Développement Web
TP n° 1 : Prise en main

Dans ce TP, vous allez découvrir l'environnement JavaScript. Aucune installation n'est nécessaire : le navigateur suffit.

Exercice 1 : Interpréteur

Ouvrez votre navigateur favori et affichez les outils de développement. La manipulation à faire dépend de votre navigateur, voici les raccourcis les plus communs :

- **Chrome / Internet Explorer / Edge.**
  CTRL+SHIFT+J (CMD+OPT+J sur Mac) ou F12,
- **Firefox.**
  F12 ou CTRL+SHIFT+J (CMD+OPT+J sur Mac) pour n'avoir que la console,
- **Opera.**
  CTRL+SHIFT+J (CMD+OPT+J sur Mac),
- **Safari.**
  Allez dans Préférences puis cliquez sur Avancées et sélectionnez "Afficher le menu Développement dans la barre des menus". Les outils de développement sont désormais accessibles via CMD+OPT+C.

Dans tous les cas, une fenêtre devrait s'ouvrir avec plusieurs onglets. Sélectionnez l'onglet Console pour accéder au mode interactif de JavaScript. Tapez quelques commandes et observez le résultat. Essayez en particulier d'appeler une des fonctions `alert`, `prompt` et `confirm`.

La console est extrêmement pratique pour tester ou déboguer vos scripts. Vous pourrez également modifier dynamiquement vos pages Web à l'aide de la console et observer le résultat en temps réel.

Exercice 2 : Un grand classique

Dans cet exercice, vous allez écrire un programme qui demande à l'utilisateur de retrouver un nombre. À chaque proposition, il faudra indiquer si le nombre a été trouvé ou s'il est plus grand ou plus petit que la proposition.

Vous écrirez votre script dans un fichier `plus-moins.js` qui sera appelé depuis la page `plus-moins.html` donnée ci-dessous.

```html
<!-- plus-moins.html -->
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8" />
    <title>Jeu du plus ou moins</title>
  </head>

  <body>
    <script src="./plus-moins.js"></script>
  </body>
</html>
```

1. Écrivez une fonction `randInt(from, to)` qui renvoie un entier aléatoire de l'intervalle `[from, to]`. Tous les entiers possibles doivent avoir la même probabilité de sortie. Consultez la documentation de l'objet `Math` pour trouver les fonctions à utiliser.
2. Modifiez `randInt` pour que l'appel `randInt(to)` renvoie un entier aléatoire de l'intervalle `[0, to]` comme le ferait `randInt(0, to)`. Par exemple, `randInt(1, 6)` est équivalent à `1 + randInt(5)`.

3. Implémentez le jeu en supposant que toutes les entrées de l'utilisateur sont valides. On pourra lancer une partie à l'aide de l'appel `demarrerPartie(max)`. L'objectif à découvrir sera un entier aléatoire entre `0` et `max`. Les entrées-sorties se font uniquement à l'aide des fonctions `alert`, `prompt` ou `confirm`.

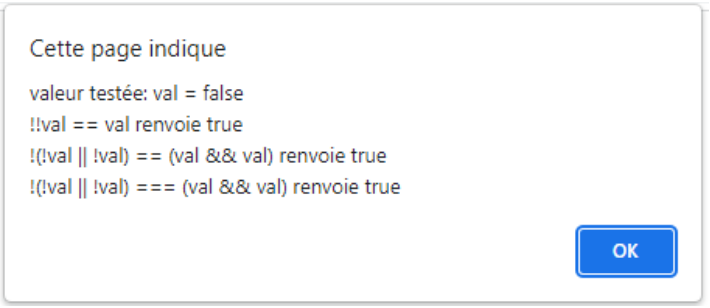
   Les points suivants constituent des améliorations successives de cette première implémentation.
4. Lorsque l'utilisateur trouve l'objectif, indiquez le nombre d'essais réalisés.
5. Lorsque vous demandez une proposition à l'utilisateur, indiquez la plage de valeurs autorisées.
6. Si l'utilisateur appuie sur Annuler à ce moment, quitter la partie.
7. Avant de quitter la partie, demandez confirmation à l'utilisateur que c'est bien son intention.
8. Si une entrée est invalide (ce n'est pas un nombre), indiquez-le et n'incrémentez pas le nombre d'essais réalisés.
9. Vérifiez que le point précédent est correct pour une entrée blanche (par exemple que des espaces et des tabulations).

Exercice 3 : Comprendre les conversions

Cet exercice vous amène à réfléchir aux conversions implicites mises en place par JavaScript et aux dangers qu'elles peuvent représenter.

1. Écrivez une fonction `test(val)` qui affiche le résultat des expressions :
   — `!!val == val`,
   — `!(!val || !val) == (val && val)` et
   — `!(!val || !val) === (val && val)`.

   Par exemple, `test(false)` affiche :



2. Prévoyez puis vérifiez les résultats de la fonction `test` pour les valeurs `"0"` et `"1"`. Si vous bloquez, demandez dans la console la valeur de chaque côté des égalités.
3. Un utilisateur a écrit la fonction suivante. Pourquoi ne fonctionne-t-elle pas comme attendu ?

```
function menu(msg = "") {
  const menuText = `Sélectionnez une option:
        1) Salutation
        2) Aujourd'hui`;
  const choice = prompt(msg ? `${msg}\n${menuText}` : menuText);
  switch (choice) {
    case null:
      break;
    case 1:
      alert("Hello world");
      break;
    case 2:
      alert(new Date());
      break;
    default:
      menu("Choix invalide, veuillez recommencer.");
  }
}
```

Exercice 4 : Des boucles sans boucle

Sans utiliser les boucles du langage (*i.e.* `for`, `while` et `do while`), définissez les fonctions suivantes :

1. `forRange(i, j, process)` qui exécute `process` pour tous les entiers de l'intervalle `[i, j]`.

   ```
   let n = 0;
   forRange(1, 10, (i) => { n += i; });
   alert("n = " + n); // n = 55
   ```

2. `forStep(val, pred, process, next)` qui exécute `process` sur la valeur `val` si elle vérifie le prédicat `pred`. En ce cas, on continue avec la valeur suivante `next(val)` et ainsi de suite.

   ```
   const val = 9;
   let syracuse = "";
   forStep(val, (v) => v != 1,
     (v) => syracuse += `${v} -> `,
     (v) => v % 2 == 0 ? v / 2 : 3 * v + 1);
   alert(syracuse + "1");
   /* 9 -> 28 -> 14 -> 7 -> 22 -> 11 -> 34 -> 17 -> 52 -> 26 -> 13 ->
       40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1 */
   ```

3. `forRangeFilter(i, j, pred, processTrue, processFalse)` qui exécute `processTrue` pour tous les entiers de l'intervalle `[i, j]` vérifiant le prédicat `pred` et `processFalse` pour ceux qui ne le vérifient pas. Les valeurs `processTrue` ou `processFalse` peuvent être non définies pour indiquer l'absence de traitement dans ces cas.

   ```
   forRangeFilter(1, 10, (v) => v % 2 == 0, null, (v) => alert(`${v} est impair.`));
   ```