Javascript

partie 2

---

1) Rappel, bonne pratique :

Pour déclarer une variable ou functions, nous pouvons utiliser l'instruction new.
L'opérateur new permet de créer une instance d'un certain type d'objet à partir du constructeur qui existe pour celui-ci (natif ou défini par l'utilisateur).

```
const str = new String('un string');
```

Cependant, toujours privilégier la synthase littérale

```
const str = new String('some string');
const str = 'some string'; // synthase littérale


const age = new Number(26);
const age = 26; // synthase littérale


const person = new Object();
const person = {}; // synthase littérale


const x = new Boolean(false);
const x = false; // synthase littérale


const add = new Function('a', 'b', 'return a + b;');
const add = function (a, b) {
  return a + b;
}; // synthase littérale
```

Garde l'opérateur new pour l'appel des instances à partir de la function constructor.

```
function User(name) { // toujours nommer la class qui appelle le
                          constructor en majuscule
          constructor{
               this.name = name;
               this.isAdmin = false;
                    }
               }
let user = new User("Jack");
alert(user.name); // Jack
alert(user.isAdmin); // false
```

A partir de ES6, la function constructor est appelée avec la synthase suivante

```
class User { // toujours nommer la class qui appelle le
                constructor en majuscule
          constructor(name){
               this.name = name;
               this.isAdmin = false;
                    }
               }
let user = new User("Jack");
alert(user.name); // Jack
alert(user.isAdmin); // false
```

2) Les opérateurs de comparaison et logiques

Opérateurs de comparaison

Opérateur	Description	Exemples qui renvoient **true**

Égalité (==)	Renvoie true si les opérandes sont égaux après conversion en valeurs de mêmes types.	`3 == var1`
		`"3" == var1`
		
		`3 == '3'`

Inégalité (!=)	Renvoie true si les opérandes sont différents.	`var1 != 4` `var2 != "3"`
Égalité stricte (===)	Renvoie true si les opérandes sont égaux et de même type. Voir `Object.is()` et égalité de type en JavaScript.	`3 === var1`
Inégalité stricte (!==)	Renvoie true si les opérandes ne sont pas égaux ou s'ils ne sont pas de même type.	`var1 !== "3"` `3 !== '3'`
Supériorité stricte (>)	Renvoie true si l'opérande gauche est supérieur (strictement) à l'opérande droit.	`var2 > var1` `"12" > 2`
Supériorité ou égalité (>=)	Renvoie true si l'opérande gauche est supérieur ou égal à l'opérande droit.	`var2 >= var1` `var1 >= 3`
Infériorité stricte (<)	Renvoie true si l'opérande gauche est inférieur (strictement) à l'opérande droit.	`var1 < var2` `"2" < "12"`
Infériorité ou égalité (<=)	Renvoie true si l'opérande gauche est inférieur ou égal à l'opérande droit.	`var1 <= var2` `var2 <= 5`

Opérateurs logiques

Opérateur	Usage	Description
ET logique (&&)	*expr1* &&*expr2*	Renvoie `expr1` si cette expression peut être convertie en `false`, sinon renvoie `expr2`. Donc, lorsqu'il est utilisé avec des valeurs booléennes, && renvoie `true` si les deux opérandes peuvent être converties en `true`, sinon il renvoie `false`.
OU logique (||)	*expr1* ||*expr2*	Renvoie `expr1` si cette expression peut être convertie en `true`, sinon renvoie `expr2`. Donc, lorsqu'il est utilisé avec des valeurs booléennes, || renvoie `true` si au moins un des deux opérandes peut être convertie en `true`. Si les deux valent `false`, il renvoie également `false`.
NON logique (!)	!*expr*	Renvoie `false` si son opérande unique peut être converti en `true`, sinon il renvoie `true`.

Exemples :

```
let a = 3, b = 2, c = 1, d = 4;

if ( (a > b) && (c > d)) {
    alert ("A est plus grand que B et C est plus grand que D");
}


let score;
if (!score){ // score est undefined donc sa valeur booléenne = false
    score = 10 // on assigne une valeur à score
}
alert(score);


let score = 20;
if (!score){ // score est une variable number valable donc sa valeur booléenne =
true
    score = 10 // on assigne une valeur à score
}
alert(score); // cependant la condition ci-dessus n'est pas remplie, donc la
valeur de score reste 20
```

3) Les structures conditionnelles

IF / ELSE IF / ELSE (« si…sinon si…sinon »)**:**

```
if (condition) {  //évaluée à true ou false.
   Instruction1 } //L'instruction qui est exécutée si la condition est vérifiée
else {
   Instruction2 //l'instruction2 est exécutée si la condition est évaluée
à false. L'expression else n'est pas obligatoire.
}
```

Exemple :

```
const score = 10 ;
if( score === 10 ){
    alert("Oui, la condition a été remplie.");
}
else{
    alert("Non, la condition n'a pas été remplie.");
}
```

La condition **if…else if…else** est une structure conditionnelle encore plus complète que la condition **if…else** puisqu'elle va nous permettre cette fois-ci de générer et de prendre en charge autant de cas que l'on souhaite.

```
if ( condition1 ) {
//premier bloc de code, sera exécuté si la condition est vraie
} else if ( condition2 ) {
//bloc de code exécuté si la condition1 échoue et que la condition2 est vraie
} else {
//bloc de code à exécuter si la condition1 et la condition2 sont fausses
}
```

Exemple :

```
if( score === 10 ){
    alert("Oui, la condition a été remplie.");
}
else if( score === 11 ){
    alert("Oui, la seconde condition a été remplie.");
}
else{
    alert("Non, la condition n'a pas été remplie.");
}
```

SWITCH

En plus de **if ... else**, JavaScript comporte une instruction appelée **switch**. **switch** est un type d'instruction conditionnelle qui évaluera une expression par rapport à plusieurs cas possibles et exécutera un ou plusieurs blocs de code en fonction de la véracité de ces conditions.

```
let score = 12;

switch(score) {
    case 10:
        alert("Oui, la première condition a été remplie"); break;
    case 11:
        alert("Oui, la seconde condition a été remplie"); break;
    case 12:
        alert("Oui, la troisième condition a été remplie"); break;
    default:
        alert("Non, la condition n'a pas été remplie");
}
```

Exemples :

```
// Exemple de l'instruction switch avec plusieurs cases qui exécutent la même instruction

let reponse = window.prompt("Tapez blanc, rouge ou jaune puis clickez su OK")

switch (reponse.toLowerCase()) {
    case "blanc":
    case "rouge":
        alert("Vous avez tapé rouge ou blanc."); break;
    case "jaune":
        alert("Vous avez tapé jaune."); break;
    default:
        alert("Désolé, cette couleur n'est pas disponible.");
}

// Exemple de switch avec des opérateurs de comparaison

let reponse = window.prompt("tapez quelque chose, puis cliquer sur OK.")

switch (true) {
    case reponse.length < 3 :
        alert("La longeur de votre reponse est inférieur à trois"); break;
    case reponse.length === 3 :
        alert("La longeur de votre reponse est égale à trois"); break;
    default:
        alert("La longeur de votre reponse est supérieur à trois");
}
```

CONDITIONS TERNAIRES

Les structures ternaires correspondent à une autre façon d'écrire nos conditions en utilisant une syntaxe basée sur l'opérateur ternaire ? : qui est un opérateur de comparaison. Cet opérateur est fréquemment utilisé comme raccourci de if...else.

```
condition ? expressionSiVrai : expressionSiFaux
```

- **condition**
  Une expression qui est évaluée en un booléen (true ou false).
- **expressionSiVrai**
  Une expression qui est évaluée si la condition est équivalente à true (truthy)
- **expressionSiFaux**
  Une expression qui est évaluée si la condition est équivalente à false (falsy).

Exemples :

```
let a = 10, b = 12;

let resultat = a > b ? "A est supérieur à B" : "B est supérieur à A";
alert(resultat);
```

// Exemple d'opérateur ternaire imbriqué.

```
let a = 10, b = 10;
let resultat = a === b ? "A est égale à B" : (a > b ? "A est supérieur à B" : "B est supérieur à A");
alert(resultat);
```

```
/*
   Si  a === b →  A est égale à B
   Sinon →
     a > b →  A est supérieur à B
     Sinon →  B est supérieur à A
*/
```