Chaînes de caractères en JavaScript

Philippe Genoud
Philippe.Genoud@univ-grenoble-alpes.fr

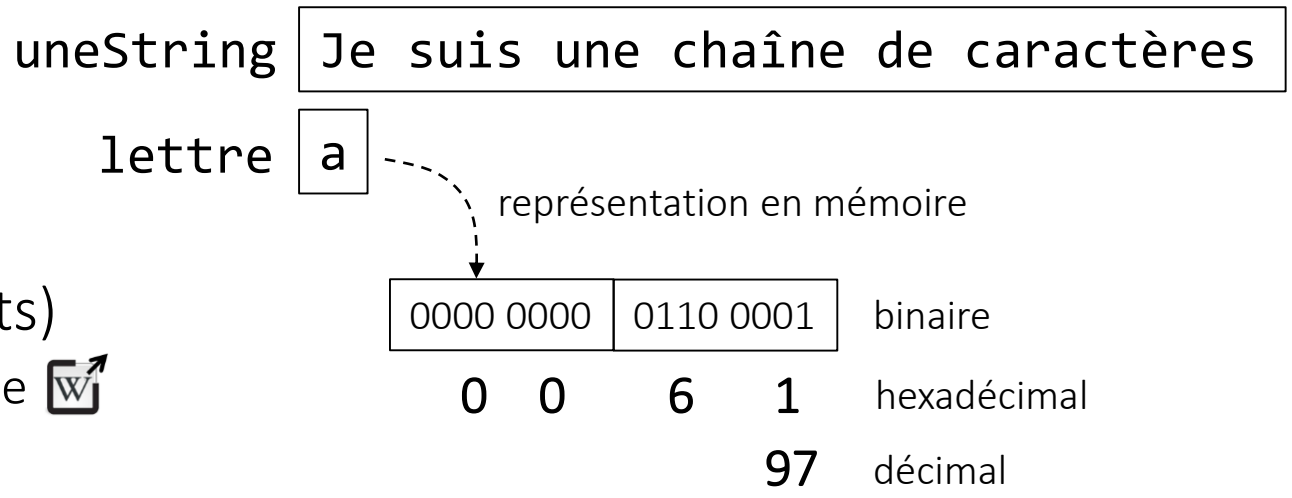
`string` : un type primitif

- chaîne de caractères stockées dans des variables de type primitif `string`
  - pas de type séparé pour stocker un simple caractère

```
let uneString = "Je suis une chaîne de caractères";
let lettre = 'a';

console.log(typeof uneString);    ----> string
console.log(typeof lettre);       ----> string
```

type primitif ➔
la valeur de la variable est la chaîne de caractères



- format interne UTF-16 (2 ou 4 octets)
  - ceci quelque soit l'encodage de la page

Expressions littérales

- trois types de délimiteurs
  - `"` : double quotes

```
let string1 = "Une chaîne de caractères";
```

  - `'` : simple quotes

```
let string2 = 'Une chaîne de caractères';
```

  - `` ` `` : backticks

```
let string3 = `Une chaîne de caractères`;
```

    - présentent plusieurs avantages par rapport aux simple ou double quotes

possibilité d'insérer n'importe quelle expression
à l'aide de **`${...}`**

```
function sum3(a, b, c) {
   return a + b + c;
}

console.log(`1 € + 2 € + 3 € = ${sum3(1,2,3)} €`);
```

```
----> 1 € + 2 € + 3 € = 6 €
```

permet d'avoir un chaîne sur plusieurs lignes

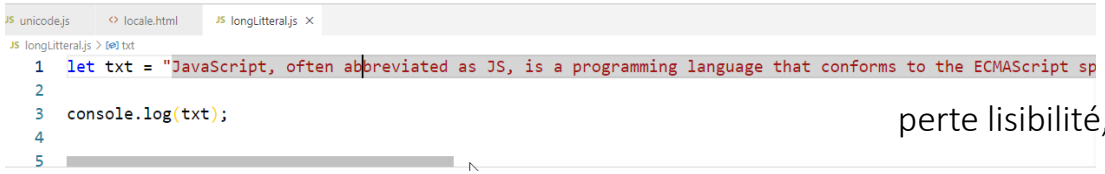
```
let calcul = ` 1 €
+ 2 €
+ 3 €
-----
 ${sum3(1,2,3)} €
`;

console.log(calcul);
```

```
---->
  1 €
+ 2 €
+ 3 €
-----
  6 €
```

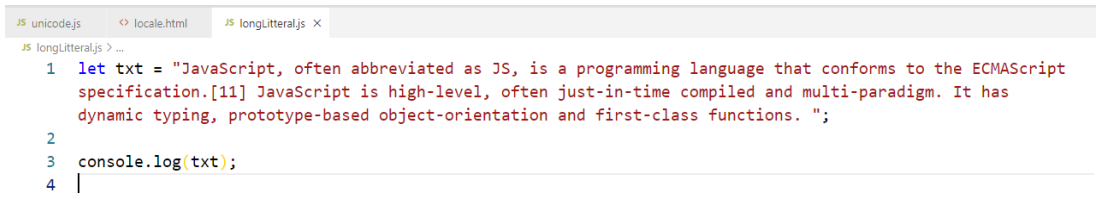
Expressions littérales

- parfois nécessité d'utiliser de longues chaînes littérales dans le code



perte lisibilité, scrolling

utilisation de l'auto wrapping, les lignes s'enroulent au gré de votre éditeur




```
let txt = "JavaScript, often abbreviated as JS, is a programming language that " +
          "conforms to the ECMAScript specification. JavaScript is high-level, " +
          "often just-in-time compiled and multi-paradigm. It has dynamic typing, " +
          "prototype-based object-orientation and first-class functions. ";

console.log(txt);
```

utilisation de l'opérateur de concaténation (**+**)
pour combiner plusieurs lignes entre elles

```
let txt = "JavaScript, often abbreviated as JS, is a programming language that \
conforms to the ECMAScript specification. JavaScript is high-level, \
often just-in-time compiled and multi-paradigm. It has dynamic typing, \
prototype-based object-orientation and first-class functions. ";

console.log(txt);
```

utilisation d'un backslash en fin de ligne (****) pour indiquer que la ligne continue sur la ligne suivante

pas de texte après le ****
pas d'indentation sur les lignes suivantes

Caractères spéciaux

```
let calcul = ` 1 €
+ 2 €
+ 3 €
-----
 ${sum3(1,2,3)} €
`;

console.log(calcul);   ---->
```

```
  1 €
+ 2 €
+ 3 €
-----
  6 €
```

Possibilité d'écrire des chaînes multi lignes délimitées par simple ou double quote en utilisant le caractère *new line* : **\n**

```
console.log(" 1 €\n+2 €\n+3 €\n-----\n  " +
    sum3(1,2,3) + " €\n";
```

- \ (backslash) permet d'introduire les caractères spéciaux

  principaux caractères spéciaux

Caractère	Description
\n	Retour à la ligne
\t	tabulation
\' \"	permet d'échapper (*escape*) les quotes 'L\'élite de ce pays permet de faire et défaire les modes, suivant la maxime qui proclame: "Je pense, donc tu suis."' *Pierre Desproges*
\xXX	caractère Unicode dont le code hexadécimal est XX '\xA9' ---> ©
\uXXXX	caractère Unicode dont le code hexadécimal est XXXX (codage UTF-16) '\u26C4' ---> ☃
\u{X…X}	caractère Unicode dont le codage hexadécimal est X…X (1 à 6 chiffres hexadécimaux) nécessite plus de 2 octets '\u{1F60D}' ---> 😍

les `strings` comme des objets

```
let s1 = "Hello";
```

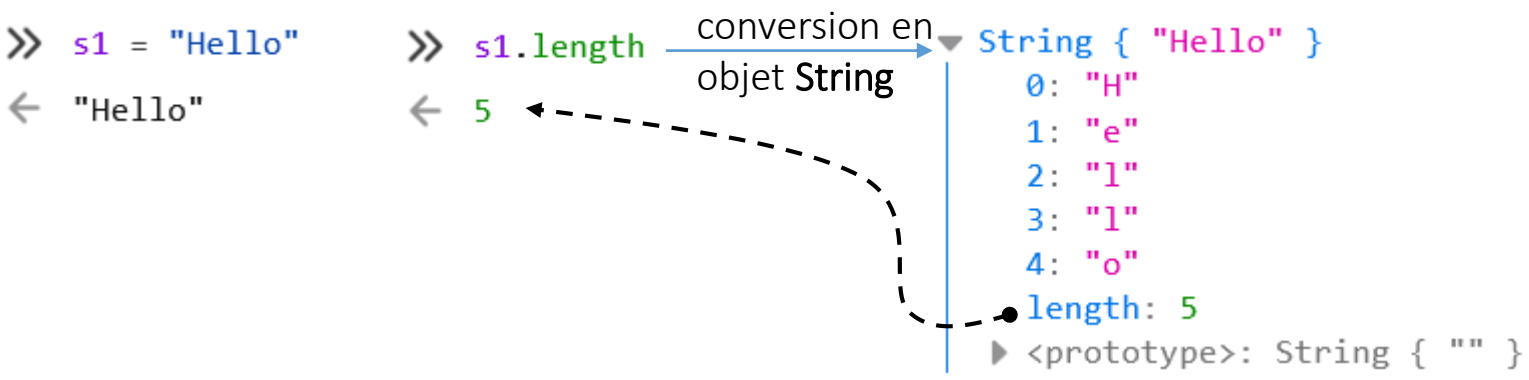
type primitif ➔ la valeur de la variable est la chaîne de caractères `s1` `Hello`

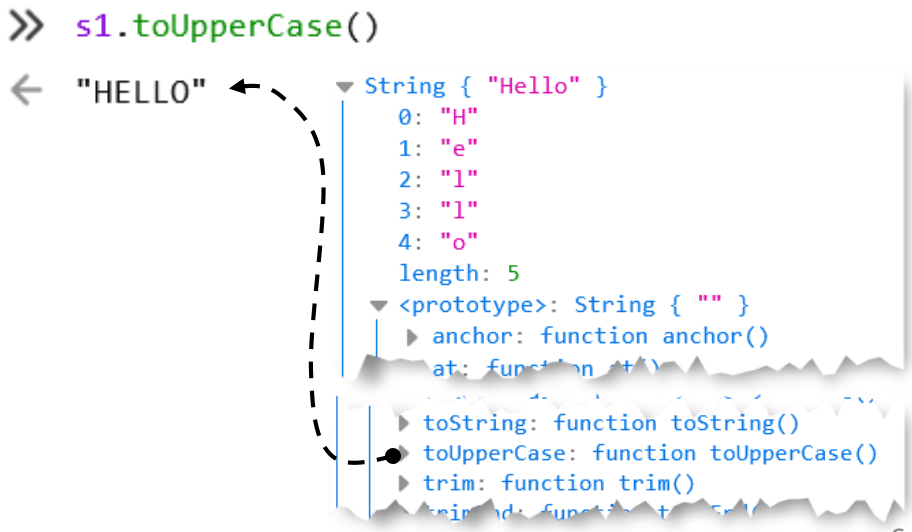
```
console.log(s1.length)
```
----> `5` accès à propriété `length`

```
console.log(s1.toUpperCase());
```
----> `HELLO` appel d'une méthode

Comment est-ce possible ?

- les string (ainsi que les variables des autres type primitifs (number, boolean …) se comportent comme des pseudo objets.
  - lorsque on accède à une de leur propriété où que l'on invoque une méthode la variable primitive est convertie en interne en **un objet temporaire** dont le constructeur est la classe 'enveloppe' (*wrapper class*) correspondante (`String`, Number, Boolean …)





string vs. String

Mais alors string et String ce n'est pas la même chose ?

Oui, le premier est un type primitif, le second un type objet

```
let chaine1 = 'une chaîne';
console.log(typeof chaine1);
```

```
---> string
```

```
let chaine2 = new String('une chaîne');
console.log(typeof chaine2);
```

```
---> object
```

chaine1 est une variable

chaine2 est la référence d'un objet enveloppe (wrapper)

Lorsqu'un message est envoyé à une variable `string`, elle est automatiquement convertie en objet `String`

Accéder aux caractères d'une chaîne

- méthode `charAt(pos)` ou `[pos]`

```
let s1 = "Hello World !";

console.log(s1.charAt(0));          ----> H

console.log(s1[s1.length - 1]);     ----> !
```

chaine vide

```
console.log(s1.charAt(100)); ----> ""
```

```
console.log(s1[100]);      ----> undefined
```

attention ! `charAt(pos)` et `[pos]` se comportent différemment si `pos` n'est pas dans l'intervalle `[0 .. length - 1]`

on peut utiliser boucle `for of` pour itérer sur les caractères d'une chaîne

```
for (let char of "Hi !") {
    console.log(char);      ---->
}
```

```
H
i

!
```

Immutabilité des chaînes de caractères

```
let s1 = "Hello";
s1.length = 4;
console.log(s1);
```

----> **Hello**

```
let s1 = "Hello";
s1[0] = 'h';
console.log(s1);
```

----> **Hello**

```
let s1 = "Hello";
s1.toUpperCase();
console.log(s1);
```

----> **Hello**

- En JavaScript une chaîne ne peut être modifiée
  - toutes les propriétés des objets String sont non modifiables et non configurables

```
» s1 = new String("Hello")
← String { "Hello" }
    0: "H"
    1: "e"
    2: "l"
    3: "l"
    4: "o"
    length: 5
  ▶ <prototype>: String { "" }
```

```
» Object.getOwnPropertyDescriptor(s1,"length")
← ▶ Object { value: 5, writable: false, enumerable: false, configurable: false }

» Object.getOwnPropertyDescriptor(s1,"0")
← ▶ Object { value: "H", writable: false, enumerable: true, configurable: false }
```

```
let s1 = "hello";
s1 = s1.toUpperCase();
console.log(s1);
```

----> **HELLO**

pour modifier la chaîne il faut lui affecter la nouvelle chaîne de caractères retournée par la méthode

Recherche dans une chaîne

- `String.prototype.indexOf(substr [, pos ])`
  - retourne l'index de la première occurrence de **`substr`** à partir de **`pos`** (optionnel, 0 par défaut), retourne **`-1`** si **`substr`** n'est pas présent dans la chaîne
- `String.prototype.lastIndexOf(substr [, pos ])`
  - même chose, mais en cherchant dans le sens inverse (depuis la fin de la chaîne)

```
let s1 = "On peut rire de tout , mais pas avec tout le monde";
// afficher la position de toutes les occurrences de "tout" dans s1
```

```
let pos = 0;
let foundPos = 0;
while (foundPos !== -1) {
    foundPos = s1.indexOf("tout",pos);
    if (foundPos !== -1) {
       console.log(`"tout" trouvé à la position ${foundPos}`);
       pos = foundPos + 4;
    }
}
```

```
----> "tout" trouvé à la position 16
      "tout" trouvé à la position 37
```

instruction d'affection peut être utilisée comme expression dont la valeur est l'expression affectée

variante plus concise

```
let pos = -4;
while ((pos = s1.indexOf("tout", pos + 4)) !== -1) {
    console.log(`"tout" trouvé à la position ${pos}`);
}
```

Recherche dans une chaîne

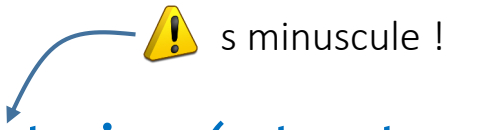
- `String.prototype.includes(substr [, pos ])`
- `String.prototype.startsWith(substr [, pos ])`
- `String.prototype.startsWith(substr [, pos ])`
  - retournent un booléen (**`true`** si **`substr`** est présente, **`false`** sinon)
  - A utiliser si on recherche la présence d'une chaîne mais que l'on a pas besoin de sa position

```
let s1 = "JavaScript";
console.log(s1.startsWith("Java")); ----> true
console.log(s1.endsWith("Script")); ----> true
console.log(s1.includes("Scr", 5)); ----> false
```

- d'autres méthodes de recherche à base d'expressions régulières
  - `String.prototype.search(regExp)`
  - `String.prototype.match(regExp)`
  - `String.prototype.matchAll(regExp)`

Récupérer une sous chaîne

- 3 méthodes
  - `String.prototype.substring(start, [end]))`
    - renvoie sous chaîne comprise entre les positions **`start`** et **`end`** (**`end`** non inclus)




```
let s1 = "0123456789" ;
```

```
s1.substring(3)
```
----> `"3456789"`

`end = str.length` par défaut

```
s1.substring(4, 7)
```
----> `"456"`

`end` non inclus

```
s1.substring(7, 4)
```
----> `"456"`

si **`start > end`** leur rôles sont inversés

```
s1.substring(6, 15)
```
----> `"6789"`

`end` peut être > à **`str.length`**
dans ce cas il prend la valeur **`str.length`**

```
s1.substring(-4, 3)
```
----> `"012"`

si **`start < 0`** sa valeur est considérée comme nulle **`(0)`**

```
s1.substring(4, -3)
```
----> `"0123"`

si **`end < 0`** sa valeur est considérée comme nulle **`(0)`**

```
s1.substring(-4, -3)
```
----> `""`

chaîne vide

Récupérer une sous chaîne

- 3 méthodes
  - `String.prototype.substr(start, [length]))`
    - renvoie sous chaîne de longueur **`length`** à partir de la positions **`start`**

```
let s1 = "0123456789" ;
```

```
s1.substr(3)
```
----> `"3456789"`

si **`length`** est omis on va jusqu'à la fin de la chaîne

```
s1.substr(4, 3)
```
----> `"456"`

`end` non inclus

```
s1.substr(7, 4)
```
----> `"456"`

si **`start > end`** leur rôles sont inversés

```
s1.substr(6, 15)
```
----> `"6789"`

`start + length` > à **`str.length`**
on va jusqu'à la fin de la chaîne

```
s1.substr(-4, 3)
```
----> `"678"`

si **`start < 0`** il définit une position de départ à partir de la fin de la chaîne

```
s1.substr(4, -3)
```
----> `""`

si **`end < 0`** retourne une chaîne vide

Récupérer une sous chaîne

- 3 méthodes
  - `String.prototype.slice(start, [end]))`
    - renvoie sous chaîne comprise entre les positions **`start`** et **`end`** (**`end`** non inclus)

```
let s1 = "0123456789" ;
```

```
s1.slice(3)
```
----> `"3456789"`

`end` = **`str.length`** par défaut

```
s1.slice(4, 7)
```
----> `"456"`

`end` non inclus

```
s1.slice(7, 4)
```
----> `""`

si **`start`** > **`end`** renvoie une chaîne vide

```
s1.slice(6, 15)
```
----> `"6789"`

`end` peut être > à **`str.length`**
dans ce cas il prend la valeur **`str.length`**

```
s1.slice(-8, 5)
```
----> `"234"`

si **`start`** < **`0`** sa valeur définit une position à partir de la fin de la chaîne

```
s1.slice(2, -5)
```
----> `"234"`

si **`end`** < **`0`** sa valeur est considérée comme nulle **`(0)`**

```
s1.slic(-2, -5)
```
----> `""`

chaîne vide (**`start`** > **`end`**)

```
s1.substring(-12, -5)
```
----> `"01234"`

si **`start`** < **`-str.length`** on part du premier caractère de str (index 0)

Comparer des chaînes (`strings`)

- les opérateurs `==`, `===`, `<` , `>`, `>=`, `<=` , `!=`, `!==` s'appliquent aux `strings`

```
'avenue' < 'avion'
                ----> true
```

```
'avenue' < 'avenue'
                ----> false
```

```
'avenue' < 'avenues'
                ----> true
```

```
'avenue' < 'Avion'
                ----> false
```

???

la relation d'ordre sur les caractères est basée sur leur code numérique (point de code) dans le codage Unicode.

pointDeCode(c1) < pointCode(c2) ⇔ c1 < c2

Les chaînes sont comparées caractère par caractère

1. Comparer le 1er caractère de s1 avec celui de s2
2. Si celui de s1 est plus petit que celui de s2 alors s1 < s2, si il est plus grand alors s2 < s1
   L'algorithme est terminé
3. Sinon les deux caractères sont identiques, on compare le second caractère de la même manière
4. Répéter jusqu'à ce que la fin d'une des deux chaînes soit atteinte
5. Si les deux chaînes ont la même longueur elles sont égales, sinon la chaîne a plus courte est la plus petite

```
function plusPetitStrict(s1,s2) {
    console.log(`'${s1}' < '${s2}'`);
    for (let i = 0; i < Math.min(s1.length,s2.length); i++) {
        if (s1[i] < s2[i]) {
            return true;
        } else if (s1[i] > s2[i]) {
            return false;
        }
    }
    return s1.length < s2.length;
}
```

Comparer des chaînes (strings)

- `String.prototype.codePointAt(pos)`
  - renvoie le point de code (rang dans jeu de caractères Unicode) du caractère situé à la position **pos**
- `String.fromCodePoint(code)`
  - renvoie une chaîne contenant le caractère de rang **code** dans le jeu de caractère Unicode

obtenir le point de code de 'A' et 'a'

```
'A'.codePoint(0)
```
----> **65**

```
'a'.codePoint(0)
```
----> **97**

65 < 97 ➔ 'A' < 'a'

obtenir les caractères pour une plage de points de codes

```
function getUnicodeChars(start, end) {
    let chars = "";
    for (let i = start; i <= end; i++) {
        chars += String.fromCodePoint(i);
    }
    return chars;
}
```

espace (space) — point de code : 48 — point de code : 65 — point de code : 97

32 à 127 : !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

160 à 255 : ¡¢£¤¥¦§¨©ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

espace non sécable (*non breaking space*, en HTML ** **)

point de code : 224

Comparer des chaînes (`strings`)

```
'Zoé' < 'Zoo'
```
----> **`false`**

car point de code de é (233) > point de code de o (111)

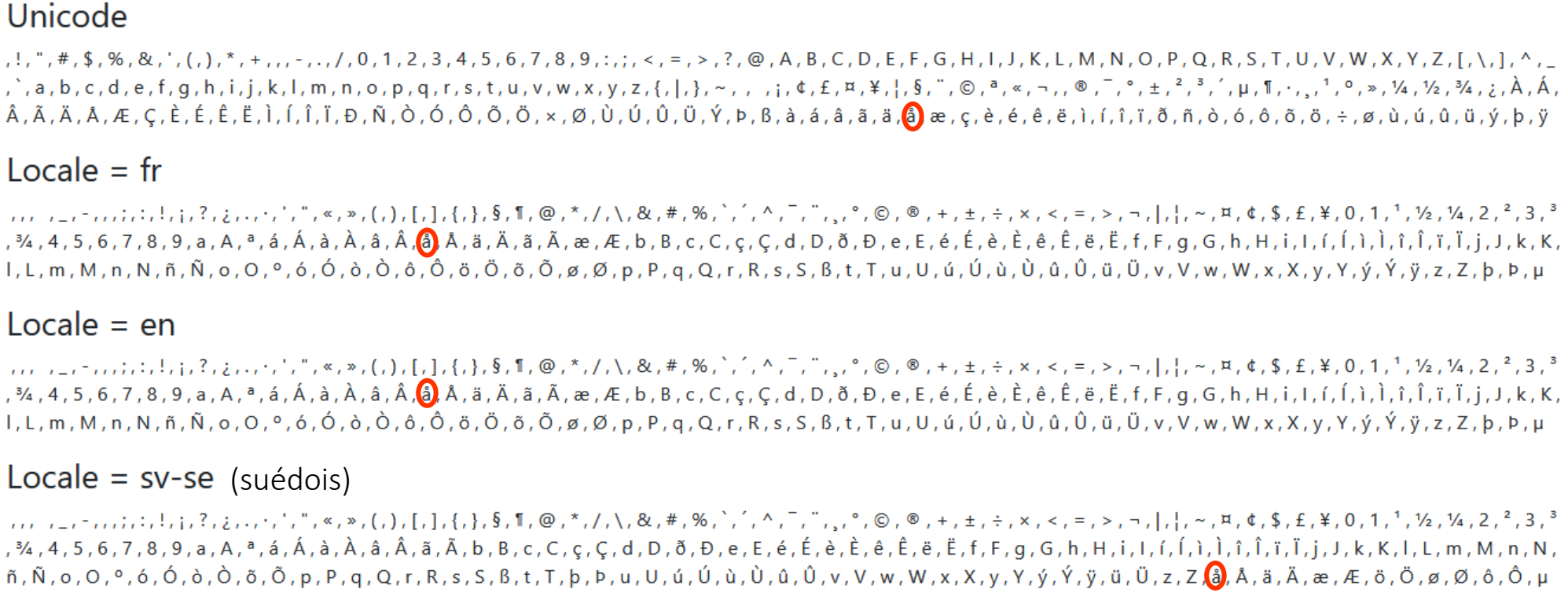
- String.prototype.localeCompare(compareString [, locale, [options]])
  - renvoie un nombre indiquant si la chaîne de caractères courante se situe avant, après ou est la même que la chaîne passée en paramètre, selon l'ordre lexicographique associé à la langue du locale.

```
'Zoé'.localeCompare('Zoo', 'fr')
```
ordre lexicographique

----> **`true`**

selon le locale utilisé l'ordre des lettres change

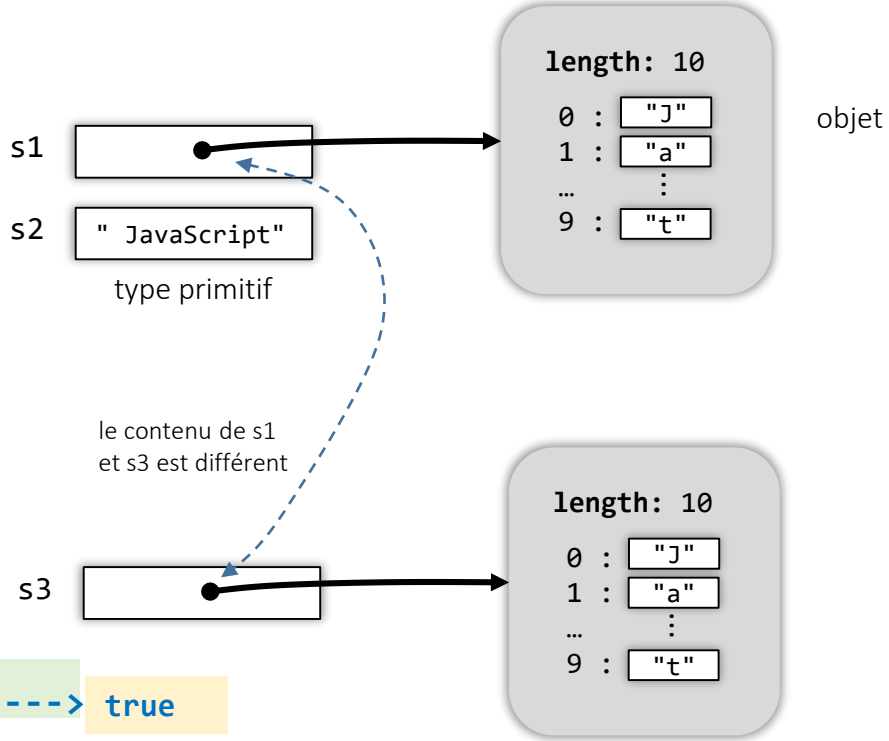
exemple å



Comparer des chaînes (`String`)

- Possibilité de créer des objets en utilisant le constructeur **`String`**

```
let s1 = new String("JavaScript");
let s2 = "JavaScript";
```




```
s1 == s2 ----> true
```

s1 est converti en **string** et la comparaison avec **s2** est effectuée

```
s1 === s2 ----> false
```

s1 et **s2** ne sont pas de même type (**object** et **string**)

```
let s3 = new String("JavaScript");
```

```
s1 == s3 ----> false
s1 === s3 ----> false
```

comparaison des références **s1** et **s3** ne désignent pas le même objet.

comparaison des valeurs

```
s1.valueOf() == s3.valueOf() ----> true
s1.valueOf() === s3.valueOf() ----> true
```

```
s2 = "Basic";
s1 > s2 ----> true
```

s1 est convertie en **string** et la comparaison avec **s2** est effectuée

```
s3 = new String("Basic");
s1 > s3 ----> true
```

s1 et **s3** sont converties en **string** et la comparaison est effectuée

Chaînes de caractères (résumé)

- type primitif `string`

```
let chaine = "essai" ;          console.log(typeof chaine); ---> string
chaine = 'essai' ;
chaine = "une chaine c'est une suite de caractères";
chaine = 'Dark Vador: "Je suis ton père."';
chaine = `la valeur ${Math.PI} correspond à 180°`;
```

- Nombre de caractères

```
chaine.length ;
```

- + concatenation

```
chaine = chaine + " une autre chaine";
```

- Bien que `string` ne soit pas un type objet possibilité d'appliquer des méthodes aux chaînes

```
let chaine = "essai" ;
console.log(chaine.toUpperCase());  ---> "ESSAI" ;
```

Chaînes de caractères (résumé)

- méthodes de manipulation de chaines
  - `charAt(i)` retourne le ième caractère de la chaîne
  - `indexOf(ch,i)` index de la première occurrence de `ch` à partir de `i` (optionnel)
  - `lastIndexOf(ch, i)` index de la dernière occurrence de `ch` à partir de `i` (optionnel)
  - `split(ch)` transforme la chaîne en tableau
  - `match(exp)` recherche les sous-chaînes correspondant à l'expression régulière `exp`
  - `search(exp)` index de la première correspondance entre la chaîne et `exp`
  - `replace(exp, ch)` remplace les occurrences de `exp` par `ch`
  - `substr(d,l)` sous-chaîne de longueur `l` commençant en `d`
  - `substring(d,f)` sous-chaîne entre `d` et `f`
  - `toLowerCase()` convertit la chaîne en minuscules
  - `toUpperCase()` convertit la chaîne en majuscules
  - …

- Définies dans le prototype de la fonction constructeur `String`