Les opérations

Table des matières

I. Contexte

Durée : 45 min

Environnement de travail : Repl.it

Pré-requis : Bases de JavaScript

Contexte

Lors du développement d'un programme, il est souvent nécessaire d'effectuer certains traitements sur les différentes valeurs manipulées. Nous allons voir quelles sont les opérations les plus souvent utilisées en JavaScript et comment s'en servir, ainsi que des méthodes pour effectuer des opérations plus complexes, sur des tableaux ou des objets.

II. Opérateurs arithmétiques

Objectif

- Connaître les opérateurs arithmétiques les plus communs et savoir les utiliser

Mise en situation

Dans un cours précédent, nous avons vu que les variables servaient à stocker des données afin de les réutiliser et les manipuler par la suite. Dans le cas des données de type numérique (`integer`, `float`), différentes opérations sont possibles par le biais des opérateurs arithmétiques. Nous allons voir à quoi ils servent et comment les utiliser.

Les opérateurs arithmétiques standards

Les opérateurs arithmétiques utilisent des valeurs numériques comme opérandes, sous forme de littéraux ou de variables, et renvoient une valeur numérique. Les plus utilisés sont l'addition, la soustraction, la multiplication et la division.

Les opérateurs arithmétiques standards (+, -, * et /) fonctionnent de la même manière que toute opération classique.

```
1  const maVariableNumerique = 10;
2
3  const addition = maVariableNumerique + 5;
4  // addition vaudra 15
5
6  const soustraction = maVariableNumerique - 5;
7  // soustraction vaudra 5
8
9
10 const multiplication = maVariableNumerique * 5;
11 // multiplication vaudra 50
12
13
14 const division = maVariableNumerique / 5;
15 // division vaudra 2
```

Remarque

En plus des opérateurs arithmétiques standards, JavaScript fournit quelques opérateurs arithmétiques supplémentaires, comme le modulo (`%`), l'incrémentation (`++`), la décrémentation (`--`), le plus et la négation unaire (`+` et `-`).

Le modulo

Le modulo (`%`) est un opérateur qui renvoie le reste entier de la division entre deux opérandes. Ce reste est toujours plus petit que la valeur du diviseur.

```
const monNumerateur = 63;
const monDenominateur = 5;

const modulo = monNumerateur % monDenominateur;
// modulo vaudra 3
```

```
let modulo = 63 % 5;
// modulo vaudra 3

modulo = -63 % 5;
// modulo vaudra -3

modulo = 63 % -5;
// modulo vaudra 3

modulo = -63 % -5;
// modulo vaudra -3
```

L'incrémentation et la décrémentation

L'incrémentation permet d'ajouter 1 à l'opérande et renvoie une valeur numérique. Si l'opérateur est utilisé en suffixe (`variable++`), il renvoie la valeur avant l'incrémentation. S'il est utilisé en préfixe (`++variable`), il renvoie la valeur après l'incrémentation.

```
let maVariable = 5;

const incrementationSuffixe = maVariable++;
// incrementationSuffixe vaudra 5 et maVariable vaudra 6

const incrementationPrefixe = ++maVariable;
// incrementationPrefixe vaudra 6 et maVariable vaudra 6
```

La décrémentation permet de soustraire 1 à l'opérande et renvoie une valeur numérique. Si l'opérateur est utilisé en suffixe (`variable--`), il renvoie la valeur avant la décrémentation. S'il est utilisé en préfixe (`--variable`), il renvoie la valeur après la décrémentation.

```
let maVariable = 12;

const decrementationSuffixe = maVariable--;
// decrementationSuffixe vaudra 12 et maVariable vaudra 11

const decrementationPrefixe = --maVariable;
// decrementationPrefixe vaudra 11 et maVariable vaudra 11
```

Le plus unaire et la négation unaire

Le plus unaire est situé avant l'opérande et retourne l'opérande, converti en nombre si cela est possible et que ce n'en est pas déjà un. Il permet donc de convertir des chaînes de caractères composées uniquement de nombres entiers ou flottants, ainsi que des booléens et la valeur `null`. Si l'opération ne peut pas être effectuée, la valeur retournée sera `NaN`.

NaN représente la valeur « Not a Number ». Cette propriété indique que la valeur n'est pas un nombre « légal » ou, plus simplement, la valeur n'est pas un nombre. Il est rare de l'utiliser dans un programme.

Généralement, on va le récupérer à la suite d'une fonction mathématique qui a échoué ou quand une fonction qui tente d'interpréter un nombre échoue (exemple : parseFloat(« coucou »)).

```
+12;
// Sera évalué à 12
+"12"
// Sera évalué à 12
+true
// Sera évalué à 1
+false
// Sera évalué à 0
+null
// Sera évalué à 0
+"chaîne de caractères"
// Sera évalué à NaN
```

Le moins unaire est situé avant l'opérande et retourne l'opposé de l'opérande, après l'avoir converti en nombre si cela est possible et nécessaire.

```
let maVariable = 12;
-maVariable;
// Sera évalué à -12

maVariable = -12;
-maVariable;
// Sera évalué à 12

-"12"
// Sera évalué à -12

-true
// Sera évalué à -1

-false
// Sera évalué à 0

-null
// Sera évalué à 0

-"chaîne de caractères"
// Sera évalué à NaN
```

Syntaxe **À retenir**

- Nous avons vu comment procéder à des opérations standards telles que les additions, soustractions, multiplications et divisions, mais aussi comment incrémenter et décrémenter une valeur, et obtenir le reste entier d'une division grâce au modulo.
- Ces opérations sont les plus courantes en JavaScript et seront donc utiles dans de nombreuses situations.

Complément

Opérateurs arithmétiques

III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question 1

[solution n°1 p.13]

À combien est égal le résultat du code suivant ?

```
let myNumber = 10
myNumber  = myNumber * 3
myNumber = myNumber + 2
```

Question 2

[solution n°2 p.13]

Que peut-on déduire de la variable '`myNumber`' si :

```
myNumber%2 = 0;
```

Question 3

[solution n°3 p.13]

À combien est égale la variable `myOtherVariable` ?

```
let myVariable = 10;
let myOtherVariable = myVariable++;
myOtherVariable--;
```

IV. Opérateurs d'affectation

Objectif

- Connaître les différents opérateurs d'affectation et leur fonctionnement

Mise en situation

Dans un cours précédent, nous avons vu ce qu'étaient des variables et à quoi elles pouvaient servir, ainsi qu'une des manières de leur affecter une valeur. Nous allons maintenant apprendre d'autres façons de les utiliser.

Remarque

Les opérateurs d'affectation assignent des valeurs basées sur l'opérande droit à leur opérande gauche.

Affectation simple

Afin d'assigner une valeur à une variable, nous pouvons utiliser l'opérateur d'affectation simple (=). Le résultat de cette affectation sera la valeur affectée. Il est possible de chaîner plusieurs opérateurs d'affectation afin d'assigner la même valeur à plusieurs variables : c'est ce qu'on appelle une assignation par transitivité.

1 https://repl.it/

```
let maVariable = 5;
let monAutreVariable = 12;
const maDerniereVariable = 50;

maVariable = monAutreVariable;
// maVariable vaut maintenant 12

maVariable = monAutreVariable = maDerniereVariable;
// maVariable, monAutreVariable et maDerniereVariable valent maintenant 50

```

Remarque

On peut aussi affecter une valeur après différentes opérations, comme après une addition, une soustraction, une multiplication...

Affectation après addition

L'affectation après une addition permet d'ajouter la valeur de l'opérande droit à la variable définie à gauche. Le résultat de l'opération est affecté à la variable en question.

L'opérateur aura un comportement différent selon le type des deux opérandes. On pourra donc obtenir une concaténation dans le cas de chaînes de caractères, et une addition dans le cas de nombres.

```
let maVariableNombre = 10;
let monAutreVariableNombre = 50;
let maVariableString = "ma chaîne de caractères";
const monAutreVariableString = "ma deuxième chaîne de caractères";
const maVariableBooleen = true;

maVariableNombre += monAutreVariableNombre;
// Nombre + nombre réalise une addition, le résultat sera 60

monAutreVariableNombre += maVariableString;
// Nombre + chaîne de caractères réalise une concaténation, le résultat sera "50ma chaîne de caractères"

maVariableString+= maVariableBooleen;
// Chaîne de caractère + booléen réalise une concaténation, le résultat sera "ma chaîne de caractèrestrue"

maVariableString+= monAutreVariableString;
// Chaîne de caractères + chaîne de caractères réalise une concaténation, le résultat sera "ma chaîne de caractèresma deuxième chaîne de caractères"
```

Affectation après soustraction

L'affectation après une soustraction permet de soustraire la valeur de l'opérande droit à la variable définie à gauche. Le résultat de l'opération est affecté à la variable en question. Pour tout type différent de `number`, l'opération retournera "`NaN`".

```
let maVariableNombre = 10;
const monAutreVariableNombre = 50;
const maVariableString = "ma chaîne de caractère";

maVariableNombre -= monAutreVariableNombre;
// Nombre - nombre, le résultat sera -40

maVariableNombre -= maVariableString;
```

```
// NaN, l'affectation après soustraction retourne NaN pour toute opération sur des types différents de nombres
```

Affectation après division

L'affectation après division renvoie le quotient de l'opérande gauche comme numérateur, et de l'opérande droit comme dénominateur. Pour tout type différent de `number`, l'opération retournera "`NaN`".

```
let monNumerateur = 10;
const monDenominateur = 5;
const maVariableString = "ma chaîne de caractère";

monNumerateur /= monDenominateur;
// monNumerateur vaudra 2

monNumerateur /= maVariableString;
// monNumerateur vaudra NaN
```

Affectation après multiplication

L'affectation après multiplication permet de multiplier une variable par la valeur de l'opérande droit et d'affecter le résultat de cette opération à la variable. Pour tout type différent de `number`, l'opération retournera "`NaN`".

```
let maVariableNombre = 10;
const monAutreVariableNombre = 5;
const maVariableString = "ma chaîne de caractère";

maVariableNombre *= monAutreVariableNombre;
// monNumerateur vaudra 50

maVariableNombre *= maVariableString;
// monNumerateur vaudra NaN
```

Affectation après modulo

L'opérateur modulo permet de diviser une valeur par la valeur de l'opérande droit, et d'affecter le reste entier de cette division à la variable.

```
let maVariableNombre = 78;
const monAutreVariableNombre = 4;
const maVariableString = "ma chaîne de caractère";


maVariableNombre %= monAutreVariableNombre;
// maVariableNombre vaudra 2

maVariableNombre %= maVariableString;
// maVariableNombre vaudra NaN
```

Syntaxe **À retenir**

- Nous connaissons maintenant des méthodes d'affectation de valeur à une variable autres que l'affectation simple : l'affectation après addition, soustraction, division, multiplication, et le modulo.

Complément

Opérateurs d'affectation

V. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°4 p.13]

Quel sera le résultat de `myVariable` ?

```
let myVariable = 0;
const number = 12;
const myText = " ma première affectation";
const mySecondNumber = 8;

myVariable += number ;
myVariable -= mySecondNumber ;
myVariable += myText;
```

Indice :

Il faut faire la différence entre une opération mathématique et une concaténation.

VI. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°5 p.13]

Exercice

De quel type est `myVariable` ?

```
let myVariable = '12';
myVariable = +myVariable ;
console.log(myVariable )
```

- ○ string
- ○ number
- ○ NaN
- ○ null

Exercice

1 https://repl.it/

Que retournera la console ?

```
let myVariable = 'ma premiere variable';
myVariable = +myVariable;
console.log(myVariable);
```

- 'ma première variable'
- 10
- NaN

Exercice

Quel sera la résultat en console ?

```
const myNumber = 12;
const mySecondeNumber = 4;
const myText = "Le résultat de l'opération ";
const mySecondText = " est ";
console.log(myText + myNumber + " - " + mySecondeNumber + mySecondText + (myNumber - mySecondeNumber));
```

Attention aux espaces !

Exercice

Quelles affirmations retourneront un type `string` ou `number` en console?

- [ ] const myVariable = "Nicolas a 6" - 3 ;
  console.log(myVariable) ;
- [ ] const myVariable = "Nicolas a 6" + 3 ;
  console.log(myVariable) ;
- [ ] const myVariable = "Nicolas a " + (6 - 3) ;
  console.log(myVariable) ;
- [ ] const myVariable = "6 " - 3 ;
  console.log(myVariable) ;

Exercice

Quel sera la résultat en console ?

```
let myVariable = 10;
myVariable++;
myVariable= myVariable* 2;
myVariable--;
myVariable%= 3
```

Exercice

Dans quel cas peut-on exprimer avec certitude que `myNumber` est impair ?

- console.log(myNumber % 3 === 0) // true
- console.log(myNumber % 0 === 0) // false
- console.log(myNumber % 2 !== 0) // true
- console.log(myNumber % 1 === 0) // true

Exercice

Que retournera la console ?

```
// ...code
const {type} = computer;
const {color} = computer;
const {size} = computer;
console.log(typeof computer);
```

B. Exercice : Défi

Pierre, Jeanne et Paul jouent aux billes pendant la récréation.

Pierre a 10 billes, Jeanne en a 15 et Paul, 5.

À chaque partie, le joueur gagnant prend la bille de ses adversaires.

Pierre gagne les deux premières parties.

Jeanne gagne la troisième.

Paul la quatrième, et Jeanne la cinquième.

Jeremy arrive avec un sac plein de calots et propose à chacun d'échanger ses calots contre des billes, avec chacun un arrangement différent.

Pour Pierre, 1 calot = 4 billes.

Pour Jeanne, 1 calot = 2 billes.

Pour Paul, 1 calot = 3 billes.

Pierre, Paul et Jeanne décident d'échanger le maximum de billes avec Jeremy.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°6 p.15]

Combien restera-t-il de billes et de calots à Pierre, Paul et Jeanne à la fin de la récréation ?

Indice :

Pensez à l'opérateur `%`.

Solutions des exercices

1 https://repl.it/

p. 6 Solution n°1

32

p. 6 Solution n°2

`myNumber` est pair. En effet, si le modulo 2 d'un nombre est égal à 0, cela signifie que la division de ce nombre par 2 est égale à 0.

p. 6 Solution n°3

9

p. 9 Solution n°4

4 ma première affectation

Exercice p. 9 Solution n°5

Exercice

De quel type est `myVariable` ?

```
let myVariable = '12';
myVariable = +myVariable ;
console.log(myVariable )
```

- ○ string
  `myVariable est` définie en tant que `string` au début du code. `+myVariable` la transforme en nombre.
- ◉ number
- ○ NaN
- ○ null

Exercice

Que retournera la console ?

```
let myVariable = 'ma premiere variable';
myVariable = +myVariable;
console.log(myVariable);
```

- ○ 'ma première variable'
  Un nombre ne peut pas être un `string`.
- ○ 10
  Une chaîne de caractères ne peut pas être de type `number`.
- ◉ NaN
  On ne peut pas transformer une chaîne de caractère en nombre de cette manière.

Exercice

Quel sera la résultat en console ?

```
const myNumber = 12;
const mySecondeNumber = 4;
const myText = "Le résultat de l'opération ";
const mySecondText = " est ";
console.log(myText + myNumber + " - " + mySecondeNumber + mySecondText + (myNumber - mySecondeNumber));
```

Attention aux espaces !

Le résultat de l'opération 12 - 4 est 8

L'opérateur +, entre une chaîne de caractères et un chiffre, concaténera.

Exercice

Quelles affirmations retourneront un type `string` ou `number` en console?

- [ ] const myVariable = "Nicolas a 6" - 3 ;
  console.log(myVariable) ;
  Retournera `NaN` (not a number). `NaN` apparaît lorsque le code s'attend à un nombre, mais que celui-ci retourne autre chose.
  Dans le cas présent, on ne peut pas soustraire une chaîne de caractères et un nombre.

- [x] const myVariable = "Nicolas a 6" + 3 ;
  console.log(myVariable) ;
  Nicolas a 63.

- [x] const myVariable = "Nicolas a " + (6 - 3) ;
  console.log(myVariable) ;
  Nicolas a 3.

- [x] const myVariable = "6 " - 3 ;
  console.log(myVariable) ;
  3, JavaScript reconnaît les chiffres, même de type `string`. Il va automatiquement transformer le `string` en `number` pour réaliser l'opération.

Exercice

Quel sera la résultat en console ?

```
let myVariable = 10;
myVariable++;
myVariable= myVariable* 2;
myVariable--;
myVariable%= 3
```

0

```
let myVariable = 10;
myVariable++;
// 10 + 1 = 11
myVariable = myVariable* 2;
// 11 * 2 = 22
myVariable--;
// 22 - 1 = 21
myVariable%= 3
```

```
9 // 21%3 = 0
```

Exercice

Dans quel cas peut-on exprimer avec certitude que `myNumber` est impair ?

- ○ console.log(myNumber % 3 === 0) // true
  Un nombre divisible par 3 peut être pair. 30%3 = 30%2 = 0
- ○ console.log(myNumber % 0 === 0) // false
  Un nombre ne peut pas être divisé par 0.
- ◉ console.log(myNumber % 2 !== 0) // true
  Tout nombre qui n'est pas divisible par 2 est forcément impair.
- ○ console.log(myNumber % 1 === 0) // true
  Un nombre divisé par 1 ne peut pas avoir de reste. Le résultat sera toujours égal à 0, qu'il soit pair ou impair.

Exercice

Que retournera la console ?

```
1 // ...code
2 const {type} = computer;
3 const {color} = computer;
4 const {size} = computer;
5 console.log(typeof computer);
```

object

On utilise ici la déstructuration pour affecter les propriétés de `computer` à des variables. `computer` est donc de type `object`.

p. 11 Solution n°6

```
1  let billesPierre = 10;
2  let billesJeanne = 15;
3  let billesPaul = 5;
4
5  // partie 1 et 2
6  billesPierre += 4;
7  billesJeanne -= 2;
8  billesPaul -= 2;
9  // partie 3 et 5
10 billesJeanne += 4;
11 billesPierre -= 2;
12 billesPaul -= 2;
13 // partie 4
14 billesPaul += 2;
15 billesPierre--;
16 billesJeanne--;
17
18 // l'échange avec Pierre
19 const billesRestantesPierre = billesPierre % 4;
20 const boulardsPierre = (billesPierre - billesRestantesPierre) / 4;
21
22 console.log(billesRestantesPierre); // 3
23 console.log(boulardsPierre); // 2
```

```
24
25 // l'échange avec Jeanne
26 const billesRestantesjeanne = billesJeanne % 2;
27 const boulardsjeanne = (billesJeanne - billesRestantesjeanne) / 2;
28
29 console.log(billesRestantesjeanne); // 0
30 console.log(boulardsjeanne); // 8
31
32 // l'échange avec Paul
33 const billesRestantesPaul = billesPaul % 3;
34 const boulardsPaul = (billesPaul - billesRestantesPaul) / 3;
35
36 console.log(billesRestantesPaul); // 0
37 console.log(boulardsPaul); // 1
```