

Documentation de la semaine 11/11 /2024 a 20/11/2024 :

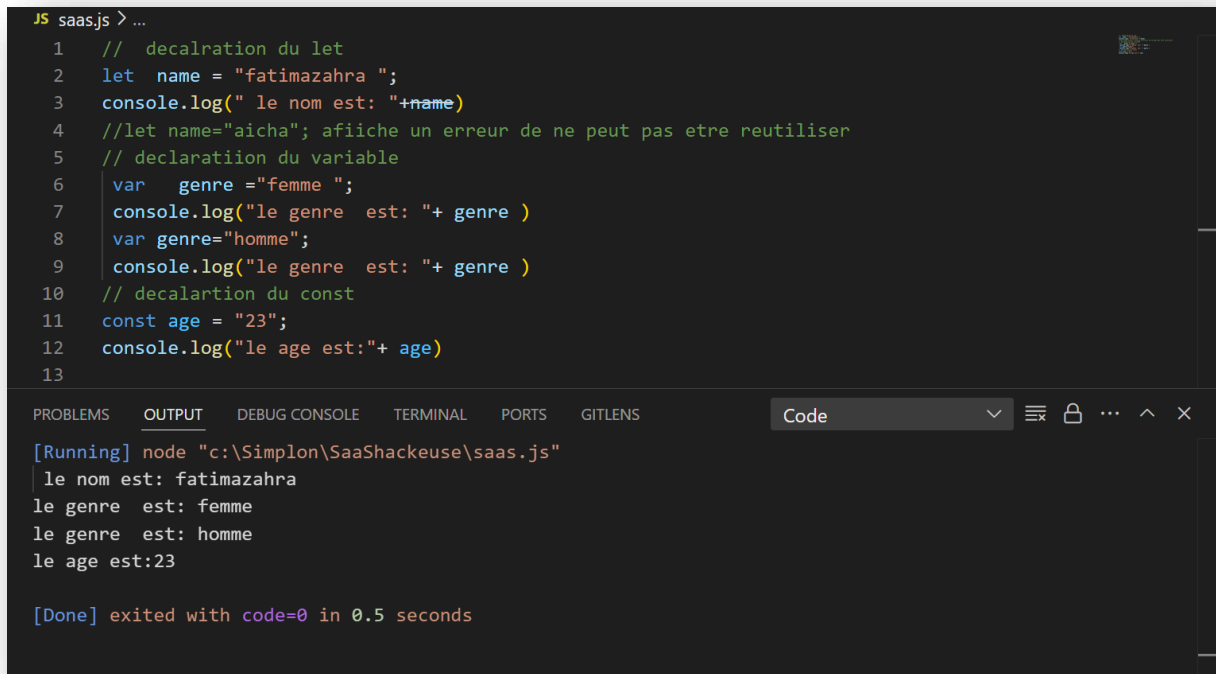
Déclaration du variable :

La déclaration du variable ce fait sur trois façon :

- Let : c'est un type de var avec un portes limite lequel été déclarer et utiliser une seul fois dans le même scope.
- Const : utiliser pour déclarer les constantes
- Var : c'est un type de var avec une portée globale déclarée en dehors d'une fonction.

Il peut être utilisés avec plusieurs valeurs dans le même porté

Exemple pratique :



```
JS saas.js > ...
1 // decalration du let
2 let name = "fatimazahra ";
3 console.log(" le nom est: "+name)
4 //let name="aicha"; afiiche un erreur de ne peut pas etre reutiliser
5 // declaratiion du variable
6 var genre ="femme ";
7 console.log("le genre est: "+ genre )
8 var genre="homme";
9 console.log("le genre est: "+ genre )
10 // decalartion du const
11 const age = "23";
12 console.log("le age est:"+ age)
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

le nom est: fatimazahra
le genre est: femme
le genre est: homme
le age est:23

[Done] exited with code=0 in 0.5 seconds

Type de données :

il existe plusieurs types de données qui peuvent être classés en deux grandes catégories : les types primitifs et les types d'objet :

🚦 Type primitifs :

- String (chaîne de caractère) : il représente un texte entourer par (") ou bien (').

```
1 let name = "fatima zahra";
2 let ville = 'Casablanca';
```

- Number : représente les entiers et les flots

```
1 let age = 23; // entier
2 let prix = 36.6; // flot
```

- Boolean : Représente une valeur (true ou false)

```
1 let etudiant = true;
2 let professeur = false;
```

- Undefined : variable déclarer son aucun valeur

```
1 let x;
2 console.log(x);
```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
[Running] node "c:\Simplon\SaaShackeuse\saas.js"				
undefined				

- Null : représente une valeur vide ou absente

```
1 let y=null;
2 console.log(y)
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
null

[Done] exited with code=0 in 0.128 seconds

- Symbol : valeur unique pour les objets

```
1 let sym = Symbol('description');
2 console.log(sym)
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
Symbol(description)

[Done] exited with code=0 in 0.185 seconds

- BigInt : représente des entiers qui dépassent la limite de nombre

```
JS saas.js > ...
1 let bigNumber = BigInt(123456789012345678901234567890);
2 console.log(bigNumber)
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
123456789012345677877719597056n

[Done] exited with code=0 in 0.119 seconds

🚧 Type objet :

- Objet : ensemble de clé valeur

```
JS saas.js > [🔍] person > [🔗] isStudent
1   let person = {
2     name: "Fatimazahra",
3     age: 23,
4     isStudent: true
5   };
6   console.log(person)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] node "c:\Simplon\SaaShackeuse\saas.js"
{ name: 'Fatimazahra', age: 23, isStudent: true }

[Done] exited with code=0 in 0.123 seconds
```

- Array : présente une liste ordonnée de valeurs

```
JS saas.js > ...
1   let fruits = ["pomme", "banane", "cerise"];
2   console.log(fruits)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] node "c:\Simplon\SaaShackeuse\saas.js"
{ name: 'Fatimazahra', age: 23, isStudent: true }

[Done] exited with code=0 in 0.123 seconds
```

Type des opérateurs :

- 🚧 les opérateurs arithmétiques : pour effectuer des opérations mathématiques

- Addition (+)

- Soustraction(-)
- Multiplication(*)
- Division(/)
- Modulo(%)
- Puissance(**)

```
JS saas.js > ...
1  let sum =5+3;
2  let dif=5-3;
3  let mul=5*3;
4  let div=5/3;
5  let mod=5%3;
6  let exp=5**3;
7  console.log(" la somme est "+sum)
8  console.log(" la dif est "+dif)
9  console.log(" la mul est "+mul)
10 console.log(" la div est "+div)
11 console.log(" la mod est "+mod)
12 console.log(" l'exp est "+exp)
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS G

```
[Running] node "c:\Simplon\SaaShackeuse\saas.js"
la somme est 8
la dif est 2
la mul est 15
la div est 1.6666666666666667
la mod est 2
l'exp est 125

[Done] exited with code=0 in 0.424 seconds
```

✚ Les opérateurs de comparaison :

- Egalité (==) : c'est non structure
- Egalité(===) : c'est structurer (vérifier aussi le type)
- Inégalité (!=)

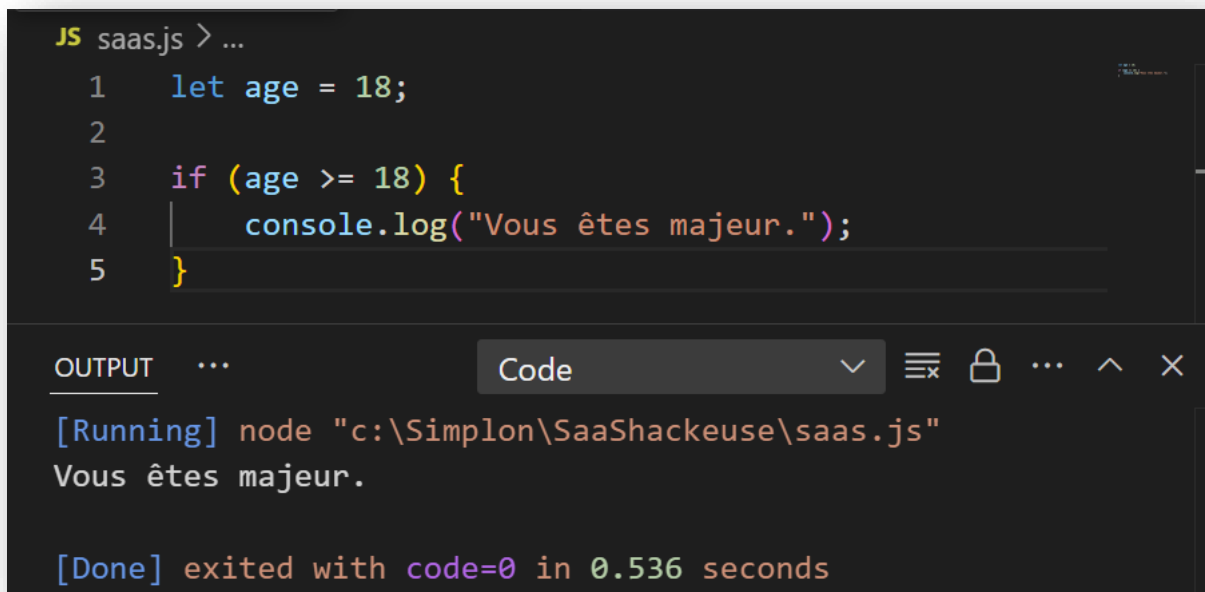
- Inégalité stricte (!=)
- Supérieur (>)
- Supérieur ou égal (>=)
- Inférieur (<)
- Inférieur ou égal (<=)

🚦 Les opérateurs logiques :

- ET logique (&&) : Renvoie true si les deux opérandes sont vrais.
- Ou logique (||) : Renvoie true si au moins un des opérandes est vrai.
- Non logique (!) : Inverse la valeur booléenne.

Les conditions :

- Instruction if : exécute un bloc si la condition est vrai :



```
JS saas.js > ...
1  let age = 18;
2
3  if (age >= 18) {
4    console.log("Vous êtes majeur.");
5  }
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

Vous êtes majeur.

[Done] exited with code=0 in 0.536 seconds

- Instruction if ... else : exécute si la condition vrai et autre bloc si faux :

```
JS saas.js > ...
1  let age = 16;
2
3  if (age >= 18) {
4      console.log("Vous êtes majeur.");
5  } else {
6      console.log("Vous êtes mineur.");
7  }
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
Vous êtes mineur.

[Done] exited with code=0 in 0.499 seconds

- Instruction if .. else if ... else : execute plusieurs condition

```
JS saas.js > ...
1  let score = 85;
2
3  if (score >= 90) {
4      console.log("Note :  très bien ");
5  } else if (score >= 80) {
6      console.log("Note : bien ");
7  } else if (score >= 70) {
8      console.log("Note : asses bien ");
9  } else {
10     console.log("Note : passable ");
11 }
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

Note : bien

[Done] exited with code=0 in 0.544 seconds

- Instruction Switch: utiliser lorsqu'il y a plusieurs conditions


```
JS saas.js > ...
1  let fruit = "banane";
2
3  switch (fruit) {
4      case "pomme":
5          console.log("C'est une pomme.");
6          break;
7      case "banane":
8          console.log("C'est une banane.");
9          break;
10     case "cerise":
11         console.log("C'est une cerise.");
12         break;
13     default:
14         console.log("Fruit inconnu.");
15 }
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
C'est une banane.

[Done] exited with code=0 in 0.357 seconds

➤ Opérateurs ternaires : de manière d'écrire if else plus petit

```
JS saas.js > ...
1  let age = 20;
2  let message = (age >= 18) ? "Vous êtes majeur." : "Vous êtes mineur.";
3  console.log(message);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

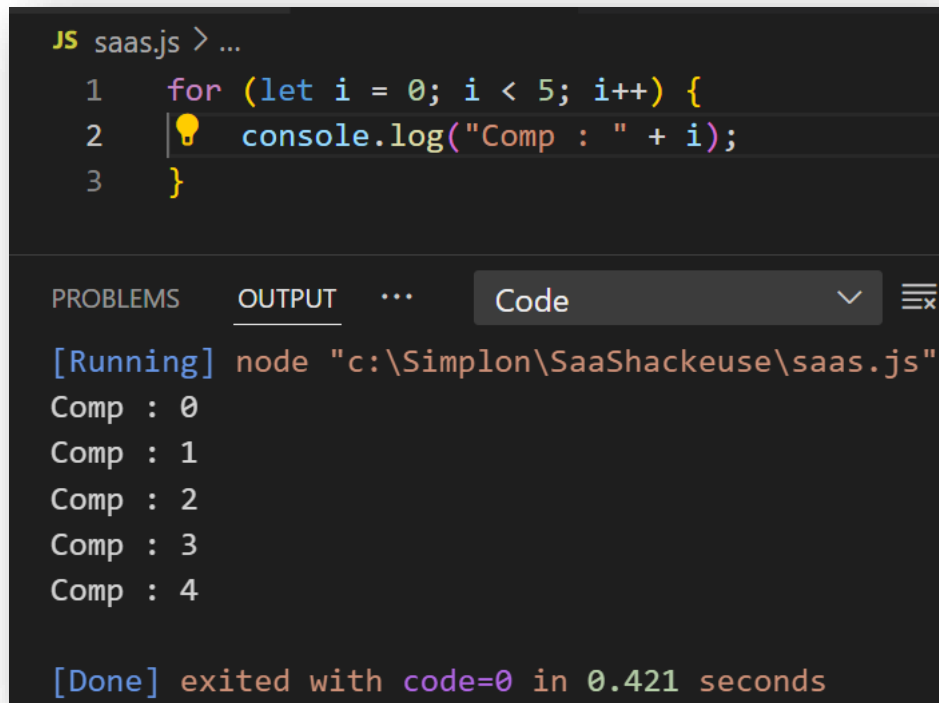
[Running] node "c:\Simplon\SaaShackeuse\saas.js"
Vous êtes majeur.

[Done] exited with code=0 in 0.402 seconds

Les boucles :

- Boucle for : pour exécuter un bloc de code un nombre spécifique de fois

Syntaxe : `for (initialisation; condition; incrément) {bloc de code à exécuter }`



```
JS saas.js > ...
1  for (let i = 0; i < 5; i++) {
2    console.log("Comp : " + i);
3  }
```

PROBLEMS OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

Comp : 0
Comp : 1
Comp : 2
Comp : 3
Comp : 4

[Done] exited with code=0 in 0.421 seconds

- boucle while : exécute un bloc de code tant qu'une condition spécifiée est vraie

Syntaxe : `while (condition) { bloc de code à exécuter }`

```
JS saas.js > ...
1  let count = 0;
2
3  while (count < 5) {
4      console.log("Compt : " + count);
5      count++;
6  }
```

OUTPUT ... Code

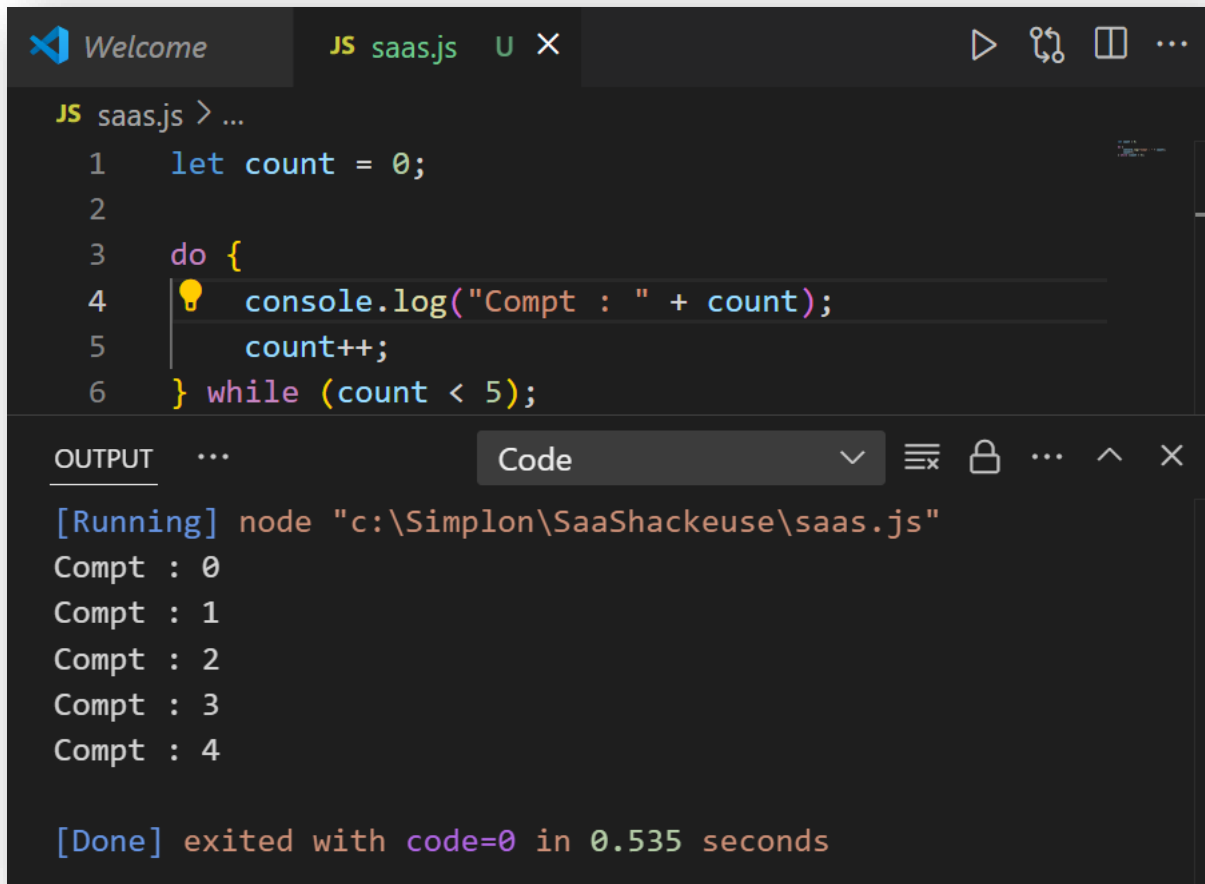
[Running] node "c:\Simplon\SaaShackeuse\saas.js"

Compt : 0
Compt : 1
Compt : 2
Compt : 3
Compt : 4

[Done] exited with code=0 in 0.45 seconds

- boucle do...while : est similaire à la boucle while, mais elle le bloc de code sera exécuté au moins une fois, même si la condition est fausse dès le départ

Syntaxe : do { bloc de code à exécuter } while (condition);



The screenshot shows a VS Code editor window with a file named `saas.js`. The code is a `do-while` loop that logs the value of `count` and increments it until it reaches 5. The output panel shows the execution results.

```
JS saas.js > ...
1 let count = 0;
2
3 do {
4     console.log("Compt : " + count);
5     count++;
6 } while (count < 5);
```

OUTPUT ... Code

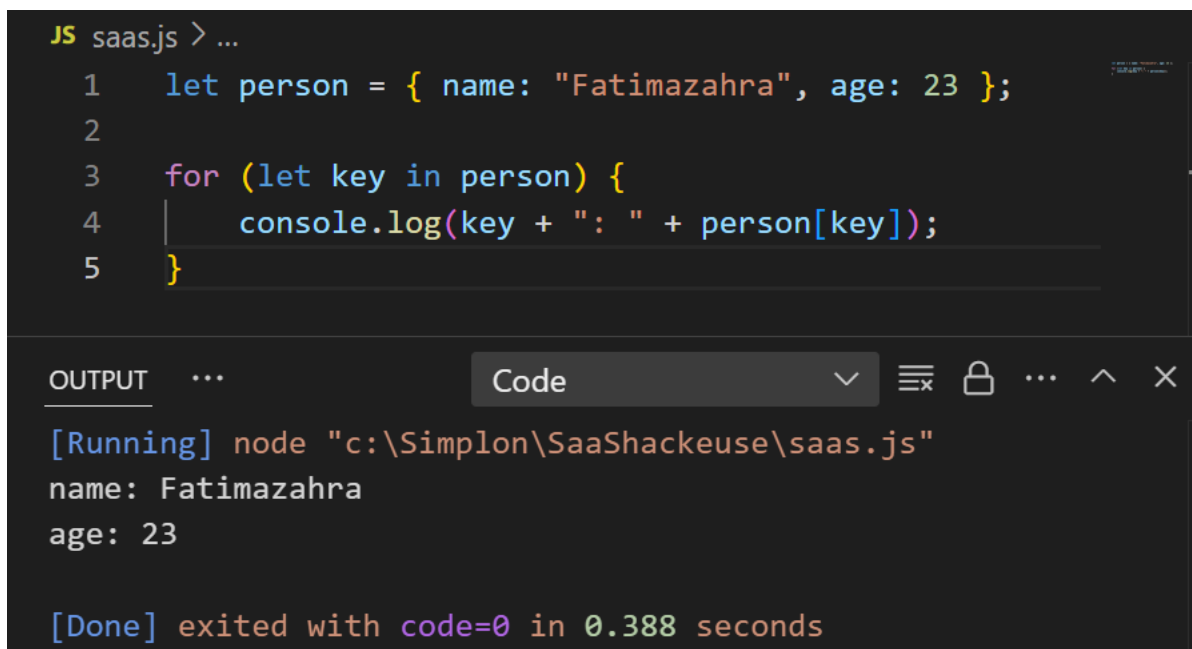
[Running] node "c:\Simplon\SaaShackeuse\saas.js"

Compt : 0
Compt : 1
Compt : 2
Compt : 3
Compt : 4

[Done] exited with code=0 in 0.535 seconds

➤ for in : pour les objets

Syntaxe : for (let key in objet) { bloc de code à exécuter }



The screenshot shows a VS Code editor window with a file named `saas.js`. The code is a `for-in` loop that iterates over the keys of an object and logs each key and its value. The output panel shows the execution results.

```
JS saas.js > ...
1 let person = { name: "Fatimazahra", age: 23 };
2
3 for (let key in person) {
4     console.log(key + ": " + person[key]);
5 }
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

name: Fatimazahra
age: 23

[Done] exited with code=0 in 0.388 seconds

➤ for of : pour tableaux et objets

Syntaxe : `for (let element of iterable) { bloc de code à exécuter }`

```
JS saas.js > ...
1  let fruits = ["pomme", "banane", "cerise"];
2
3  for (let fruit of fruits) {
4    console.log(fruit);
5  }
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

pomme
banane
cerise

[Done] exited with code=0 in 0.451 seconds

Déclaration de la fonction :

Une fonction est un bloc de code qui peut être utilisé lorsqu'il est appelé .

Syntaxe : `function nomDeLaFonction (paramètre1, paramètre2) {bloc de code à exécuter }`

```
JS saas.js > saluer
1  function saluer(nom) {
2    console.log("Bonjour, " + nom );
3  }
4
5  saluer("Fatimazahra");
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

Bonjour, Fatimazahra

[Done] exited with code=0 in 0.352 seconds

La fonction peut avoir une valeur de return :

```
JS saas.js > ...
1  const additionner = function(a, b) {
2    |    return a + b;
3  };
4
5  console.log(additionner(5, 3));
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

8

[Done] exited with code=0 in 0.474 seconds

Fonction prédéfinie de chaîne de caractère :

- length : renvoie la longueur d'une chaîne

```
JS saas.js > ...
1  let texte = "Bonjour";
2  console.log(texte.length); // Affiche : 7
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

7

- charAt(index) : renvoie le caractère à l'index spécifié

```
JS saas.js > ...
1  let texte = "Bonjour";
2  console.log(texte.charAt(0));
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

B

- indexOf(String) : renvoie l'index de la première occurrence d'une sous-chaîne.

```
JS saas.js > ...
1 let texte = "Bonjour";
2 console.log(texte.indexOf("jour"));

OUTPUT ... Code
[Running] node "c:\Simplon\SaaShackeuse\saas.js"
3
```

➤ Slice(start, end) : compter a partir de la fin de chaine

```
JS saas.js > ...
1 let texte = "Bonjour";
2 console.log(texte.slice(1, 4));
3 console.log(texte.slice(-3));

OUTPUT ... Code
[Running] node "c:\Simplon\SaaShackeuse\saas.js"
onj
our
```

➤ toLowerCase : renvoi les caractères en minuscules

```
JS saas.js > ...
1 let texte = "Bonjour";
2 console.log(texte.toLowerCase());

OUTPUT ... Code
[Running] node "c:\Simplon\SaaShackeuse\saas.js"
bonjour
```

➤ toUpperCase : renvoi les caractères en majuscules

```
JS saas.js > ...
1 let texte = "Bonjour";
2 console.log(texte.toUpperCase());

OUTPUT ... Code
[Running] node "c:\Simplon\SaaShackeuse\saas.js"
BONJOUR
```

- trim() : supprime les espaces

```
JS saas.js > ...
1 let texte = "  Bonjour  ";
2 console.log(texte.trim());
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
Bonjour

- split() : divise une chaine en tableau

```
JS saas.js > ...
1 let texte = "Bonjour, comment ça va ?";
2 let mots = texte.split(" ");
3 console.log(mots);
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
['Bonjour,', 'comment', 'ça', 'va', '?', '']]

- startWith et endsWith : vérifie si chaine commence et termine par sous – chaines

```
JS saas.js > ...
1 let texte = "Bonjour";
2 console.log(texte.startsWith("Bon"));
3 console.log(texte.endsWith("jour"));
```

OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
true
true

- replace(valeurdeja, newValue) : remplace la premier valeur par nouvelle


```
JS saas.js > ...
1 let texte = "Bonjour tout le monde";
2 let nouveauTexte = texte.replace("tout", "à tous");
3 console.log(nouveauTexte);
```

PROBLEMS OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

Bonjour à tous le monde

Les objets :

Un objet est une collection de propriétés, où chaque propriété est une paire clé-valeur. Les objets sont des structures de données essentielles qui permettent de regrouper des informations.

```
JS saas.js > [?] personne
1 let personne = {
2   nom: "Fatimazahra",
3   age: 23,
4   profession: "Développeuse",
5 };
6 console.log(personne);
```

PROBLEMS OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

{ nom: 'Fatimazahra', age: 23, profession: 'Développeuse' }

Arrays :

Un tableau (ou array) est une collection ordonnée de valeurs. Les tableaux peuvent contenir des éléments de différents types, y compris d'autres tableaux, objets.

```
JS saas.js > ...
1 let fruits = ["pomme", "banane", "cerise"];
2 console.log(fruits);
```

PROBLEMS OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

['pomme', 'banane', 'cerise']

Les fonctions prédéfinies pour Arrays et objet :

- `push()` : ajouter des éléments au tableaux

```
JS saas.js > ...
1  let fruits = ["pomme", "banane"];
2  fruits.push("cerise");
3  console.log(fruits);
```

PROBLEMS OUTPUT ... Code ▾

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
['pomme', 'banane', 'cerise']

- `pop()` : supprimer dernier élément

```
JS saas.js > ...
1  let fruits = ["pomme", "banane"];
2  let dernierFruit = fruits.pop();
3  console.log(fruits);
```

PROBLEMS OUTPUT ... Code ▾

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
['pomme']

[Done] exited with code=0 in 0.282 seconds

- `map()` : créer un nouveaux tableaux contient les résultats d'appelle

```
JS saas.js > ...
1  let fruits = ["pomme", "banane"];
2  let longueurs = fruits.map(function(fruit) {
3    |   return fruit.length;
4  | });
5  console.log(longueurs);
```

PROBLEMS OUTPUT ... Code ▾ ☰

[Running] node "c:\Simplon\SaaShackeuse\saas.js"
[5, 6]

[Done] exited with code=0 in 0.483 seconds

- `filter()` : créer un nouveau tableau avec tous les éléments qui passent le test de la fonction

```
JS saas.js > ...
1 let fruits = ["pomme", "banane"];
2 let fruitsLongs = fruits.filter(function(fruit) {
3     return fruit.length > 5;
4 });
5 console.log(fruitsLongs);
```

PROBLEMS OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

['banane']

- `object.keys()` :renvoi un tableau avec des clés d'objets

```
JS saas.js > ...
1 let personne = { nom: "Ali", age: 30 };
2 console.log(Object.keys(personne));
```

PROBLEMS OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

['nom', 'age']

- `object.values()` :renvoie un tableau avec les valeurs d'un objet

```
JS saas.js > ...
1 let personne = { nom: "Ali", age: 30 };
2 console.log(Object.values(personne));
```

PROBLEMS OUTPUT ... Code

[Running] node "c:\Simplon\SaaShackeuse\saas.js"

['Ali', 30]

Git et GitHub :

Git est l'outil qui gère les versions de votre code, tandis que **GitHub** est un service en ligne qui vous permet de stocker et de collaborer sur des projets Git.

- Les commandes de Git :

- `git init`
- `git clone [url]`
- `git branch [nom]`

- `git add [fichiers]`
- `git commit -m "message"`
- `git push`