TD 2 : Fonctions

Elément de Cours : Fonctions

Comme nous l'avons vu, Octave permet d'écrire des scripts mais aussi **des fonctions définies par l'utilisateur** qui permet d'enrichir la bibliothèque Octave. Ces fonctions peuvent être réutilisées en ligne de commande ou encore dans des scripts Octave.

Dans les fonctions Octave, les arguments sont toujours passés par **valeur** et non par ***référence*** (cela signifie **que les valeurs passées par argument ne seront pas modifiées par la fonction**). Autre aspect important, Les fonctions dans Octave peuvent retourner plus d'une valeur.

En général, une fonction est définie dans un fichier texte (tout comme les scripts). La différence essentielle consiste en l'écriture de la première ligne d'une fonction qui s'écrit de la manière suivante :

```
function [sortie1, sortie2, …] = nom_fonction(entree1, entree2, …)
```

Chaque fonction est stockée dans un fichier .m différent où le nom du .m est le même que la fonction définie dans le fichier. Définir une fonction permet de rendre du code plus lisible et éviter de recopier plusieurs fois la même suite d'instructions.

Exemple :

Cosinus renvoyé en degrés (et non en radians comme par défaut)

```
function c= cosd(x)

%COSD(X) CALCULE COS(X) AVEC X EXPRIME EN DEGRES

      c = cos(x*pi/180)

end
```

Exercice 1 : Extraction de fonctions

Regardez l'algorithme ci-dessous :

```
a = [1 2 3; 4 5 6; 7 8 9]
b = [154 59 548; 489 456 55; 12 23 7]
c = [1.00 5.00 56.78; 19.6 121.45 78.0; 45.1 39.99 23.95]
mat = a;
a(1,2) = mat(2,1);
a(1,3) = mat(3,1);
a(2,1) = mat(1,2);
a(2,3) = mat(3,2);
a(3,1) = mat(1,3);
a(3,2) = mat(2,3);
mat = b;
b(1,2) = mat(2,1);
b(1,3) = mat(3,1);
b(2,1) = mat(1,2);
b(2,3) = mat(3,2);
b(3,1) = mat(1,3);
b(3,2) = mat(2,3);
```

Questions

1. Que fait cet algorithme ?
2. Une opération est répétée plusieurs fois, laquelle ? Extrayez cette opération et transformez-la en fonction
3. Réécrivez l'algorithme ci-dessus en utilisant la fonction que vous aurez préalablement défini.

Exercice 2 : Portée des variables

Soit la fonction suivante :

division.m

```
function d = division(a, b)
      if(b != 0)
            d = a/b;
      else
            d = NaN ; % NaN = Not a Number
      end
end
```

Ci-dessous un exemple d'utilisation de la fonction division :

script.m

```
a = 5;
b = 0;
c = 30;
d = 6;
res = 0;
division(a,b);
printf("d est egal a %d\n",d);
```

```
printf("res est egal a %d\n",res);

b = 9;
division(a,b);
printf("d est egal a %d\n",d);
printf("res est egal a %d\n",res);
res = division(c,d);
if(d == res)
      disp("d == res");
else
      disp("d != res")
end
```

Questions

1. Déterminez ce que ce programme affiche à l'écran.
2. Qu'en déduisez-vous sur l'équivalence entre les variables utilisées à l'intérieur de la fonction et celles utilisées à l'extérieur de la fonction (lors de son utilisation) ?

Exercice 3 : Simulation d'un lancer de dés !

Ecrivez une fonction qui lorsqu'elle est appelée renvoie de manière aléatoire un nombre compris entre 1 et 6.

Note : La fonction **rand()** en GNU/Octave permet de générer des nombres aléatoires compris entre 0 et 1 (non compris)

Modifiez la fonction afin que l'on puisse tirer aléatoirement un nombre entre et n (n = nombre de faces du dé)

Exercice 4 : Min,Max

Programmez une fonction *minmax* qui prend en entrée un tableau de valeurs et qui renvoie le min et le max de ces nombres. Proposez une utilisation de votre fonction montrant plusieurs cas possibles.

Exercice 5 : Factorielle récursive

En informatique, une fonction est dite récursive lorsque qu'elle s'appelle elle-même.

Ecrire la fonction **rfacto(n)** qui calcule la fonction factorielle sous forme récursive