

Semaine 13

Fonctions utilitaires et résolution de problèmes

Intro. à la programmation



- ❖ Fonctions pour chaînes de caractères
 - ❖ Fonctions mathématiques
 - ❖ Apprendre d'autres langages
 - ❖ Résolution de problèmes
-
- ❖ Fonctions préexistantes
 - ◆ Depuis le début de la session, nous avons vu quelques **fonctions préexistantes** en JavaScript : `Math.random()`, `alert(...)`, `console.log(...)`, etc.
 - ◆ Aujourd'hui, nous allons en voir quelques-unes de plus qui pourraient vous servir dans le futur.



❖ Fonctions pour les chaînes de caractères

- ◆ .length
- ◆ .substring()
- ◆ .replace()
- ◆ .replaceAll()
- ◆ .toUpperCase()
- ◆ .toLowerCase()
- ◆ .charAt()



❖ Obtenir la taille d'une chaîne de caractères

◆ `.length` fonctionne aussi avec les chaînes de caractères !

- `.length` n'est pas tout à fait une fonction. (Donc pas de **parenthèses**) C'est une propriété qui contient le **nombre de caractères** dans la chaîne.

```
>> let texte1 = "Banane";  
← undefined  
  
>> texte1.length  
← 6
```

```
>> let texte2 = "A B C";  
← undefined  
  
>> texte2.length  
← 5
```



❖ Obtenir une partie de la chaîne de caractères

◆ `.substring(...)` retourne une « sous-chaîne » :

- Le **premier paramètre** est l'**index** du premier caractère à conserver.
- Le **deuxième paramètre** est l'**index** du caractère où la sous-chaîne s'arrête. (**Exclus**)

```
>> let texteComplet = "Patate";  
    let textePartiel = texteComplet.substring(2, 4);  
← undefined  
  
>> textePartiel  
← "ta"
```

Patate

0 1 2 3 4 5

...substring(2, 4);

```
>> let texteComplet = "20 chaises bleues";  
    let textePartiel = texteComplet.substring(3, 10);  
← undefined  
  
>> textePartiel  
← "chaises"
```

20 chaises bleues

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

...substring(3, 10);



❖ Remplacer un caractère ou un groupe de caractères

- ◆ `.replace(...)` permet de **remplacer** un groupe de caractères. Elle retourne la chaîne de caractères modifiée.
 - Le **premier paramètre** est le groupe de 1+ caractère(s) à remplacer.
 - Le **deuxième paramètre** est le groupe de remplacement.

```
>> let texte = "J'aime les licornes";  
    let texteModif = texte.replace("cornes", "vres");  
← undefined  
  
>> texteModif  
← "J'aime les livres"
```

```
>> let texte = "Je suis très content";  
    let texteModif = texte.replace("très ", "");  
← undefined  
  
>> texteModif  
← "Je suis content"
```

```
Console | Inspecteur | Débogueur | Rés  
Filtrer  
  
>> let texte = "J'ai pris un déjeuner";  
← undefined  
  
>> let texteModif = `${texte.replace("un", "quatre")}s`;  
← undefined  
  
>> texteModif  
← "J'ai pris quatre déjeuners"
```

Ci-dessus, on peut voir que même si « **un** » apparaît également dans le mot « déjeuner », **seule la première occurrence est remplacée**.



❖ Remplacer plusieurs groupes de caractères

◆ `.replaceAll(...)` est similaire à `.replace()`, mais elle remplace **toutes les occurrences** du texte à remplacer.

- Le **premier paramètre** est le groupe de 1+ caractère(s) à remplacer.
- Le **deuxième paramètre** est le groupe de remplacement.

```
>> let texte = "J'ai pris un déjeuner";  
    let texteModif = texte.replaceAll("un", "quatre");
```

```
← undefined
```

```
>> texteModif
```

```
← "J'ai pris quatre déjeuner"
```

```
>> let texte = "À cause de mon rhume, mes amis se moquent de moi";  
    let texteModif = texte.replaceAll("m", "b");
```

```
← undefined
```

```
>> texteModif
```

```
← "À cause de bon rhube, bes abis se boquent de boi"
```



❖ Tout mettre en majuscules / minuscules

- ◆ `.toLowerCase()` et `.toUpperCase()` retournent la chaîne de caractères, mais complètement en minuscules / majuscules.
 - Ça n'a aucun impact sur les caractères qui ne sont pas des lettres.

```
>> let texte = "Je trouve que Sophie est normale.";
    let texteMaj = texte.toUpperCase();
    let texteMin = texte.toLowerCase();

← undefined

>> texteMaj
← "JE TROUVE QUE SOPHIE EST NORMALE."

>> texteMin
← "je trouve que sophie est normale."
```




- ❖ Obtenir un caractère à une position précise dans une chaîne
 - ◆ `.charAt(...)` retourne le caractère situé à l'**index** de notre choix dans une chaîne de caractères.
 - Un **paramètre** est nécessaire : c'est l'**index** du caractère demandé.

```
>> let texte = "zèbre";  
    let lettre = texte.charAt(0);  
← undefined  
  
>> lettre  
← "z"
```

```
>> let texte = "Omar fait de la bicyclette."  
    let lettre = texte.charAt(4);  
← undefined  
  
>> lettre  
← " "
```

```
>> let texte = "Simone va ouvrir une buvette."  
    let lettre = texte.charAt(texte.length - 1);  
← undefined  
  
>> lettre  
← "."
```



❖ Fonctions mathématiques

- ◆ `Math.round()`
- ◆ `Math.ceil()` et `Math.floor()`
- ◆ `Math.min()` et `Math.max()`
- ◆ `Math.random()`



❖ Arrondir un nombre

- ◆ `Math.round(...)` retourne un nombre arrondi au **nombre entier** le plus près.

```
>> let distance = 24.1563235;  
← undefined  
  
>> let texte = `J'habite à ${Math.round(distance)} kilomètres d'ici !`;  
← undefined  
  
>> texte  
← "J'habite à 24 kilomètres d'ici !"
```

Nombre arrondi **vers le bas**

```
>> let age = 24.67;  
    let ageArrondi = Math.round(age);  
← undefined  
  
>> ageArrondi  
← 25
```

Nombre arrondi **vers le haut**



❖ Arrondir un nombre forcément vers le bas / vers le haut

- ◆ `Math.floor(...)` retourne un nombre arrondi vers le bas.
- ◆ `Math.ceil(...)` retourne un nombre arrondi vers le haut.

```
>> let age = 17.98;  
← undefined  
  
>> `J'ai ${Math.floor(age)} ans.`  
← "J'ai 17 ans."
```

Généralement, quand on mentionne son âge, on arrondit toujours vers le bas 🙄

```
>> let longueur = 13.05;  
← undefined  
  
>> `J'ai besoin de ${Math.ceil(longueur)} mètres de corde.`  
← "J'ai besoin de 14 mètres de corde."
```

Si on ne veut pas manquer de quelque chose, on l'arrondit vers le haut 😬



❖ Obtenir le maximum ou le minimum

- ◆ `Math.max(...)` retourne la valeur maximale entre plusieurs nombres.
- ◆ `Math.min(...)` retourne la valeur minimale entre plusieurs nombres.

```
>> let a = 5;  
    let b = 3;  
← undefined  
  
>> Math.max(a, b);  
← 5
```

```
>> Math.min(2, 4, 1, 3)  
← 1
```

Autant pour `max` que pour `min`, on peut mettre 2 valeurs ou plus, séparées par des virgules.

Cette fonction permet de soigner le joueur (lui redonner de la vie) sans dépasser le maximum de 100. Ici, `gVieJoueur + valeurSoin` donnait 143, alors 100 a été choisi par `Math.min()`.

```
function soignerJoueur(valeurSoin){  
    gVieJoueur = Math.min(100, gVieJoueur + valeurSoin);  
}
```

```
>> gVieJoueur  
← 93  
  
>> soignerJoueur(50);  
← undefined  
  
>> gVieJoueur  
← 100
```



❖ Obtenir un nombre aléatoire

- ◆ Nous avons déjà vu `Math.random()`, mais voici des exemples un peu plus sophistiqués.

Obtenir un nombre aléatoire entre 0 et 0.99999...

```
>> let nombreAleatoire = Math.random();  
← undefined  
  
>> nombreAleatoire  
← 0.8758528200387565
```

Obtenir un nombre aléatoire entre 0 et 4.99999...

```
>> let nombreAleatoire = Math.random() * 5;  
← undefined  
  
>> nombreAleatoire  
← 3.814723212502409
```

Obtenir un nombre aléatoire entre 5 et 9.99999...

```
>> let nombreAleatoire = 5 + Math.random() * 5;  
← undefined  
  
>> nombreAleatoire  
← 7.241046042976543
```



❖ Obtenir un nombre aléatoire

- ◆ Nous avons déjà vu `Math.random()`, mais voici des exemples un peu plus sophistiqués.

Obtenir un nombre aléatoire parmi 1, 2, 3, 4 et 5.
(Arrondir vers le bas élimine tous les nombres à virgule !)

```
>> let nombreAleatoire = 1 + Math.floor(Math.random() * 5);  
← undefined  
  
>> nombreAleatoire  
← 3
```

Recette pour obtenir un nombre entier entre X et Y :

```
X + Math.floor(Math.random() * (Y - X + 1))
```

Exemple pour un nombre entre 1 et 10 :

```
>> let nombre = 1 + Math.floor(Math.random() * 10);
```

Exemple pour un nombre entre 17 et 23 :

```
>> let nombre = 17 + Math.floor(Math.random() * 7);
```

(23 - 17 + 1 donne 7)



❖ Bonne nouvelle ! 🤖

◆ Les langages de programmation ont énormément de similarités !

```
let a = 5;

if(a > 100){
    a += 3;
}
```

Code en JavaScript

```
int a = 5;

if(a > 100){
    a += 3;
}
```

Code en Java

```
int a = 5;

if(a > 100){
    a += 3;
}
```

Code en C#

```
int a{ 5 };

if(a > 100){
    a += 3;
}
```

Code en C++

```
local a = 5

if a > 100 then
    a += 3
end
```

Code en Lua

```
a = 5

if a > 100
    a += 3
end
```

Code en Ruby

```
a = 5

if a > 100:
    a += 3
```

Code en Python



- Bien entendu, c'est un **petit échantillon** et il y a plus de **différences** que cela.
- Cela dit, lorsqu'on connaît déjà un langage de programmation, en apprendre d'autres est de plus en plus facile ! ✨



❖ Apprendre d'autres langages

- ◆ Bien que nous ayons appris beaucoup de notions en **JavaScript**, nous n'avons vu **que les bases** ! Si vous explorez d'autres langages, vous rencontrerez plusieurs nouvelles notions qui ne vous seront pas familières.
- ◆ Malgré tout, n'ayez pas peur d'apprendre d'autres langages dès maintenant !
 - Il faut apprendre le langage **C#** pour exploiter Unity en profondeur.
 - Il faut apprendre **Kotlin**, **Java** ou autre pour faire des applications Android.
 - Il faut apprendre **C#**, **Java**, **Python** ou autre pour faire un serveur Web.
 - Il faut apprendre **SQL** pour exploiter les bases de données relationnelles.
 - etc.
- ◆ Selon les types de projets qui vous intéressent, il y a beaucoup à apprendre
 - Bien entendu, dans l'univers TIM, vous n'aurez pas forcément besoin d'apprendre beaucoup de langages de programmation 😊



❖ Résolution de problèmes

- ◆ Nous avons terminé de faire le tour des bases en JavaScript 🤖
- ◆ Pour ce dernier labo, nous nous concentrerons surtout sur la « **résolution de problèmes** ».
 - (Sauf les 2 premiers exercices, qui sont pour les **fonctions de texte** et **mathématiques**)
 - Vous ferez face à des exercices plus « **libres** » et **difficiles** que d'habitude :
 - Vous pouvez utiliser **n'importe quelle notion** apprise cette session, **tant que le résultat fonctionne !**
 - Le but est de réfléchir et de réussir à exploiter les bases que nous avons acquises durant toute la session.
 - 📝 **Attention !** À l'**examen final**, un exercice parmi les **exercices 3 à 10** sera présent. Il est donc important de comprendre tous les exercices, quitte à prendre le temps, avec l'enseignant(e), de refaire ceux qui vous donnent plus de fil à retordre.

❖ Résolution de problèmes

- ◆ L'aide-mémoire est maintenant complet. En effectuant les exercices de résolution de problème, essayez de garder à l'esprit toutes les notions qu'on a vues !

Besoin de manipuler des variables ou des valeurs ?

```
let      =, +=, -=      +, -, *, /
```

Fonctions mathématiques

Concaténation

Fonctions pour chaînes de caractères

Seulement exécuter du code si une condition est respectée ?

if else else if .. ? .. : ..
<, <=, >, >=, ==, != &&, || switch

Répéter ou réutiliser du code similaire ?

function() function(a, b)

while do while for

Besoin de modifier la page Web ?

querySelector querySelectorAll .textContent

.style.propriété .classList... .nomAttribut

```
.createElement      .addEventListener
```

Besoin de manipuler beaucoup de valeurs liées ?

[1, 2, 3] .length .pop

`.push` `.splice`

Autres ...

currentTarget setTimeout setInterval

clearInterval alert console.log