Matlab :

Ce fichier est préparé par **Compil'Court** d'ENSA Agadir.
$\forall$ error found $\in$ doc : contact us on discord.
Let's make ENSA AGADIR great again !

Dans cette partie, nous verrons l'utilisation des fonctions dans Matlab. Nous verrons comment écrire une fonction dans Matlab, ainsi que comment appeler une fonction crée dans notre programme.

Écrire des fonctions simples :

Pour écrire une fonction dans Matlab, la première règle à respecter est de donner au fichier .m le même nom que la fonction que l'on est en train d'écrire. Par exemple, une fonction qui s'appellerait mafact devra être écrite dans le fichier `mafact.m`
Pour écrire, une fonction dans Matlab, on doit d'abord donner le noms des valeurs en sortie générées par la fonction, puis le nom de la fonction, et enfin le noms des paramètres en entrée de la fonction :

```
function [sortie1,..,sortieN] = nom_fonction(entree1,..,entreeN)
        instruction1
        instruction2
end
```

Par exemple, si on souhaite faire une fonction produit, qui calcule et renvoie en sortie le produit de deux scalaires passés en paramètre, on écrira dans le fichier `produit.m` :

```
function [res] = produit(a,b)
res = a*b;
end
```

Pour tester cette fonction, placez-vous (à l'aide de l'explorateur de fichiers Matlab) dans le répertoire qui contient le fichier `produit.m` et on écrira dans la fenêtre de commande :

```
>> e = produit(2,4);
e =
     12
```

Une fois cette commande exécutée, on récupère bien dans `e` la valeur de sortie de la fonction, c'est à dire le produit des deux paramètres entrée. On remarquera que l'on ne voit pas, dans la zone des variables de Matlab, une variable nommée `res`. En effet, la variable `res` est interne à la fonction produit, et ne vit que pendant l'appel de cette fonction. Une fois la fonction terminée, la variable est effacée. De plus, la variable `res` n'existe que pour la fonction produit : elle ne peut pas être lue par une autre fonction ou par une commande directement dans Matlab. On récupère la valeur de `res` en la déclarant comme une valeur de sortie de la fonction, et en récupérant cette sortie dans la variable `e`. Considérez cette autre fonction :

```
function [res] = somme
res = a+b;
end
```

Cette fois-ci, on appellera cette fonction ainsi :

```
a=2;
b=4;
e = somme();
```

Cette syntaxe ne fonctionne pas : même s'il existe des variables a et b dans les variables de Matlab, ce ne sont pas des variables internes 0 la fonction somme. Or, une fonction ne peut pas lire les variables de Matlab, elle ne connait que ses variables internes et les variables passées en paramètre. Voilà pourquoi,dans la fonction `produit.m`, on donnait les valeurs à multiplier comme des paramètres d'entrée de la fonction : c'est le seul moyen de transmettre à la fonction une ou plusieurs valeurs.
Si on récapitule, on doit, pour écrire une fonction, décider de plusieurs éléments :

1. Un nom de fonction, qui devra être aussi le nom du fichier dans lequel sera écrit la fonction.
2. Des paramètres d'entrée de la fonction, qui seront les valeurs des éléments dont aura besoin la fonction pour réaliser son calcul, et qu'elle ne pourra pas décider d'elle-même. Il est important de comprendre que la fonction ne peut pas lire les variables de la zone des variable de Matlab : une fonction ne connait que ses variables internes, c'est à dire celles déclarées en paramètres d'entrée ou déclarées dans le code même de la fonction.
3. Des paramètres en sortie de la fonction, qui seront les valeurs calculées par la fonction, et que l'on souhaite récupérer une fois la fonction terminée. Si vous regardez bien l'appel que vous avez fait de la fonction produit juste avant, vous verrez que la variable `res` interne à la fonction n'est pas dans la zone des variables de Matlab. En effet, une fois la fonction terminée, toutes ses variables internes sont détruites. Ici, on récupère le résultat du calcul grâce à la variable `e` qui n'est pas interne à la fonction (elle n'est pas déclarée dans la fonction) et qui est utilisée, lors de l'appel à la fonction, comme une variable de sortie.

Exemples :

1. Écrire une fonction qui prend en argument 5 nombres, et renvoie leur somme, la moyenne, et puis les ordonne en ordre croissant.

```
function [somme, moyenne, ordre] = exemple1(a1, a2, a3, a4, a5)
        somme=a1+a2+a3+a4+a5;
        moyenne=somme/5;
        ordre=[a1, a2, a3, a4, a5];
        for i=1:4
                for j=(i+1):5
                        if ordre(i)>ordre(j)
                                temp=ordre(j);
                                ordre(j)=ordre(i);
                                ordre(i)=temp;
                        end
                end
        end
end
```

Après avoir enregistré la fonction en nom `exemple1.m`, on exécute la fonction ainsi :

```
>> [arg1, arg2, arg3]=exemple1(213, 342, 532, 51, 429)
arg1=
                1567
arg2=
                313.4000
arg3=
                51  213  342  429  532
```

2. Écrire une fonction qui prend en argument le rang n de la suite de Fibonacci et renvoie en sortie les valeurs du 1^e terme jusqu'au U_n (voir manipulation 6 en TP2)

```
function [U] = fibonacci(n)
        U(1)=1;
        U(2)=1;
        if n==1
                U=[1];
        end
        for i=3:n
                U(i)=U(i-1)+U(i-2);
        end
end
```

3. Écrire une fonction qui reçoit en entrée une chaine de caractère et un symbole, et renvoie en sortie le nombre d'occurrence de ce symbole dans la chaine.

```
function [nbr] = occurence( chaine, symbole )
        nbr=0;
        j=length(chaine);
        for i=1:j
                if strcmp(chaine(i),symbole)==0
                        nbr=nbr+1;
                end
        end
end
```