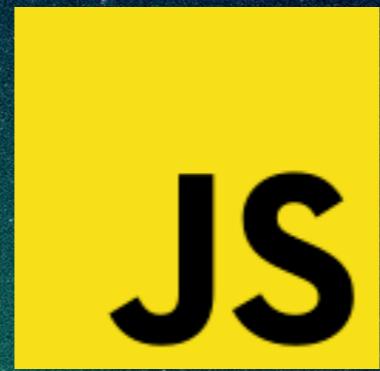
const planesInLine = function (n) {
  console.log(`There are ${n} planes in line ${'*'.repeat(n)}`);
};
planesInLine(5);
planesInLine(3);
planesInLine(12);
```

# Data Structures, Modern Operators and Strings

## CodingChallenge #4

[LIEN DU CHALLENGE](#)





# Working With Arrays

# Working With Arrays

## Simple Arrays Methods

```
let arr = ['a', 'b', 'c', 'd', 'e'];

// SLICE
console.log(arr.slice(2));
console.log(arr.slice(2, 4));
console.log(arr.slice(-2));
console.log(arr.slice(-1));
console.log(arr.slice(1, -2));
console.log(arr.slice());
console.log([...arr]);

// SPLICE
// console.log(arr.splice(2));
arr.splice(-1);
console.log(arr);
arr.splice(1, 2);
console.log(arr);

// REVERSE
arr = ['a', 'b', 'c', 'd', 'e'];
const arr2 = ['j', 'i', 'h', 'g', 'f'];
console.log(arr2.reverse());
console.log(arr2);

// CONCAT
const letters = arr.concat(arr2);
console.log(letters);
console.log([...arr, ...arr2]);

// JOIN
console.log(letters.join(' - '));
```

# Working With Arrays

## The new `.at` Method

```
// the new at Method
const arr3 = [23, 11, 64];
console.log(arr3[0]);
console.log(arr2.at(0));

// getting last array element
console.log(arr3[arr3.length - 1]);
console.log(arr3.slice(-1));
console.log(arr3.at(-1));

console.log('hedi'.at(2));
```

# Working With Arrays

## Looping Arrays: forEach

```
// looping arrays: forEach
const movements = [200, 450, -400, 3000, -650, -130, 70, 1300];

for (const [i, movement] of movements.entries()) {
  if (movement > 0) {
    console.log(`Movement ${i + 1} : You deposited ${movement}`);
  } else {
    console.log(`Movement ${i + 1} : You withdrew ${Math.abs(movement)}`);
  }
}

console.log('---- forEach ----');
movements.forEach(function (mov, i, arr) {
  if (mov > 0) {
    console.log(`Movement ${i + 1} : You deposited ${mov}`);
  } else {
    console.log(`Movement ${i + 1} : You withdrew ${Math.abs(mov)}`);
  }
});
```

# Working With Arrays

## forEach With Maps and Sets

```
// forEach with maps and sets

const currencies = new Map([
  ['USD', 'United States dollar'],
  ['EUR', 'Euro'],
  ['GBP', 'Pound sterling'],
]);
// Map
console.log('---- forEach with maps ----');
currencies.forEach(function (value, key, map) {
  console.log(` ${key}: ${value}`);
});

// Set
console.log('---- forEach with sets ----');
const currenciesUnique = new Set(['USD', 'GBP', 'USD', 'EUR', 'EUR']);
console.log(currenciesUnique);
currenciesUnique.forEach(function (value, _, map) {
  console.log(` ${value}: ${value}`);
});
```

# Project « Bankist » App

L'app



# Working With Arrays

Première chose à faire, allez dans le CSS et commentez la ligne 97.

# Working With Arrays

## Creating DOM Elements

Nous allons commencer par mettre en place le bloc HTML qui contiendra les mouvements bancaires.

Jetons un oeil au code HTML. L'objectif ici est d'itérer à travers le tableau « movements » et de créer un bloc HTML dynamiquement pour chaque données du tableau.

Pour cela, nous allons créer une fonction `displayMovements` qui prendra en paramètre `movements`, et qui permettra d'afficher sur notre site les données du tableau grâce à la méthode `insertAdjacentHTML`.

# Working With Arrays

## Creating DOM Elements

```
const displayMovements = function (movements) {
  containerMovements.innerHTML = '';

  movements.forEach(function (mov, i) {
    const type = mov > 0 ? 'deposit' : 'withdrawal';

    const html = `
      <div class="movements__row">
        <div class="movements__type movements__type--${type}"> ${i + 1}
          ${type}</div>
        <div class="movements__date">3 days ago</div>
        <div class="movements__value">${mov}€</div>
      </div>`;

    containerMovements.insertAdjacentHTML('afterbegin', html);
  });
}

displayMovements(account1.movements);
console.log(containerMovements.innerHTML);
```

# Working With Arrays

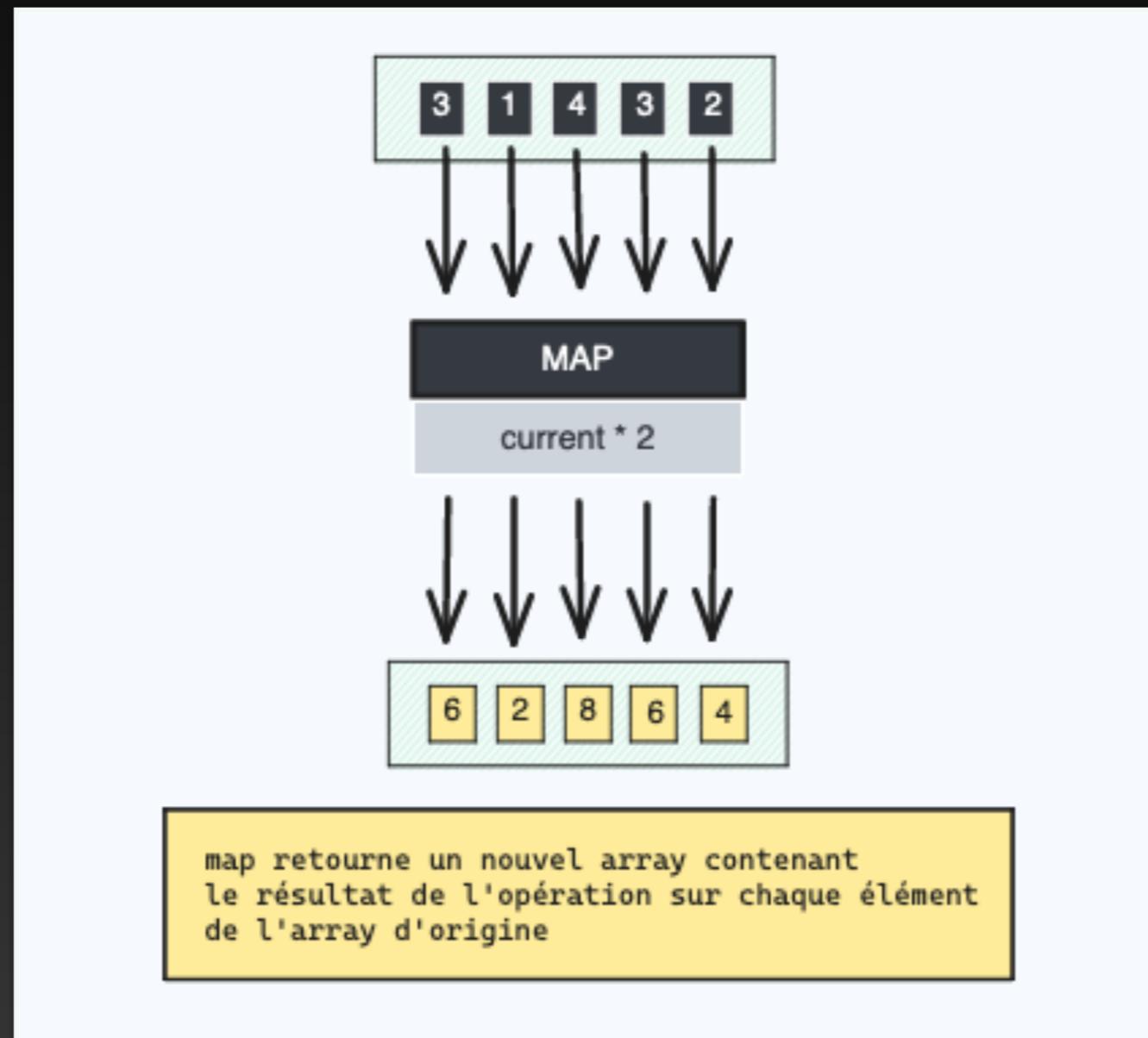
## CodingChallenge #1

[LIEN DU CHALLENGE](#)



# Working With Arrays

## The map Method



# Working With Arrays

## The map Method

```
// the map method

const eurToUsd = 1.1;
const movementsUSD = movements.map(function (mov) {
  return mov * eurToUsd;
});
console.log(movements);
console.log(movementsUSD);

const movementsUSDfor = [];
for (const mov of movements) {
  movementsUSDfor.push(mov * eurToUsd);
}
console.log(movementsUSDfor);

const movementsDescriptions = movements.map((mov, i) => {
  return `Movement ${i + 1} : You ${
    mov > 0 ? 'deposited' : 'withdrew'
  } ${Math.abs(mov)}`;
});
console.log(movementsDescriptions);
```

# Working With Arrays

## Computing Usernames - Banker App

L'objectif ici est de créer un nouveau champ dans chaque object accounts. Ce champ s'appellera username et il contiendra les initiales de chaque propriétaires de compte (owner). Nous nous servirons de ces initiales pour nous connecter au différents comptes. Par exemple pour se connecter au compte de Jessica Davis il faudra taper **jd** dans l'input user et **2222** pour l'input PIN.

Créez une fonction `createUsernames` qui prendra en paramètre « accs » et qui devra prendre en arguments le tableau **accounts**. Vous devrez utiliser les méthodes `forEach` et `map` pour faire cet exercice.

# Working With Arrays

## Computing Usernames - Banker App

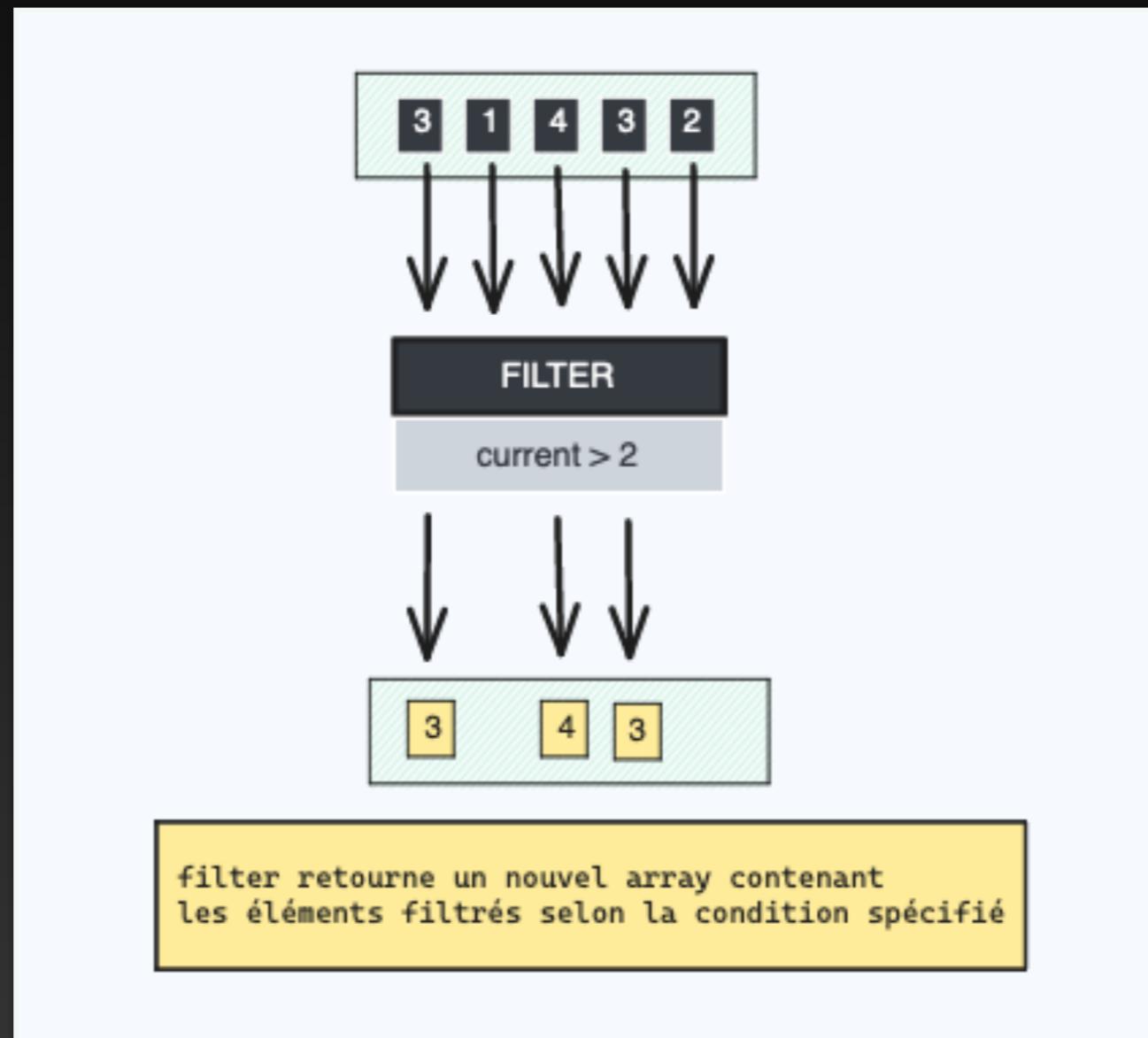
```
//Computing usernames

const createUsernames = function (accs) {
  accs.forEach(function (acc) {
    acc.username = acc.owner
      .toLowerCase()
      .split(' ')
      .map((name) => name[0])
      .join('');
  });
}

createUsernames(accounts);
console.log(accounts);
```

# Working With Arrays

## The filter Method



# Working With Arrays

## The filter Method

```
// the filter method

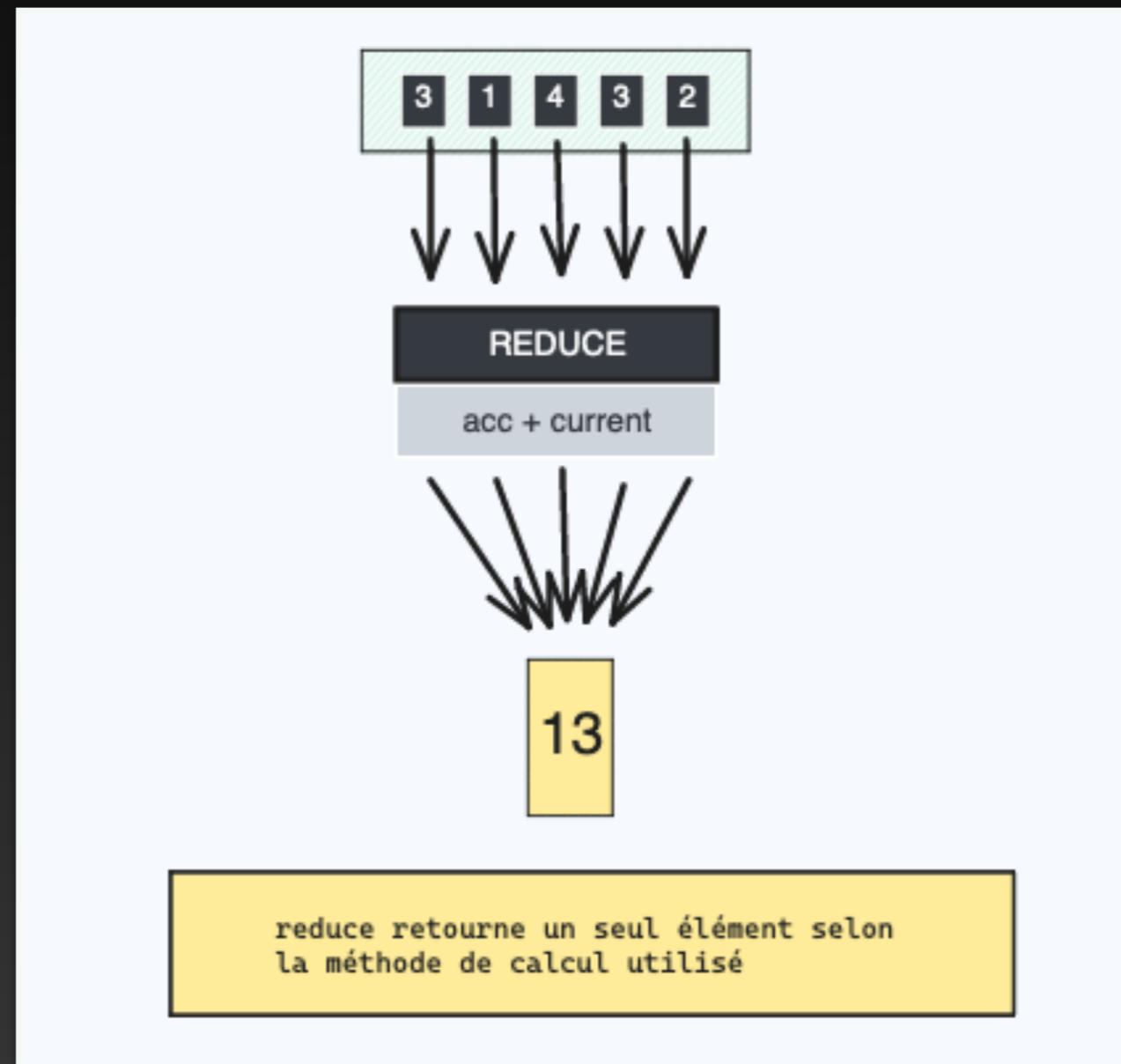
const deposits = movements.filter(function (mov) {
  return mov > 0;
});
console.log(movements);
console.log(deposits);

const depositsFor = [];
for (const mov of movements) {
  if (mov > 0) {
    depositsFor.push(mov);
  }
}
console.log(depositsFor);

const withdrawals = movements.filter(function (mov) {
  return mov < 0;
});
console.log(withdrawals);
```

# Working With Arrays

## The reduce Method



# Working With Arrays

## The reduce Method

```
// the reduce method

console.log(movements);
const balance = movements.reduce(function (acc, cur, i) {
  console.log(`Iteration ${i}: ${acc}`);
  return acc + cur;
}, 0);
console.log(balance);

let balance2 = 0;
for (const mov of movements) {
  balance2 += mov;
}
console.log(balance2);

// maximum value

const max = movements.reduce((acc, mov) => {
  if (acc > mov) return acc;
  else return mov;
}, movements[0]);
console.log('max', max);
```

# Working With Arrays

## The reduce Method - Banker App

Nous allons maintenant dynamiser le champ « Current balance » qui est actuellement à 0 €.

Créez une fonction calcDisplayBalance qui prend en paramètre « movements » et en argument le tableau movements de l'account1. Pour calculer la current balance il faudra utiliser la méthode reduce.

Une fois le total calculé grâce à la méthode reduce, affichez la valeur de la balance sur le compte.

# Working With Arrays

## The reduce Method - Banker App

```
// the reduce method

const calcDisplayBalance = function (movements) {
  const balance = movements.reduce((acc, mov) => acc + mov, 0);
  console.log(balance);
  labelBalance.textContent = `${balance} EUR`;
};

calcDisplayBalance(account1.movements);
```

# Working With Arrays

## CodingChallenge #2

[LIEN DU CHALLENGE](#)



# Working With Arrays

## The Magic of Chaining Methods

```
// the magic of chaining methods

const totalDepositsUSD = movements
  .filter(mov => mov > 0)
  .map(mov => mov * eurToUsd)
  .reduce((acc, mov) => acc + mov, 0);
console.log(totalDepositsUSD);
```

# Working With Arrays

The Magic of Chaining Methods - Banker App

Créez une fonction **calcDisplaySummary** qui prend en paramètre « movements » et qui contiendra 3 constantes : **incomes**, **out** et **interest**. Cette fonction prend `account1.movements` comme argument.

L'objectif ici est de calculer les entrées, les sorties et les intérêts. Les intérêts sont de 1,2 %. Et la banque n'accepte de verser que des intérêts supérieurs à 1€.

Utilisez les méthodes filter, reduce et/ou map pour arriver à vos fins.

La fonction **calcDisplaySummary** prendra en argument le tableau `movements` de `account1`.

# Working With Arrays

## The Magic of Chaining Methods - Banker App

```
// the magic of chaining methods

const calcDisplaySummary = function (movements) {
  const incomes = movements
    .filter((mov) => mov > 0)
    .reduce((acc, mov) => acc + mov, 0);
  labelSumIn.textContent = `${incomes}€`;

  const out = movements
    .filter((mov) => mov < 0)
    .reduce((acc, mov) => acc + mov, 0);
  labelSumOut.textContent = `${Math.abs(out)}€`;

  const interest = movements
    .filter((mov) => mov > 0)
    .map((deposit) => (deposit * 1.2) / 100)
    .filter((int, i, arr) => {
      return int >= 1;
    })
    .reduce((acc, int) => acc + int, 0);
  labelSumInterest.textContent = `${interest}€`;
};

calcDisplaySummary(account1.movements);
```

# Working With Arrays

## CodingChallenge #3

[LIEN DU CHALLENGE](#)



# Working With Arrays

## The find Method

```
// the find method

const firstWithdrawal = movements.find(mov => mov < 0);
console.log(movements);
console.log(firstWithdrawal);

console.log(accounts);
const account = accounts.find(acc => acc.owner === 'Jessica Davis');
console.log(account);
```

# Working With Arrays

## Implementing Login

Pour commencer, créez une variable `currentAccount` initialisé sans valeur.

Ensuite, en utilisant la méthode **find**, vous devrez attribuer à la variable `currentAccount` l'account qui s'est connecté. Il vous faudra pour cela comparer les initiales présentes dans le champ `username` de chaque account avec le `username` tapé dans l'input « user ». Maintenant, il faut comparer le pin du `currentAccount` avec le pin tapé dans l'input « pin ».

Si le mot de passe concorde alors on affiche un message « Welcome back, « le prénom du user »! », et on affiche les données bancaires de la personne : il faudra changer l'opacity et utiliser les fonctions créées précédemment pour mettre à jour les données selon le `currentAccount`.

# Working With Arrays

## Implementing Login

```
//Implementing login
let currentAccount;
btnLogin.addEventListener('click', (e) => {
    // Prevent form from submitting
    e.preventDefault();

    currentAccount = accounts.find(
        (acc) => acc.username === inputLoginUsername.value
    );

    if (currentAccount?.pin === Number(inputLoginPin.value)) {
        // display message UI
        labelWelcome.textContent = `Welcome back, ${currentAccount.owner.split(' ')[0]}`;
        containerApp.style.opacity = 1;

        //clear input login
        inputLoginUsername.value = '';
        inputLoginPin.value = '';
        inputLoginPin.blur();

        //display movements
        displayMovements(currentAccount.movements);

        //display balance
        calcDisplayBalance(currentAccount.movements);

        //display summary
        calcDisplaySummary(currentAccount);
    }
});
```

# Working With Arrays

## Implementing Transfers

Passons maintenant à l'implémentation de la partie « transfers ».

Créez une variable **amount** qui contiendra la valeur (en type Number) entrée dans l'inputTransferAmount.

Créez une variable **receiverAcc** dans laquelle vous utiliserez la méthode find dans le tableau accounts et qui devra vous retourné l'account correspondant au username tapé dans l'inputTransferTo.

Ensuite faites une condition qui devra vérifier si le montant du transfert est bien supérieur à 0, s'il existe bien un receveur, si la balance de l'envoyeur est supérieur au montant qu'il veut envoyer et si le username du receveur est bien différent du username de l'envoyeur.

Si toutes ces conditions sont vérifiées, le transfert peut avoir lieu et le compte peut être mis à jour.

# Working With Arrays

## Implementing Transfers

```
//Implementing transfers

btnTransfer.addEventListener('click', function (e) {
  e.preventDefault();
  const amount = Number(inputTransferAmount.value);
  const receiverAcc = accounts.find(
    (acc) => acc.username === inputTransferTo.value
  );

  if (
    amount > 0 &&
    receiverAcc &&
    currentAccount.balance >= amount &&
    receiverAcc?.username !== currentAccount.username
  ) {
    //doing the transfer
    currentAccount.movements.push(-amount);
    receiverAcc.movements.push(amount);

    //display movements
    displayMovements(currentAccount);

    //display balance
    calcDisplayBalance(currentAccount);

    //display summary
    calcDisplaySummary(currentAccount);
  }

  inputTransferAmount.value = inputTransferTo.value = '';
});
```

# Working With Arrays

## Implementing Transfers - Refactoring

Vous devriez remarquer qu'il est possible de factoriser son code en accord avec le principe DRY.

Créez une fonction `updateUI` et collez à l'intérieur le bloc qui se répète, puis appelez la fonction `updateUI` au endroit qui le nécessite.

# Working With Arrays

## Implementing Transfers - Refactoring

```
// Updating the UI
const updateUI = function (acc) {
    //display movements
    // displayMovements(currentAccount.movements);
    displayMovements(acc.movements);

    //display balance
    // calcDisplayBalance(currentAccount);
    calcDisplayBalance(acc);

    //display summary
    // calcDisplaySummary(currentAccount);
    calcDisplaySummary(acc);
};
```

# Working With Arrays

## Implementing Transfers - Refactoring

```
//Implementing transfers

btnTransfer.addEventListener('click', function (e) {
  e.preventDefault();
  const amount = Number(inputTransferAmount.value);
  const receiverAcc = accounts.find(
    (acc) => acc.username === inputTransferTo.value
  );

  if (
    amount > 0 &&
    receiverAcc &&
    currentAccount.balance >= amount &&
    receiverAcc?.username !== currentAccount.username
  ) {
    //doing the transfer
    currentAccount.movements.push(-amount);
    receiverAcc.movements.push(amount);

    //update UI
    updateUI(currentAccount);
  }

  inputTransferAmount.value = inputTransferTo.value = '';
});
```

# Working With Arrays

## Implementing Transfers - Refactoring Login Part

```
//Implementing login
btnLogin.addEventListener('click', (e) => {
    // Prevent form from submitting
    e.preventDefault();

    currentAccount = accounts.find(
        (acc) => acc.username === inputLoginUsername.value
    );

    if (currentAccount?.pin === Number(inputLoginPin.value)) {
        // display message UI
        labelWelcome.textContent = `Welcome back, ${currentAccount.owner.split(' ')[0]}`;
        containerApp.style.opacity = 1;

        //clear input login
        inputLoginUsername.value = '';
        inputLoginPin.value = '';
        inputLoginPin.blur();

        //display movements
        // displayMovements(currentAccount.movements);

        // //display balance
        // calcDisplayBalance(currentAccount);

        // //display summary
        // calcDisplaySummary(currentAccount);

        //update UI
        updateUI(currentAccount);
    }
});
```

# Working With Arrays

## The `findIndex` Method

```
// the findIndex method

console.log('movements', movements);

const index = movements.findIndex(mov => mov === -130);
console.log(index);
```

# Working With Arrays

## The `findIndex` Method - banker App

Implémentez maintenant la partie qui permet de fermer un compte.

Créez deux constantes **user** et **pin** dans lesquelles seront stockées respectivement les valeurs des inputs `inputCloseUsername` et `inputClosePin` (en type number).

Faites une condition qui devra vérifier si les constantes user et pin correspondent bien à l'utilisateur qui est connecté. Si la condition est vérifiée, créez une const **index** qui utilisera la méthode **`findIndex()`** dans le tableau **accounts** et qui devra renvoyer l'index de l'account connecté.

Ceci étant fait, il ne vous reste plus qu'à utiliser une méthode d'array qui permet de supprimer un élément sans retourner un nouveau tableau, de mettre à jour le `labelWelcome` comme à l'origine et de cacher le tableau de mouvements en utilisant l'opacité.

# Working With Arrays

## The `findIndex` Method - banker App

```
//Delete an account

btnClose.addEventListener('click', function (e) {
  e.preventDefault();

  const user = inputCloseUsername.value;
  const pin = Number(inputClosePin.value);

  if (user === currentAccount.username && pin === currentAccount.pin) {
    const index = accounts.findIndex(
      (acc) => acc.username === currentAccount.username
    );

    //delete account
    accounts.splice(index, 1);

    //hide Ui
    containerApp.style.opacity = 0;
    labelWelcome.textContent = 'Log in to get started';
  }
  inputCloseUsername.value = inputClosePin.value = '';
});
```

# Working With Arrays

some and every

```
// some and every  
  
console.log(movements);  
  
// equality  
console.log(movements.includes(-130));  
  
// condition  
console.log(movements.some((mov) => mov === -130));  
console.log(movements.every((mov) => mov > 0));
```

# Working With Arrays

some and every - Banker App

Passons à l'implémentation de la demande de prêt (loan).

Créez deux constantes, **loanAmount** et **requestedAmount**. La première constante contiendra la valeur de l'input `inputLoanAmount` (type number !) et la seconde devra vérifier qu'un dépôt d'au moins 10% de la valeur du prêt à déjà été déposé. Réfléchissez à laquelle et utilisez une des deux méthodes apprises juste avant (`some` ou `every`).

Si le montant du Loan est supérieur à 0 et si `requestedAmount` est true alors le prêt est accordé et l'account peut être mis à jour.

# Working With Arrays

some and every - Banker App

```
//Implementing loan
btnLoan.addEventListener('click', function (e) {
  e.preventDefault();

  const loanAmount = Math.floor(inputLoanAmount.value);
  const requestedAmount = currentAccount.movements.some(
    (mov) => mov >= loanAmount * 0.1
  );

  if (loanAmount > 0 && requestedAmount) {
    //add movement
    currentAccount.movements.push(loanAmount);

    //update UI
    updateUI(currentAccount);
  }

  inputLoanAmount.value = '';
});
```

# Working With Arrays

## flat and flatMap

```
const array = [[1, 2, 3], [4, 5, 6], 7, 8];
console.log(array.flat());

const arrayDeep = [[[1, 2], 3], [4, [5, 6]], 7, 8];
console.log(arrayDeep.flat(2));

// flat
const overallBalance = accounts
  .map((acc) => acc.movements)
  .flat()
  .reduce((acc, mov) => acc + mov, 0);
console.log(overallBalance);

// flatMap
const overallBalance2 = accounts
  .flatMap((acc) => acc.movements)
  .reduce((acc, mov) => acc + mov, 0);
console.log(overallBalance2);
```

# Working With Arrays

## Sorting Arrays

```
// sorting arrays

// strings
const owners = ['Hedi', 'Zied', 'Rami', 'Amine'];
console.log(owners.sort());
console.log(owners);

// numbers
console.log(movements);
console.log(movements.sort());
```

# Working With Arrays

## Sorting Arrays

```
// return < 0, A, B (keep order)
// return > 0, B, A (switch order)

// ascending
// movements.sort((a, b) => {
//   if (a > b) {
//     return 1;
//   }
//   if (b > a) {
//     return -1;
//   }
// });
// console.log(movements);

movements.sort((a, b) => a - b);
console.log(movements);

// descending
// movements.sort((a, b) => {
//   if (a > b) {
//     return -1;
//   }
//   if (b > a) {
//     return 1;
//   }
// });
// console.log(movements);

movements.sort((a, b) => b - a);
console.log(movements);
```

# Working With Arrays

## Sorting Arrays - Banker App

Implémentez l'utilisation du bouton de tri (sort).

Dans la fonction `displayMovements`, ajoutez un nouveau paramètre appelé `sort` et qui devra avoir comme valeur par défaut `false`, créez ensuite une constante qui devra contenir une ternary qui devra répondre à la logique suivante : si `sort` est `true` alors triez le tableau sinon ne le triez pas.

Tips : utilisez la méthode `slice()` à vide sur `movement` pour faire une copie du tableau et pouvoir chainer d'autre méthode directement après le `slice ;)`

# Working With Arrays

## Sorting Arrays - Banker App

Une fois avoir fait cela, créez une variable sorted initialisé à false et juste en dessous faites une écoute d'évènements au clic sur le bouton sort.

Appelez dans le block-scope de la fonction de callback la fonction displayMovements en faisant passez en premier paramètre le tableau de movements du currentAccount et en second paramètre l'inverse de la valeur de sorted. Enfin, à chaque clic, faites en sorte que la valeur de sorted passe de true à false, de false à true, etc...

# Working With Arrays

## Sorting Arrays - Banker App

```
const displayMovements = function (movements, sort = false) {
  // Clear the container
  containerMovements.innerHTML = '';

  const mows = sort ? movements.slice().sort((a, b) => a - b) : movements;

  // creating DOM element for each movement
  mows.forEach(function (mov, i) {
    const type = mov > 0 ? 'deposit' : 'withdrawal';

    const html = `
      <div class="movements__row">
        <div class="movements__type movements__type--${type}"> ${i + 1} ${type}</div>
        <div class="movements__date">3 days ago</div>
        <div class="movements__value">${mov}€</div>
      </div>`;

    containerMovements.insertAdjacentHTML('afterbegin', html);
  });
};
```

# Working With Arrays

## Sorting Arrays - Banker App

```
//Sort movements

let sorted = false;
btnSort.addEventListener('click', function () {
  displayMovements(currentAccount.movements, !sorted);
  sorted = !sorted;
});
```

# Working With Arrays

Array Methods Practice

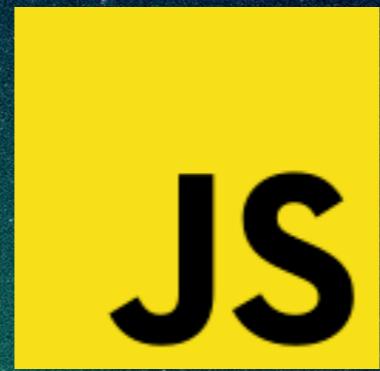
[\*\*LIEN DU CHALLENGE\*\*](#)

# Working With Arrays

## CodingChallenge #4

[LIEN DU CHALLENGE](#)





# Number, Dates, Intl and Timers

# Number, Dates, Intl and Dates

## Math and Rounding

```
// Math and rounding
console.log(Math.sqrt(25));
console.log(25 ** (1 / 2)); // 25 to the power of 1/2
console.log(8 ** (1 / 3)); // 8 to the power of 1/3

console.log(Math.max(5, 18, 23, 11, 2)); // 23
console.log(Math.max(5, 18, '23', 11, 2)); // 23
console.log(Math.max(5, 18, '23px', 11, 2)); // NaN

console.log(Math.min(5, 18, 23, 11, 2)); // 2

console.log(Math.PI * Number.parseFloat('10px') ** 2); // 314.1592653589793 (area of a circle with radius 10px)

console.log(Math.trunc(Math.random() * 6) + 1); // random number between 1 and 6

const randomInt = (min, max) =>
  Math.floor(Math.random() * (max - min) + 1) + min; // random number between min and max

console.log(randomInt(10, 20));
```

# Number, Dates, Intl and Dates

## Math and Rounding

```
// Rounding integers
console.log(Math.trunc(23.3)); // 23

console.log(Math.round(23.3)); // 23
console.log(Math.round(23.9)); // 24

console.log(Math.ceil(23.3)); // 24
console.log(Math.ceil(23.9)); // 24

console.log(Math.floor(23.3)); // 23
console.log(Math.floor('23.9'))); // 23

console.log(Math.trunc(-23.3)); // -23
console.log(Math.floor(-23.3)); // -24

// Rounding decimals
console.log((2.7).toFixed(0)); // 3 (toFixed() returns a string)
console.log((2.7).toFixed(3)); // 2.700
console.log((2.345).toFixed(2)); // 2.35
console.log(+ (2.345).toFixed(2)); // 2.35
```

# Number, Dates, Intl and Dates

## Math and Rounding - Banker App

```
//Implementing loan
btnLoan.addEventListener('click', function (e) {
  e.preventDefault();
  const loanAmount = Math.floor(inputLoanAmount.value);
  const requestedAmount = currentAccount.movements.some(
    (mov) => mov >= loanAmount * 0.1
  );

  if (loanAmount > 0 && requestedAmount) {
    //add movement
    currentAccount.movements.push(loanAmount);
  }

  //update UI
  updateUI(currentAccount);
  inputLoanAmount.value = '';
});
```

# Number, Dates, Intl and Dates

## Math and Rounding - Banker App

```
const displayMovements = function (movements, sort = false) {
    // Clear the container
    containerMovements.innerHTML = '';

    const mows = sort ? movements.slice().sort((a, b) => a - b) : movements;

    // creating DOM element for each movement
    mows.forEach(function (mov, i) {
        const type = mov > 0 ? 'deposit' : 'withdrawal';

        const html = `
<div class="movements__row">
<div class="movements__type movements__type--${type}"> ${
            i + 1
        } ${type}</div>
<div class="movements__date">3 days ago</div>
<div class="movements__value">${mov.toFixed(2)}€</div>
</div>`;

        containerMovements.insertAdjacentHTML('afterbegin', html);
    });
};
```

# Number, Dates, Intl and Dates

## The Remainder Operator

```
// The remainder operator
console.log(5 % 2); // 1
console.log(5 / 2); // 5 = 2 * 2 + 1

console.log(8 % 3); // 2
console.log(8 / 3); // 8 = 2 * 3 + 2

console.log(6 % 2); // 0
console.log(6 / 2); // 6 = 2 * 3 + 0

console.log(7 % 2); // 1
console.log(7 / 2); // 7 = 2 * 3 + 1

const isEven = (n) => n % 2 === 0;
console.log(isEven(8)); // true
console.log(isEven(23)); // false
console.log(isEven(514)); // true
```

# Number, Dates, Intl and Dates

## Numeric Separators

```
// Numeric separators
const diameter = 287_456_000_000;
console.log(diameter);

const price = 345_99;
console.log(price);

const transferFee = 15_00;
console.log(transferFee);

const transferFee2 = 1_500;
console.log(transferFee2);

console.log(Number('230_000'));
console.log(parseInt('230_000'));
```

# Number, Dates, Intl and Dates

## Working with BigInt

```
// bigInt

console.log(2 ** 53 - 1);
console.log(Number.MAX_SAFE_INTEGER);
console.log(2 ** 53 + 1);
console.log(2 ** 53 + 2);
console.log(2 ** 53 + 3);
console.log(2 ** 53 + 4);
console.log(2 ** 53 + 5);
console.log(2 ** 53 + 6);

console.log(66574738829292948756046672288n);
console.log(BigInt(665747388292288));

// Operations
console.log(10000n + 10000n);
console.log(66574738829292948756046672288n * 1000000n);
// console.log(Math.sqrt(16n)); // error

const huge = 66574738829292948756046672288n;
const num = 23;
//console.log(huge * num); // error
console.log(huge * BigInt(num));

// Exceptions
console.log(20n > 15);
console.log(20n === 20);
console.log(typeof 20n);
console.log(20n == '20');

console.log(huge + ' is REALLY big!!!');

// Divisions
console.log(10n / 3n);
console.log(10 / 3);
```

# Number, Dates, Intl and Dates

## Creating Dates

```
// Create a date
const now = new Date();
console.log(now);

console.log(new Date('Aug 02 2020 18:05:41'));
console.log(new Date('December 24, 2015'));

console.log(new Date(2037, 10, 19, 15, 23, 5));
console.log(new Date(2037, 10, 31));

console.log(new Date(0)); // Unix creation date
console.log(new Date(3 * 24 * 60 * 60 * 1000)); // 3 days after Unix creation date
```

# Number, Dates, Intl and Dates

## Creating Dates

```
// Working with dates
const future = new Date(2037, 10, 19, 15, 23);
console.log(future);

console.log(future.getFullYear());
console.log(future.getMonth()); // zero based
console.log(future.getDate());
console.log(future.getDay()); // day of the week
console.log(future.getHours());
console.log(future.getMinutes());
console.log(future.getSeconds());
console.log(future.toISOString()); // international standard
console.log(future.getTime()); // timestamp

console.log(new Date(214225698000)); //milliseconds after Unix creation date

console.log(Date.now()); // current timestamp

future.setFullYear(2040);
console.log(future); |
```

# Number, Dates, Intl and Dates

## Adding Dates to Banker App

```
//Implementing login
let currentAccount;

// fake always logged in
currentAccount = account1;
updateUI(currentAccount);
containerApp.style.opacity = 100;

const now = new Date();
const day = `${now.getDate()}`.padStart(2, 0);
const month = `${now.getMonth() + 1}`.padStart(2, 0);
const year = now.getFullYear();
const hour = now.getHours();
const minute = now.getMinutes();

labelDate.textContent = `${day}/${month}/${year}, ${hour}:${minute}`;
```

# Number, Dates, Intl and Dates

## Adding Dates to Banker App

On refactorise ici un peu le code, pour pouvoir afficher l'ensemble des données présentes dans les objets account1 et account2.

```
const displayMovements = function (acc, sort = false) {
  // Clear the container
  containerMovements.innerHTML = '';

  const mous = sort
    ? acc.movements.slice().sort((a, b) => a - b)
    : acc.movements;
```

```
let sorted = false;
btnSort.addEventListener('click', function () {
  displayMovements(currentAccount, !sorted);
  sorted = !sorted;
});
```

```
// Updating the UI
const updateUI = function (acc) {
  //display movements
  displayMovements(acc);

  //display balance
  calcDisplayBalance(acc);

  //display summary
  calcDisplaySummary(acc);
};
```

# Number, Dates, Intl and Dates

## Adding Dates to Banker App

```
const displayMovements = function (acc, sort = false) {
  // Clear the container
  containerMovements.innerHTML = '';

  const mows = sort
    ? acc.movements.slice().sort((a, b) => a - b)
    : acc.movements;

  // creating DOM element for each movement
  mows.forEach(function (mov, i) {
    const type = mov > 0 ? 'deposit' : 'withdrawal';

    const date = new Date(acc.movementsDates[i]);
    const day = `${date.getDate()}`.padStart(2, 0);
    const month = `${date.getMonth() + 1}`.padStart(2, 0);
    const year = date.getFullYear();

    const displayDate = `${day}/${month}/${year}`;

    const html = `
      <div class="movements__row">
        <div class="movements__type movements__type--${type}"> ${i + 1} ${type}</div>
        <div class="movements__date">${displayDate}</div>
        <div class="movements__value">${mov.toFixed(2)}€</div>
      </div>`;

    containerMovements.insertAdjacentHTML('afterbegin', html);
  });
}
```

# Number, Dates, Intl and Dates

## Adding Dates to Banker App

```
btnTransfer.addEventListener('click', function (e) {
  e.preventDefault();
  const amount = Number(inputTransferAmount.value);
  const receiverAcc = accounts.find(
    (acc) => acc.username === inputTransferTo.value
  );

  if (
    amount > 0 &&
    receiverAcc &&
    currentAccount.balance >= amount &&
    receiverAcc?.username !== currentAccount.username
  ) {
    //doing the transfer
    currentAccount.movements.push(-amount);
    receiverAcc.movements.push(amount);

    //add transfer date
    currentAccount.movementsDates.push(new Date().toISOString());
    receiverAcc.movementsDates.push(new Date().toISOString());

    //update UI
    updateUI(currentAccount);
  }

  inputTransferAmount.value = inputTransferTo.value = '';
});
```

# Number, Dates, Intl and Dates

## Adding Dates to Banker App

```
//Implementing loan
btnLoan.addEventListener('click', function (e) {
  e.preventDefault();

  const loanAmount = Math.floor(inputLoanAmount.value);
  const requestedAmount = currentAccount.movements.some(
    (mov) => mov >= loanAmount * 0.1
  );

  if (loanAmount > 0 && requestedAmount) {
    //add movement
    currentAccount.movements.push(loanAmount);

    //add loan date
    currentAccount.movementsDates.push(new Date().toISOString());
  }

  //update UI
  updateUI(currentAccount);
  inputLoanAmount.value = '';
});
```



# Number, Dates, Intl and Dates

## Operating With Dates

```
const formatedMovementDate = function (date) {
  const calcDatePassed = (date1, date2) =>
    Math.abs(date2 - date1) / (1000 * 60 * 60 * 24);

  const daysPassed = Math.round(calcDatePassed(new Date(), date));
  console.log(daysPassed);

  if (daysPassed < 1) return 'Today';
  if (daysPassed === 1) return 'Yesterday';
  if (daysPassed <= 7) return `${daysPassed} days ago`;
  else {
    const day = `${date.getDate()}`.padStart(2, 0);
    const month = `${date.getMonth() + 1}`.padStart(2, 0);
    const year = date.getFullYear();

    return `${day}/${month}/${year}`;
  }
};
```

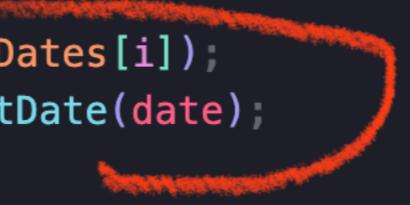
# Number, Dates, Intl and Dates

## Operating With Dates

```
const displayMovements = function (acc, sort = false) {
    // Clear the container
    containerMovements.innerHTML = '';

    const mows = sort
        ? acc.movements.slice().sort((a, b) => a - b)
        : acc.movements;

    // creating DOM element for each movement
    mows.forEach(function (mov, i) {
        const type = mov > 0 ? 'deposit' : 'withdrawal';

        const date = new Date(acc.movementsDates[i]);
        const displayDate = formatedMovementDate(date);
  

        const html = `
            <div class="movements__row">
                <div class="movements__type movements__type--${{type}}> ${
                    i + 1
                } ${type}</div>
                <div class="movements__date">${displayDate}</div>
                <div class="movements__value">${mov.toFixed(2)}€</div>
            </div>`;

        containerMovements.insertAdjacentHTML('afterbegin', html);
    });
}
```

# Number, Dates, Intl and Dates

## Internationalizing Dates (Intl)

```
// Intl API
const now = new Date();
const options = {
  hour: 'numeric',
  minute: 'numeric',
  day: 'numeric',
  month: 'long',
  // month: 'numeric',
  year: 'numeric',
  weekday: 'long',
};

const locale = navigator.language;

console.log(new Intl.DateTimeFormat('en-US', options).format(now));
console.log(new Intl.DateTimeFormat('fr-FR', options).format(now));
console.log(new Intl.DateTimeFormat(locale, options).format(now));
```

# Number, Dates, Intl and Dates

## Internationalizing Dates (Intl) - Banker App

```
btnLogin.addEventListener('click', (e) => {
  // Prevent form from submitting
  e.preventDefault();

  currentAccount = accounts.find(
    (acc) => acc.username === inputLoginUsername.value
  );

  if (currentAccount?.pin === Number(inputLoginPin.value)) {
    // display message UI
    labelWelcome.textContent = `Welcome back, ${currentAccount.owner.split(' ')[0]}`;
    containerApp.style.opacity = 1;

    // create current date and time
    const now = new Date();
    const options = {
      hour: 'numeric',
      minute: 'numeric',
      day: 'numeric',
      // month: 'long',
      month: 'numeric',
      year: 'numeric',
      // weekday: 'long',
    };

    // const locale = navigator.language;
    //console.log(locale);

    labelDate.textContent = new Intl.DateTimeFormat(
      currentAccount.locale,
      options
    ).format(now);

    // const now = new Date();
    // const day = `${now.getDate()}`.padStart(2, 0);
```



You

# Number, Dates, Intl and Dates

## Internationalizing Dates (Intl) - Banker App

```
const formatedMovementDate = function (date, locale) {
  const calcDatePassed = (date1, date2) =>
    Math.abs(date2 - date1) / (1000 * 60 * 60 * 24);

  const daysPassed = Math.round(calcDatePassed(new Date(), date));

  if (daysPassed < 1) return 'Today';
  if (daysPassed === 1) return 'Yesterday';
  if (daysPassed <= 7) return `${daysPassed} days ago`;

  // const day = `${date.getDate()}`.padStart(2, 0);
  // const month = `${date.getMonth() + 1}`.padStart(2, 0);
  // const year = date.getFullYear();
  // return `${day}/${month}/${year}`;

  return new Intl.DateTimeFormat(locale).format(date);
};
```

# Number, Dates, Intl and Dates

## Internationalizing Numbers (Intl)

```
//Formated Number with Intl

const num = 3345171.23;

const options = {
  style: 'currency', //unit, percent and currency
  unit: 'celsius',
  // unit: 'mile-per-hour',
  currency: 'EUR',
  // useGrouping: false,
};

console.log('US:      ', new Intl.NumberFormat('en-US', options).format(num));
console.log(
  'France:     ',
  new Intl.NumberFormat('fr-FR', options).format(num)
);
console.log(
  'Syria:     ',
  new Intl.NumberFormat('ar-SY', options).format(num)
);
console.log(
  navigator.language,
  new Intl.NumberFormat(navigator.language).format(num)
);
```

# Number, Dates, Intl and Dates

## Internationalizing Numbers (Intl) - Banker App

Création d'une fonction pour éviter du code répétitif

```
const formatCur = function (value, locale, currency) {
  return new Intl.NumberFormat(locale, {
    style: 'currency',
    currency: currency,
  }).format(value);
};
```

# Number, Dates, Intl and Dates

## Internationalizing Numbers (Intl) - Banker App

```
const displayMovements = function (acc, sort = false) {
  // Clear the container
  containerMovements.innerHTML = '';

  const mows = sort
    ? acc.movements.slice().sort((a, b) => a - b)
    : acc.movements;

  // creating DOM element for each movement
  mows.forEach(function (mov, i) {
    const type = mov > 0 ? 'deposit' : 'withdrawal';

    const date = new Date(acc.movementsDates[i]);
    const displayDate = formatedMovementDate(date);

    const html = `
      <div class="movements__row">
        <div class="movements__type movements__type--${type}"> ${i + 1} ${type}</div>
        <div class="movements__date">${displayDate}</div>
        <div class="movements__value">${mov.toFixed(2)}€</div>
      </div>`;

    containerMovements.insertAdjacentHTML('afterbegin', html);
  });
};
```

# Number, Dates, Intl and Dates

## Internationalizing Numbers (Intl) - Banker App

```
const calcDisplayBalance = function (acc) {
  acc.balance = acc.movements.reduce((acc, mov) => acc + mov, 0);

  labelBalance.textContent = formatCur(acc.balance, acc.locale, acc.currency);
};
```

# Number, Dates, Intl and Dates

## Internationalizing Numbers (Intl) - Banker App

```
const calcDisplaySummary = function (acc) {
  const incomes = acc.movements
    .filter((mov) => mov > 0)
    .reduce((acc, mov) => acc + mov, 0);
  labelSumIn.textContent = formatCur(incomes, acc.locale, acc.currency);

  const out = acc.movements
    .filter((mov) => mov < 0)
    .reduce((acc, mov) => acc + mov, 0);
  labelSumOut.textContent = formatCur(Math.abs(out), acc.locale, acc.currency);

  const interest = acc.movements
    .filter((mov) => mov > 0)
    .map((deposit) => (deposit * acc.interestRate) / 100)
    .filter((int, i, arr) => {
      return int >= 1;
    })
    .reduce((acc, int) => acc + int, 0);
  labelSumInterest.textContent = formatCur(interest, acc.locale, acc.currency);
};
```

# Number, Dates, Intl and Dates

## Timers: setTimeout and setInterval

```
// setTimeout
const ingredients = ['olives', 'spinach'];
// setTimeout(() => console.log('Here is your pizza'), 3000);
const pizzaTimer = setTimeout(
  (ing1, ing2) => console.log(`Here is your pizza with ${ing1} and ${ing2}`),
  3000,
  ...ingredients
);
console.log('Waiting...');

if (ingredients.includes('spinach')) clearTimeout(pizzaTimer);

//setInterval
setInterval(() => {
  const now = new Date();
  console.log(now);
}, 1000);
```

# Number, Dates, Intl and Dates

Timers: setTimeout and setInterval - Banker App

```
//Implementing loan
btnLoan.addEventListener('click', function (e) {
  e.preventDefault();

  const loanAmount = Math.floor(inputLoanAmount.value);
  const requestedAmount = currentAccount.movements.some(
    (mov) => mov >= loanAmount * 0.1
  );

  if (loanAmount > 0 && requestedAmount) {
    setTimeout(function () {
      //add movement
      currentAccount.movements.push(loanAmount);

      //add loan date
      currentAccount.movementsDates.push(new Date().toISOString());

      //update UI
      updateUI(currentAccount);
    }, 3000);
  }

  inputLoanAmount.value = '';
});
```

# Number, Dates, Intl and Dates

## Implementing a Countdown Timer

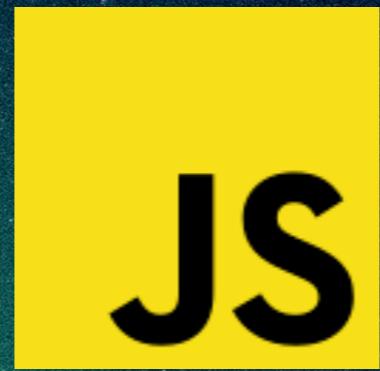
```
// Timer
const startLogOutTimer = function () {
    // set time to 5 minutes
    let time = 300;

    const tick = function () {
        const min = String(Math.trunc(time / 60)).padStart(2, 0);
        const sec = String(time % 60).padStart(2, 0);

        // in each call print the remaining time to UI
        labelTimer.textContent = `${min}:${sec}`;

        // when 0 seconds, stop timer and log out user
        if (time === 0) {
            clearInterval(timer);
            containerApp.style.opacity = 0;
            labelWelcome.textContent = 'Log in to get started';
        }
        //decrease 1s
        time--;
    };

    // call the timer every second
    tick();
    const timer = setInterval(tick, 1000);
    return timer;
};
```



# Asynchronous JavaScript

# Asynchronous JavaScript

## Asynchronous JavaScript and API's

Jusqu'à maintenant et la plupart du temps nous avons utilisé du code synchrone (synchronous) mais il y a certains cas où cela peut poser problème.

```
const p = document.querySelector('p');
alert('Hello!');
p.style.color = 'red';
```

Ici le problème est que tant que l'utilisateur n'a pas fermé l'alerte le reste du code n'est pas lu !

# Asynchronous JavaScript

## Asynchronous JavaScript and API's

Le code ci-dessous est asynchrone, en effet la fonction de callback présente dans le setTimeout ne s'exécutera que lorsque le temps configuré sera atteint. En attendant le reste du code est lu.

```
const p = document.querySelector('p');
console.log('1');
setTimeout(() => {
  p.textContent = 'Hello World!';
  console.log('2');
}, 5000);
p.style.color = 'red';
console.log('3');
```

# Asynchronous JavaScript

Welcome to Callback Hell

# Asynchronous JavaScript

Promises and fetch API

# Asynchronous JavaScript

Consuming Promises

# Asynchronous JavaScript

Chaining Promises

# Asynchronous JavaScript

Handling Rejected Promises

# Asynchronous JavaScript

Throwing Errors Manually

# Asynchronous JavaScript

## CodingChallenge #1

[LIEN DU CHALLENGE](#)



# Asynchronous JavaScript

The Event Loop in practice

# Asynchronous JavaScript

Building a Simple Promise

# Asynchronous JavaScript

Promisifying the Geolocation API

# Asynchronous JavaScript

## CodingChallenge #2

[LIEN DU CHALLENGE](#)



# Asynchronous JavaScript

Consuming Promises with Async/Await

# Asynchronous JavaScript

Error Handling With try...catch

# Asynchronous JavaScript

Returning Values from Async Functions

# Asynchronous JavaScript

Running Promises in Parallel

# Asynchronous JavaScript

## CodingChallenge #3

[LIEN DU CHALLENGE](#)





# Challenges Solutions

# Les Fondamentaux - Partie 1

## CodingChallenge Solutions

[LIEN DU CHALLENGE #1](#)

[LIEN DU CHALLENGE #2](#)

[LIEN DU CHALLENGE #3](#)

[LIEN DU CHALLENGE #4](#)

# Les Fondamentaux - Partie 2

## CodingChallenge Solutions

[LIEN DU CHALLENGE #1](#)

[LIEN DU CHALLENGE #2](#)

[LIEN DU CHALLENGE #3](#)

[LIEN DU CHALLENGE #4](#)

# Data Structures, Modern Operators and Strings

## CodingChallenge Solutions

[\*\*LIEN DU CHALLENGE #1\*\*](#)

[\*\*LIEN DU CHALLENGE #2\*\*](#)

[\*\*LIEN DU CHALLENGE #3\*\*](#)

[\*\*LIEN DU CHALLENGE #4\*\*](#)

# Working With Arrays

## CodingChallenge Solutions

[LIEN DU CHALLENGE #1](#)

[LIEN DU CHALLENGE #2](#)

[LIEN DU CHALLENGE #3](#)

[LIEN DU CHALLENGE #4](#)

Working With Arrays  
Arrays Methods Practice

# CodingChallenge Solutions

## Arrays Methods Practice

[LIEN DU CHALLENGE](#)