TP et labo de la semaine 8 + solution

1) Correction de l'exercice noté #4.

Voici la correction de l'exercice noté #4. Il y a évidemment plusieurs variantes possibles.

J'en donne quelques-unes pour vous montrer différentes approches acceptables dans le cadre de ce travail.

La présence de commentaires utiles, la bonne indentation du code, l'utilisation d'identificateurs appropriés, et l'absence de calculs redondants sont des critères d'évaluation utilisés dans ce travail

La valeur imprimée pouvait se trouver entre 3.1415426535898203 et 3.1415426585893247

(dépendamment de la façon de sommer les termes).

```
var n = 20000;  // nombre de termes à sommer
var pi = 0;     // approximation de pi

for (var i=0; i<n; i++) {
    pi += Math.pow(-1,i) * 4/(i*2+1); // terme à ajouter
}
print(pi);
```

```
var n = 20000;  // nombre de termes à sommer
var pi = 0;     // approximation de pi

for (var i=0; i<n; i++) {
    var terme = 4/(i*2+1); // terme à ajouter ou retirer
    if (i % 2 == 0) {
        pi += terme;
    } else {
        pi -= terme;
    }
}
print(pi);
```

```
var n = 20000;  // nombre de termes à sommer
var pi = 0;     // approximation de pi

for (var i=1; i<n*2; i+=2) {
    var terme = 4/i; // terme à ajouter ou retirer
    if (i % 4 == 1) {
        pi += terme;
```

```
    } else {
        pi -= terme;
    }
}
print(pi);

var n = 20000;  // nombre de termes à sommer (doit être pair)
var pi = 0;     // approximation de pi

for (var i=1; i<n*2; i+=4) {

    pi += 4/i; // terme à ajouter

}
for (var i=3; i<n*2; i+=4) {

    pi -= 4/i; // terme à retirer

}
print(pi);

var n = 20000;  // nombre de termes à sommer (doit être pair)
var pi = 0;     // approximation de pi

for (var i=1; i<n*2; i+=4) {

    pi += 4/i - 4/(i+2); // faire 2 termes à la fois

}
print(pi);

var n = 20000;  // nombre de termes à sommer (doit être pair)
var pi = 0;     // approximation de pi

for (var i=n*2+1; i>0; i-=4) { // boucle descendante plus précise

    pi += 4/i - 4/(i+2); // faire 2 termes à la fois

}
print(pi);
```

a) Expliquer ce qu'affiche le code suivant :

```
var x = 10;
var test = 0;

var fct = function(x) {
    var test = 50;

    x += 1;
```

```
    print(x);

    return x;
};

print("x=" + x);
print("test=" + test);

print("la fonction a retourné : " + fct(x));
print("x=" + x);
print("test=" + test);
```

Solution :

Tester dans codeBoot pour constater que `x` et `test` ne sont pas changés, puisque ce sont des variables locales à `fct()`

b) Que se passe-t-il lorsqu'on enlève le "var" devant la variable "`test`" dans la fonction ?

Solution : La variable `test` est considéré comme étant la variable globale définie plus tôt et la valeur de `test` est alors changée.

c) Expliquer le code suivant :

```
var x = 10;
var y = 20;

var afficherSomme = function(x, y) {
    print(x + " plus " + y + " = " + (x + y));
};

print(afficherSomme(x, y));
print(afficherSomme(y, x));
```

Solution :

Lancer le code dans codeBoot et constater que les variables "`x`" et "`y`" ne sont pas liés aux variables passées en paramètres, mais seulement à l'ordre des paramètres.

3) Écriture de fonctions

On veut écrire une fonction permettant de calculer le nombre de k-combinaisons dans un ensemble de n valeurs, qui est généralement notée mathématiquement comme:

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$$

Le point d'exclamation désigne la factorielle: n! est le produit des entiers de 1 à n (inclus). Sauf dans le cas où n vaut 0, dans ce cas 0! vaut 1.

- Comment décomposeriez-vous ce problème en l'écriture de fonctions?
- Écrivez les fonctions permettant de faire ces calculs, avec leurs tests unitaires.

Solution :

```
var factorielle = function(x) {
    var resultat = 1;
    for(var i=1; i<=x; i++) {
        resultat *= i;
    }
    return resultat;
}
var combinaisons = function(x, k) {
    return factorielle(x) / (factorielle(x - k) * factorielle(k));
}
```