1. Introduction

Une procédure est une donnée Maple constituée d'une suite d'instructions servant à effectuer une tâche spécifique. Comme tous les objets Maple, une procédure peut être désignée par un nom.
Une fonction est une procédure particulière (plus simple). Attention à ne pas confondre le terme fonction au sens de Maple et au sens mathématique. Une "fonction Maple" par exemple n'a pas d'espace de départ. Je déconseille l'utilisation des "fonctions Maple" pour représenter des fonctions mathématiques. Utiliser simplement des expressions.
Il faut bien distiguer la *définition* et les *appels* d'une procédure.
La définition est une assignation à un nom (le nom choisi pour la procédure) des instructions qui la constituent.
Un appel est une instruction servant à faire exécuter à la procédure la tâche pour laquelle elle a été écrite et à lui indiquer les valeurs des paramètres qu'elle utilise.
À côté des noms réservés par Maple pour désigner des objets mathématiques usuels (`Pi 1 2 3 ...`) figurent des noms de procédures (`subs solve irem evalf ...`)
Par exemple, la commande

```
> cos(Pi);
```

est un appel de la procédure `cos`.
À chaque appel, une procédure fonctionne de manière autonome. Cette autonomie se traduit par un domaine spécifique pour la validité des noms (on dira *local*). Autrement dit, elle dispose de son propre système d'état-civil, le même nom (par exemple `i`) peut désigner des objets differents dans des procédures et dans le programme qui les appelle (comme un même prénom désigne des personnes différentes dans des familles différentes). À l'inverse, on peut déclarer dans la définition d'une procédure un nom de variable comme *global* pour qu'il désigne le même objet que dans l'instruction qui l'appelle.
Une procédure doit pouvoir échanger des informations avec le système qui l'a appelée ainsi qu'avec l'extérieur.
Pour faire entrer les données dans la procédure, on peut utiliser les *paramètres* ou les *variables globales*. Pour faire sortir les données on peut utiliser le retour (ce que renvoie la fonction), les variables globales ou une interface vers un dispositif extérieur (le plus souvent l'écran mais éventuellement une imprimante ou un fichier).
Toutes les procédures que nous écrirons passent les *valeurs* des paramètres et non les *références*[1]. La syntaxe Maple pour le passage des paramètres par référence n'entre pas dans le cadre de ce cours.
Pour Maple, le *retour* est le résultat de la dernière instruction de la procédure. On peut aussi utiliser `return` pour renvoyer un résultat et c'est une bonne pratique de le faire systématiquement.

[1] Il est possible de faire passer les références de paramètres. Par exemple les procédures de la bibliothèque de géométrie le font assez systématiquement.

2. Syntaxe d'appel

Le nom de la procédure est suivie par des parenthèses contenant une séquence de paramètres. Par exemple pour appeler la procédure `machin` avec le paramètre `bidule` :

```
> machin(bidule);
```

Ici Maple ne connait ni l'un ni l'autre et affiche seulement l'appel sans rien faire. Les nom `subs solve irem evalf` sont connus de Maple et désignent des procédures. Pour d'autres procédures plus spécifiques les noms ne sont pas chargés automatiquement en mémoire. Elles sont stockées dans des *bibliothèques* qu'il est nécessaire d'appeler à l'aide de `with`.

```
> plot(cos,-Pi..Pi);
  plot([[0,0],[1,0],[1,1],[0,0]],axes=none, color=blue);
  desscos:=plot(cos,-Pi..Pi,scaling=constrained):
  desstri:=plot([[0,0],[1,0],[1,1],[0,0]]
                ,axes=none, color=blue,scaling=constrained):
  display([desscos,desstri]);
  with(plots);
  display([desscos,desstri]);
  with(geometry);
  point(pA,[0,0]):point(pB,[0,1]):point(pC,[1,0]):
  draw([pA,pB,pC],axes=none);
  triangle(tri,[pA,pB,pC]):
  circumcircle(circ,tri):
  draw([pA,pB,pC,circ(color=blue)],axes=none);
```

Les instructions précédentes utilisent la bibliothèque `plots` et fournissent aussi des exemples de retour.

3. Syntaxe de définition

1. Pour une procédure

```
> nomchoisi := proc(para1,para2,...)
```

```
    local varloc1,varloc2,...;
    global varglob1,varglob2,...;
    instruction1;
      ...
    derninstruction;
  end proc;
```

Exemples de définitions :

```
> factorielle:= proc(n)
    local i,resultat;
    resultat :=1;
    for i from 1 to n do
      resultat := resultat*i;
    od;
    resultat;  # ou return resultat;
  end proc;

  factoriellebis:= proc(n)
    local i,resultat;
    resultat :=1;
    for i from 1 to n do
      resultat := resultat*i;
    od;
    print(resultat);
  end proc;

  #appels
  factorielle(5);factoriellebis(5);
  f:=factorielle(5): ff:=factoriellebis(5);
  f;ff;
```

Il ne faut pas confondre *renvoyer* et *afficher*. Lorsque la définition se termine par un `print`, la procédure *ne renvoie rien* et affiche quelque chose. Par défaut, une procédure renvoie le résultat de l'évaluation de la dernière commande. Si cette commande est un `print`, il n'y a pas d'évaluation, il y a *affichage* sur l'écran.
Je conseille d'utiliser systématiquement `return expr;`. Cette commande termine l'appel de la procédure et renvoie l'évaluation de `expr`.
Si on veut que la procédure renvoie plusieurs objets, il faut en faire une séquence puis l' *empaqueter* dans un seul objet comme une liste ou un tableau.

2. Pour une fonction

```
 nomchoisi := (para1,para2,...) -> expr des paramètres;
```

la fonction renvoie l'évaluation de l'expression

```
> cbin:= (m,n)->mul(m-i,i=0..n-1)/n!;
  cbin(4,2);
```

4. Exemples

Définir une procédure `cle` telle que l'appel `cle(x,A)` (où `A` est le nom d'un tableau de dimension 2) renvoie

- la liste `[0,0]` si l'évaluation de `x` ne figure pas parmi les valeurs de `A`
- une liste `[i,j]` (n'importe laquelle) telle que la valeur du tableau associée soit égale à l'évaluation de `x`.

Modifier la procédure précédente pour définir `cle1` utilisant une variable globale `bonnecle`. Un appel `cle1(x,A)` ne renvoie rien, affiche `"trouvé"` ou `"pas trouvé"` et assigne à `bonnecle` la liste `[i,j]` comme pour la procédure précédente.

Définir une procédure qui "retourne" un tableau `A` à une dimension.
Si `A` désigne un tableau de valeurs `[1,2,3,truc,4]` , l'appel `retourne(A)` doit modifier le tableau désigné par `A`. Ses valeurs doivent être `[4,truc,3,2,1]` après l'appel.
Dans cette définition, vous devez utiliser

- des variables globales `A` et `n` pour le tableau `array(1..n)`.
- une variable locale `i` qui prendra des valeurs entières.
- une variable locale "à tout faire" `faitout`

et rien d'autre.