Javascript : exercices séance 1

Javascript est un langage aux nombreux interpréteurs. Bien sûr chaque navigateur possède le sien, mais on le retrouve aussi du côté serveur avec Node.js par exemple. Pour le moment nous utiliserons le moteur Javascript fournit par Java 8, qui s'appelle Nasshorn.

Prenez le fichier Nasshorn.zip se trouvant sur l'ecampus, décompressez-le, et importez le projet dans Eclipse :

File/Import/General/Existing Project Into Workspace/Browse/pointez vers le repertoire que vous avez dézippé, puis Next etc jusqu'à terminer l'import.

Exécutez la classe Nasshorn.

Dans la console, entrez `1+1`, faites Enter et hop, vous avez le résultat.

Le moteur Javascript possède ce qu'on appelle un store : un espace de stockage auquel on ajoute de l'information.

Entrez :

```
function hello() {return "hello";}
```

La fonction est ajoutée au store. Entrez :

```
hello()
```

et elle est appelée.

Ce serait plus pratique d'enregistrer tout cela dans un fichier. Créez un fichier `hello.js` à la racine[1] de votre projet Eclipse et copiez-y la définition de la fonction `hello` ci-dessus. Pour charger le fichier, il faut faire :

```
> file.loadScript("hello.js")
```

Le script est alors chargé et exécuté immédiatement.

A noter : chaque chargement ajoute au store déjà existant. Si vous désirez nettoyer complètement le store, il faut faire :

```
> location.reload()
```

---

[1] Le répertoire src n'est pas la racine de votre projet. Son répertoire parent l'est par contre !

Ex 1.

A. Ecrivez une boucle qui va de 1 à 100. Imprimez chaque numéro à la console. C'est `console.log(variable)` qui permet d'afficher variable à la console.

B. Ecrivez une boucle qui va de 1 à 100. Imprimez chaque numéro à la console, mais les chiffres multiples de 3 sont remplacés par Fizz, ceux multiples de 5 par Buzz et ceux multiples d'à la fois 3 et 5 sont remplacés par FizzBuzz.
Nota : x est multiple de y ⇔ le restant de la division entière de x par y est zéro. En Javascript, c'est l'opérateur `%` (appelé modulo) qui calcule le restant d'une division entière.

C. Transformons la solution de l'exercice précédent en une fonction appelée `FizzBuzz` et prenant le nombre d'itérations en paramètre. Appeler `FizzBuzz(100)` va itérer sur les 100 premiers nombres, tandis que `FizzBuzz(200)` sur les 200, etc. Sauvez cette fonction dans un fichier `ex1c.js` à la racine de votre projet. Pour la charger :

```
> file.loadScript("ex1c.js")
```

Notez le résultat : `<function>` qui indique que la dernière chose évaluée lors de l'exécution de ce script est une fonction. Appelez-la :

```
> FizzBuzz(200)
```

Ex 2.

Nous allons demander à l'ordinateur de jouer à Ding Ding Bottle. Les règles sont les suivantes :

- Les multiples de 5 sont remplacés par Ding Ding
- Les multiples de 7 et les chiffres se terminant par 7 sont remplacés par Bottle. Donc 7,14, 21, … mais aussi 17, 27, 37, …
- Les multiples de 5 et 7 sont remplacés par Ding Ding Bottle.
- Le jeu se joue autours d'une table ; chaque fois qu'un joueur a dit son chiffre ou son remplacement, c'est au joueur qui le suit dans le sens actuel de jeu qui doit s'occuper du chiffre suivant. Cependant, lorsqu'un Bottle est dit, le sens s'inverse ! C'est donc alors au joueur précédent dans l'ancien sens de rotation qui doit s'occuper du chiffre suivant.

A. Nous allons d'abord implémenter un joueur :
   - Il possède un nom. Cette valeur est déterminée une fois pour toute.
   - Il possède un joueur à sa droite et un joueur à sa gauche. Ces valeurs peuvent être changées n'importe quand.
   - Il possède une fonction pour s'occuper d'un chiffre. Cette fonction reçoit le chiffre et la référence de qui l'appelle : ce sera soit son joueur de droite, soit son joueur de gauche. Cette seconde information permet de déterminer le sens du jeu. N'oubliez pas d'inverser ce sens en cas de Bottle !
     Cette fonction affiche le nom du joueur suivi de ce qu'il doit dire, puis elle passe la main au joueur suivant. Quand le chiffre à afficher est plus grand que 100, cette fonction ne fait plus rien.

   Implémentez ceci en utilisant la technique de pseudo-classe vue dans les diapos.

B. Créons une partie en adaptant le code suivant en fonction de votre propre code :

```
var a=joueur('a');
var b=joueur('b');
var c=joueur('c');
a.setDroite(b);
b.setDroite(c);
c.setDroite(a);
a.setGauche(c);
c.setGauche(b);
b.setGauche(a);
a.traitement(1,c);
```

Notez la dernière ligne : elle demande au joueur a de traiter la valeur 1, et en déclarant c comme étant son prédécesseur afin d'imposer le sens initial a->b->c->a->…

Le résultat devrait être :

```
a: 1
b: 2
c: 3
a: 4
b: Ding ding
c: 6
a: Bottle
c: 8
b: 9
a: Ding ding
c: 11
b: 12
a: 13
c: Bottle
a: Ding ding
b: 16
c: Bottle
b: 18
a: 19
c: Ding ding
b: Bottle
c: 22
a: 23
b: 24
c: Ding ding
a: 26
b: Bottle
a: Bottle
b: 29
c: Ding ding
a: 31
b: 32
c: 33
a: 34
```

```
b: Ding Ding Bottle
a: 36
c: Bottle
a: 38
b: 39
c: Ding ding
a: 41
b: Bottle
a: 43
c: 44
b: Ding ding
a: 46
c: Bottle
a: 48
b: Bottle
a: Ding ding
c: 51
b: 52
a: 53
c: 54
b: Ding ding
a: Bottle
b: Bottle
a: 58
c: 59
b: Ding ding
a: 61
c: 62
b: Bottle
c: 64
a: Ding ding
b: 66
c: Bottle
b: 68
```

```
a: 69
c: Ding Ding Bottle
a: 71
b: 72
c: 73
a: 74
b: Ding ding
c: 76
a: Bottle
c: 78
b: 79
a: Ding ding
c: 81
b: 82
a: 83
c: Bottle
a: Ding ding
b: 86
c: Bottle
b: 88
a: 89
c: Ding ding
b: Bottle
c: 92
a: 93
b: 94
c: Ding ding
a: 96
b: Bottle
a: Bottle
b: 99
c: Ding ding
```

C. Améliorons le programme : si b est à droite de a, alors a est à gauche de b. Automatisez cela : chaque fois qu'on appelle `setDroite` ou `setGauche`, l'élément dans l'autre sens est aussi

automatiquement placé. Comment faire pour éviter d'avoir une boucle infinie : `a.setDroite(b)` => `b.setGauche(a)` => `a.setDroite(b)` => … ?
C'est possible de trouver une solution sans introduire de getter sur les droites et gauches, et sans introduire de fonctions supplémentaires.

D. Même s'il ne faut plus qu'utiliser `setDroite` ou `setGauche`, configurer une table de jeu reste pénible. Créons la fonction `dingdingbottle` qui prend un nombre arbitraire de paramètres. Chaque paramètre est un nom de joueur. La fonction configure une table avec ces joueurs dans l'ordre dans lesquels ils sont fournis, le dernier étant relié au premier. Ensuite elle lance la partie. Ainsi :

```
> dingdingbottle("a","b","c")
```

donne de nouveau le résultat de la page précédente.
Pour réaliser cette fonction, nous aurons besoin de la variable appelée `arguments` : c'est une variable de type tableau et automatiquement définie dans les fonctions. Elle contient l'ensemble des arguments passés en paramètre à la fonction : `arguments[0]` sera le premier paramètre, `arguments[1]` le second, `arguments.length` donnera le nombre d'arguments réellement passés, etc…

Ex 3.

Jouons à puissance 4 : https://fr.wikipedia.org/wiki/Puissance_4

A. Créez un pseudo-objet grille.
   a. Cette grille contient 7 colonnes de 6 cases. Chaque case contient soit : rien, un pion jaune ou un pion rouge.
   b. Une fonction imprime la grille. Utilisez `console.log`. Comme cette fonction ne permet pas d'afficher de la couleur, utilisez X pour les pions jaunes et O pour les pions rouges.
   c. Une fonction permet de jouer en désignant une colonne et une couleur. Elle renvoie vrai si le coup est valide (il reste de la place dans la colonne), faux au sinon. La case la plus basse de cette colonne prend la couleur désignée.
   d. Une fonction interne (privée) vérifie s'il y a une suite de 4 pions de la même couleur dans chacune des colonnes. Elle retourne une valeur représentant rien, rouge ou jaune (p.ex. un entier, ou un string, ou ….).
   e. Une fonction interne (privée) vérifie s'il y a une suite de 4 pions de la même couleur dans chacune des lignes. Elle retourne une valeur représentant rien, rouge ou jaune.
   f. Une fonction interne (privée) vérifie s'il y a une suite de 4 pions de la même couleur en diagonale, dans les deux directions possibles. Elle retourne une valeur représentant rien, rouge ou jaune.
   g. Une fonction qui vérifie s'il y a un gagnant. Elle retourne une valeur représentant rien, rouge ou jaune.
   h. Une fonction qui vérifie s'il reste de la place dans la grille. Elle renvoie vrai quand la grille est remplie, faux au sinon.
B. Testez votre objet grille.
   a. A l'aide de lignes de code, ajoutez des points à la grille et validez que son état est correct via sa fonction d'affichage.

b. A l'aide de lignes de code, ajoutez des points à la grille pour créer une combinaison gagnante horizontalement. Validez que la fonction de vérification de gagnant détecte bien cette victoire.
c. Idem pour une combinaison verticale.
d. Idem pour une combinaison diagonale. En particulier, vérifier que cela fonctionne pour des diagonales pour lesquels un des points est un des quatre coins de la grille.

C. Il est temps de jouer. Construisez une fonction de jeu.
   a. Cette fonction demande alternativement à deux joueurs d'insérer un pion de leur couleur. La fonction `console.readln()` retourne la chaîne de caractère rentrée par l'utilisateur. La fonction `parseInt(s)` retourne le `Number` équivalent à la chaîne de caractère passée en paramètre, ou `null` si la conversion n'a pas pu se faire (chaîne invalide).
   b. Après chaque pion posé, la fonction regarde s'il y a une victoire et l'affiche le cas échéant : le jeu est terminé. S'il n'y a pas de victoire, la fonction regarde si la grille est pleine. Dans ce cas, c'est une égalité et le jeu est aussi terminé. S'il n'y a ni victoire ni égalité, le jeu boucle en passant au joueur suivant.

Ex 4.

Jouons à la bataille navale : https://fr.wikipedia.org/wiki/Bataille_navale_(jeu)

Cette fois l'énoncé sera moins directif, à vous de voir ce qui doit être fait pour que cela fonctionne !

A. Créez une pseudo classe grille composée de 10 lignes et 10 colonnes.
   a. Ajoutez les fonctions nécessaires à y placer les bateaux (4 bateaux de 2 cases, 3 de 3 cases, 2 de 4 cases et 1 de 5 cases).
   b. Ajoutez les fonctions permettant de tirer sur les bateaux.
   c. Ajoutez les fonctions permettant d'afficher l'état d'une grille. Une fonction affiche la grille du joueur qui voit ses propres bateaux, tandis que l'autre fonction n'affiche que les coups qu'il a joués et s'ils ont touchés ou pas.
   d. Ajoutez une fonction permettant de placer les bateaux aléatoirement. Pour information, `Math.random()` retourne un nombre compris en 0 et 1 exclus.
   e. Créez une fonction déterminant s'il reste des bateaux vivants sur la grille.
B. Testez votre grille.
C. Construisez une fonction de jeu.
   a. Cette fonction instancie une grille pour l'ordinateur et une grille pour le joueur. Elle tire au hasard des positions pour les bateaux des deux grilles.
   b. A chaque tour, les deux grilles sont affichées ; celle de l'ordinateur n'affiche que les coups joués par le joueur, tandis que l'autre affiche aussi ses propres bateaux.
   c. Quand c'est son tour, le joueur rentre la ligne puis ensuite il rentre la colonne du coup qu'il joue. L'ordinateur lui tire au hasard.
   d. A chaque tour, il faut vérifier si un des joueurs a gagné et ne continuer que tant que ce n'est pas le cas.