Semaine 13

Fonctions utilitaires et résolution de problèmes

Intro. à la programmation

Menu du jour 🍴



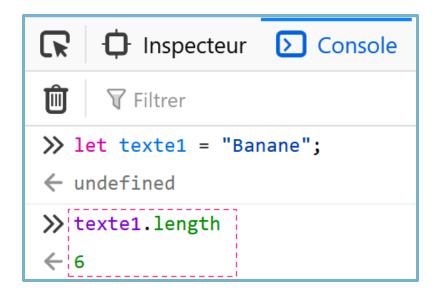
- Fonctions pour chaînes de caractères
- Fonctions mathématiques
- Commenter ses fonctions
- Apprendre d'autres langages
- * Résolution de problèmes
- Fonctions préexistantes
 - ◆ Depuis le début de la session, nous avons vu quelques fonctions préexistantes en JavaScript : Math.random(), alert(...),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

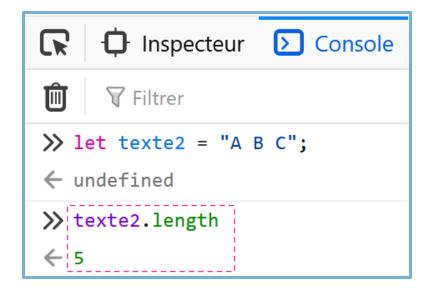


- Fonctions pour les chaînes de caractères
 - ◆ .length
 - .substring()
 - .replace()
 - ◆ .replaceAll()
 - ◆ .toUpperCase()
 - ◆ .toLowerCase()
 - .charAt()



- 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.







- 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)

```
Patate

0 1 2 3 4 5

...substring(2, 4);
```

```
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.

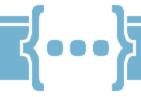
Ci-dessus, on peut voir que même si « un » apparait également dans le mot « déje<u>un</u>er », seule la première occurrence est remplacée.



- 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.



- 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.



- 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.



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

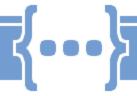


* Arrondir un nombre

♦ Math.round(...) retourne un nombre arrondi au nombre entier le plus près.

Nombre arrondi vers le haut

Nombre arrondi vers le bas



- 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.

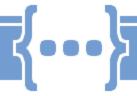
```
>> let age = 17.98;
← undefined
>> `J'ai ${Math.floor(age)} ans.`
← "J'ai 17 ans."
```

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 manguer 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.

```
>> 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. lci, gVieJoueur + valeurSoin donnait 143, alors 100 a été choisi par Math.min().

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



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...

Obtenir un nombre aléatoire entre 0 et 4.99999...

```
>> let nombreAleatoire = Math.random();

    undefined

>> nombreAleatoire

    0.8758528200387565
```

```
Obtenir un nombre aléatoire entre 5 et 9.99999...
```



- 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!)

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);
```



Commentaires dans le code

- ♦ Nous les connaissons depuis la semaine 2, mais nous ne les avons jamais rédigés nous-mêmes !
- ◆ TP3 : Vous devrez créez vos propres fonctions et vous devrez ... les commenter !
 - Lorsqu'on crée du code, c'est important de le commenter (le décrire) pour que nos collègues puissent comprendre et naviguer facilement notre travail.
 - Les prochaines diapos expliquent comment rédiger de bons commentaires.

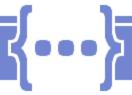


Commenter une fonction

- ♦ Points clés :
 - 1. Décrire brièvement l'utilité de la fonction. (Sans entrer dans des détails trop techniques)
 - 2. Si la fonction reçoit des paramètres, que représentent-ils ?
 - 3. Si la fonction retourne une valeur, que représente-t-elle ?
- ♦ Exemple 1

```
// Reçoit une valeur numérique au choix, un minimum et un maximum en paramètres.
// Retourne true si la valeur est située entre le minimum et le maximum fournis
// et false sinon.
function respecteIntervalle(valeur, min, max){

   if(valeur >= min && valeur <= max) {
      return true;
   }
   return false;
}</pre>
```



- Commenter une fonction
 - ♦ Exemple 2 et 3

```
// À l'aide d'un chemin vers une image source et d'une largeur en nombre
// de pixels, crée un nouvel élément img et le retourne.
function creerImage(source, largeur){

let image = document.createElement("img");
   image.style.width = largeur + "px";
   image.classList.add("thumbnail");
   image.src = source;
   image.alt = "Miniature de photo";
   return image;
}
```

```
// Si le jeu n'est pas déjà en cours, lance le jeu et réinitialise
// le score.
function lancerJeu(){

   if(gJeuActif == false){

       gJeuActif = true;
       gScore = 0;
       document.querySelector(".score").textContent = `Score : ${gScore}`;
       gPlanif = setInterval(miseAJour, 50);
}
```







Commenter une fonction

- **♦ Erreurs** typiques
 - Trop décrire la fonction et aller dans les détails techniques

```
//iSi gJeuActif veut false, met gJeuActif à true, puis met gScore à 0,
// puis met à jour le contenu textuel de .score dans la page et lance
// un planificateur à intervalle qui appelle la fonction miseAJour()
// toutes les 50 millisecondes.
function lancerJeu(){
   if(gJeuActif == false){
        gJeuActif = true;
        gScore = 0:
       document.querySelector(".score").textContent = `Score : ${gScore}`;
       gPlanif = setInterval(miseAJour, 50);
```

```
// Reçoit une valeur numérique au choix, un minimum et un maximum en paramètres.
/// Si valeur est plus grande ou égale à min et que valeur est plus petite ou égale à
// max, on retourne true, sinon on retourne false.
function respecteIntervalle(valeur, min, max){
    if(valeur >= min && valeur <= max) {
        return true;
    return false;
```



X Au lieu de résumer conceptuellement en français les fonctions, on décrit les lignes de code de la fonction. X







Commenter une fonction

- **♦ Erreurs** typiques
 - Oublier de parler des paramètres

```
// Retourne une valeur multipliée par 3
function triplerValeur(n){
   return n * 3;
        Quelle valeur?
}
```



```
// Retourne la valeur numérique reçue en paramètre, multipliée par 3
function triplerValeur(n){
    return n * 3;
}
```

Oublier de parler du retour

```
Amélioration
```

```
// Calcul le prix avec taxe pour un prix reçu en paramètre et le retourne.
function prixAvecTaxes(prix){
    return prix * 1.15;
}
```

Apprendre d'autres langages



❖ 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 connait déjà un langage de programmation, en apprendre d'autres est de plus en plus facile!

Apprendre d'autres langages



- 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.
 - o 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



* 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



* 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 ?

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