Javascript

Nombres, chaînes et booléens

Valeurs primitives ou objets ?

Ce sont des valeurs primitives que nous définissons

```
let age = 25;
typeof age; //number

let nom = "toto";
typeof nom; //string

let bool = true;
typeof bool; //boolean

let obj = {};
typeof obj; //object
```

Valeurs primitives ou objets ?

Mais on peut appeler des méthodes comme s'il s'agissait d'objets

```
let age = 25;
age.toFixed(2); //"25.00"

let nom = "toto";
nom.toUpperCase(); //"TOTO"
```

Javascript effectue automatiquement les conversions

Valeurs primitives ou objets ?

Les constructeurs `String, Boolean, Number` existent, mais on ne les utilise pas pour créer des chaînes, booléens et des nombres. C'est inutile et source d'erreurs.

```
//sans les conversions automatiques, il faudrait écrire
let age = 25;
let objTmp = new Number(age);
let str = objTmp.toFixed(2);
objTmp = null;

//en pratique on écrit
let age = 25;
let str = age.toFixed(2);
```

Valeurs primitives ou objets ?

En revanche l'invocation des constructeurs `String`, `Boolean`, `Number` sans le mot clef `new` est utile pour convertir l'argument en valeur primitive correspondante.

```
Number("25.5"); //25.5
Number("25.5Ko"); //NaN
Number(true); //1
Number(false); //0

String(25.5); //"25.5"
String(true); //"true"
String(false); //"false"

Boolean("toto"); //true
Boolean(""); //false
Boolean(25); //true
Boolean(0); //false
```

Méthodes usuelles

Chaînes

- `at` : renvoie le caractère à la position spécifiée
- `includes` : teste si une chaîne est contenue dans une autre
- `indexOf` : renvoie l'indice de la 1ère occurence
- `lastIndexOf` : renvoie l'indice de la dernière occurence
- `replace` : renvoie la chaîne modifiée à l'aide d'une expression rationnelle et un texte de remplacement
- `search` : recherche grâce à une expression rationnelle
- `slice` : extrait une section de la chaine de caractères
- `split` : divise la chaîne en tableau de sous-chaînes
- `toLowerCase` : renvoie la chaîne en minuscules
- `toUpperCase` : renvoie la chaîne en majuscules
- `trim` : renvoie la chaîne en retirant les blancs en début et fin

➡ Référence complète

Méthodes usuelles

Nombres

- `toFixed` : renvoie une chaîne de caractères avec un nombre spécifié de chiffres après la virgule
- `toPrecision` : renvoie une chaîne de caractères avec un nombre spécifié de chiffres significatifs

```
let num = 2.4567887899;
num.toFixed(2); // "2.46"
num.toPrecision(2); // "2.5"

num = 0.05;
num.toFixed(3); // "0.050"
num.toPrecision(3); // "0.0500"
```

➡ Référence complète

Méthodes usuelles

Toutes ces méthodes ne modifient pas la variable elle-même mais renvoient une nouvelle valeur primitive.

```
let nom = "Toto";
nom.toUpperCase();
console.log(nom); //Toto

let nomModifie = nom.toUpperCase();
console.log( nomModifie ); //TOTO
```

Quelques fonctions globales

Chaînes

- `decodeURI` : décode une URL
- `decodeURIComponent` : décode un paramètre d'URL
- `encodeURI` : encode une URL
- `encodeURIComponent` : encode un paramètre d'URL

```
encodeURI("http://monDomaine.fr/mon article/?param=toto");
//"http://monDomaine.fr/mon%20article/?param=toto"

encodeURIComponent("toto&tata?titi");
//"toto%26tata%3Ftiti"
```

Quelques fonctions

Nombres

- `parseInt` : convertit une variable en nombre entier
- `parseFloat` : convertit une variable en nombre décimal
- `isFinite` : détermine si l'argument est un nombre fini
- `isNaN` : détermine si l'argument est `NaN` (not a number)

```
Number.parseInt("25.5°C"); //25 les décimaux sont tronqués
Number.parseFloat("25.5°C"); //25.5
Number("25.5°C"); //NaN

Number.isFinite(2/3); //true
Number.isFinite(Infinity); //false
Number.isFinite(NaN); //false

Number.isNaN(3/0); //false
Number.isNaN( parseInt("a") ); //true
```

Quelques fonctions

Nombres

Pour des raisons historiques, ces fonctions sont aussi disponibles dans l'espace global.

```
parseInt("25.5°C"); //25 les décimaux sont tronqués
parseFloat("25.5°C"); //25.5
isFinite(2/3);
isNaN( Number.parseInt("a") ); //true
```

Modèles de libellés

(gabarits de chaînes, template strings)

Ce sont les chaînes définies par les guillemets obliques qui permettent d'y inclure des expressions javascript.

```
let nom = "Toto";
`Mon nom est ${nom}`; // Mon nom est Toto

let age = 32;
`Mon âge sera ${ age+1 } ans` //Mon âge sera 33 ans
```

Modèles balisés
(tagged templates)

Les modèles de libellés peuvent être traités par une fonction.

```
function ajoutUnites(chaines, t, rr) {
  let str0 = chaines[0]; //"Il fait "
  let str1 = chaines[1]; //" et il a plu "
  return `${ chaines[0] }${t}°C${ chaines[1] }${rr}mm`
}

let t = 30;
let rr = 0.8;

ajoutUnites`Il fait ${t} et il a plu ${rr}`;
/*Il fait 30°C et il a plu 0.8mm*/
```