

JavaScript

- Chaînes de caractère -



Chaînes de caractères en JavaScript

- JavaScript possède un type primitif pour modéliser les chaînes de caractères : **string**
- Il n'existe pas de type pour modéliser un caractère
- Les guillemets simples ' ou doubles " permettent tous deux d'initialiser une chaîne (pas de différence de comportement)
- Exemples :

```
let s1 = 'guillemets simples';  
let s2 = "guillemets doubles";  
console.log(typeof s1); // => string  
console.log(typeof s2); // => string
```

Différences avec les chaînes en C

- Les chaînes sont **immuables**. Cela signifie qu'une fois défini, leur contenu est figé.
- Les chaînes encapsulant leur taille, il n'a pas de délimiteur '\0' en fin de chaîne
- Certains opérateurs (comme +, [] et ==) sont redéfinis pour les chaînes, ce qui en simplifie la manipulation
- Les (dés)allocations de mémoire sont automatiquement effectuées lors de traitements

Opérations de base

- Taille : `s.length` donne la taille de la chaîne de caractères `s`
- Accès au caractère : `s[i]` donne le (i+1)ème caractère de `s` (les indices commencent par 0)
- Exemple :

```
let s = "bonjour";  
  
console.log(s.length); // => 7  
console.log(s[3]); // => j
```

```
console.log("texte en dur".length); // => 12  
console.log("autre texte en dur"[1]); // => u
```

- Concaténation : en utilisant +
 - Dès qu'une expression avec des + contient une chaîne, l'ensemble est converti en chaîne

- Exemples :

```
let s = "i contient ";  
let i = 42;
```

```
console.log(s + i);    // => i contient 42
```

```
let s = "3";  
let i = 1;
```

```
console.log(s + i);    // => 31 (et non 4)
```

- Concaténation : en utilisant **les gabarits**
 - Les gabarits sont des littéraux délimités par ` (backtick), contenant des emplacements (placeholders) de la forme **`${}`**

- Exemples :

```
let nbErrors = 42;  
let s = `Ce code contient ${nbErrors} erreur(s)`;  
console.log(s); // => Ce code contient 42 erreur(s)
```

```
let x = 42;  
let s = `${Math.abs(x)} est égal à ${Math.sqrt(x**2)}`;
```

```
let content = "Contenu du paragraphe";  
let s =  
  `

<p> ${content} </p>  
  </div>`;


```

- Comparaison :
 - ==
 - ===
 - <, >, <=, >= (les caractères sont comparés un par un dans l'ordre lexicographique)
- Exemple :

```
let s1 = "opérations ";  
let s2 = "de base";  
  
console.log(s1 + s2);    // => opérations de base  
console.log(s1 > s2);    // => true (car "o" > "d")  
console.log(s1 == s2);   // => false  
console.log(s1 === s1);  // => true
```

Méthodes de manipulation

- Les objets de type String contiennent de nombreuses méthodes permettant de les manipuler. La liste complète est fournie [MDN](#)
- A la différence d'une fonction, pour appliquer une méthode f() sur une chaîne s, il faut écrire **s.f()** et non f(s).

- Exemples :

```
console.log ( "bonjour".toUpperCase () ); // => BONJOUR  
console.log ( "au revoir".substring (1, 6) ); // => u rev
```


- Toute opération sur une chaîne de caractères renvoie une nouvelle chaîne, sans modifier l'original.

- Exemples :

```
let s = "bonjour";  
s.toUpperCase();  
console.log(s);      // => bonjour
```

```
let s1 = "bonjour";  
let s2 = s1.toUpperCase();  
console.log(s2);      // => BONJOUR
```

- L'opérateur `[]` fonctionne en lecture seule :

```
let s = "bonjour";  
s[5] = "x";          // aucun effet*  
console.log(s);      // => bonjour
```

Conversions chaînes <-> nombres

- **1ère solution** : L'objet Number contient une méthode toString() qui renvoie la représentation d'un nombre sous la forme d'une chaîne de caractères

```
let i = 1;  
let s = i.toString(); // s contient "1"
```

- **2ème solution** :

```
let i = 1;  
let s = "" + i; // s contient "1"
```

Conversions chaînes <-> nombres

- Les fonctions `parseFloat()` et `parseInt()` permettent d'extraire et convertir des nombres contenus dans des chaînes de caractères
- Ces fonctions convertissent le contenu tant qu'elles rencontrent des caractères valides. Dès qu'un caractère invalide est détecté, le reste de la chaîne est ignoré (mais aucune erreur n'est levée)
- Dans `parseFloat()`, le caractère "." est interprété comme une virgule, alors qu'il sera invalide pour dans `parseInt()`
- Exemples :

```
console.log(parseInt("35.4"));           // => 35  
console.log(parseInt("123toto"));        // => 123  
console.log(parseFloat("35.4"));        // => 35.4  
console.log(parseFloat("123toto"));      // => 123
```

```
console.log(parseInt("toto123"));        // => NaN  
console.log(isNaN(parseFloat("toto123"))); // => true
```