

# Partie 7

## Les Fonctions et procédures

---

### 1/ Les Fonctions Prédéfinies

Tout langage de programmation propose un certain nombre de **fonctions** ; elles servent à soulager le programmeur, en lui épargnant de longs et pénibles algorithmes.

Exemple : le calcul du sinus d'un angle. Pour en obtenir une valeur approchée, il faudrait appliquer une formule très complexe. Aussi, sur les calculatrices, on nous fournit quelques touches spéciales, dites **touches de fonctions**, qui vous permettent par exemple de connaître immédiatement ce résultat. Sur votre calculatrice, si vous voulez connaître le sinus de 35°, vous taperez 35, puis la touche SIN, et vous aurez le résultat. C'est exactement le même principe pour une fonction prédéfinie d'un langage de programmation.

*Ecriture algorithmique :*

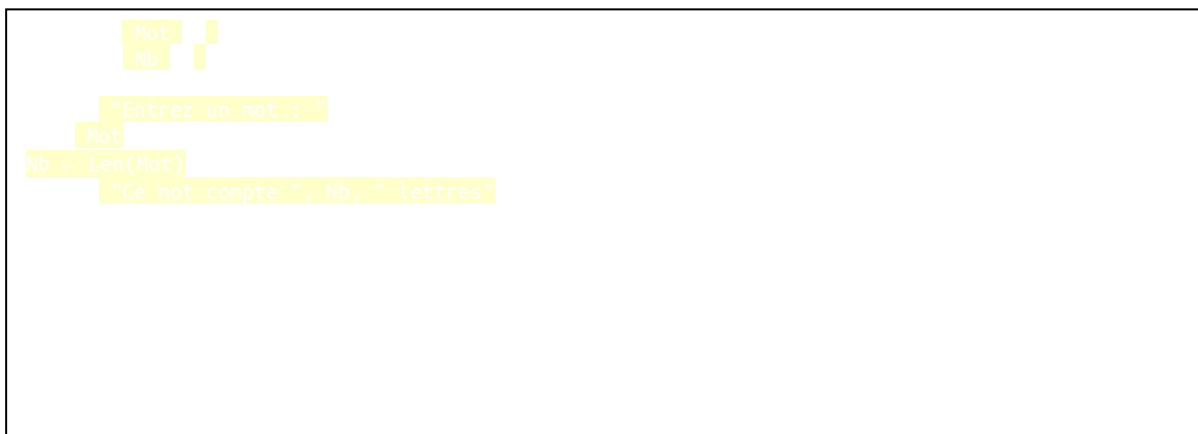
**A ← Sin(25)**

Une fonction est donc constituée de trois parties :

- le **nom** proprement dit de la fonction. Dans notre exemple, ce nom est **Sin**.
- **deux parenthèses**, une ouvrante, une fermante. Ces parenthèses sont toujours obligatoires, même lorsqu'on n'écrit rien à l'intérieur.
- une **liste de valeurs**, indispensables à la bonne exécution de la fonction. Ces valeurs s'appellent des **arguments, ou des paramètres**. Le nombre d'arguments nécessaire pour une fonction donnée ne s'invente pas : il est fixé par le langage. Par exemple, la fonction sinus a besoin d'un argument. Si vous essayez de l'exécuter en lui donnant deux arguments, ou aucun, cela déclenchera une erreur à l'exécution. Les arguments doivent être d'un certain type, il faut respecter ces types.

#### Exercice 7.1 :

Ecrivez un algorithme qui demande un mot à l'utilisateur et qui affiche à l'écran le nombre de lettres de ce mot . On dispose de la fonction len() qui prend en argument une variable de type chaîne de caractères et qui renvoie le nombre de caractères de cette chaîne.



#### Quelques catégories de fonctions prédéfinies :

- **les fonctions mathématiques**, à ne pas confondre avec les opérateurs mathématiques de base ( sin, cos, racine, log, etc...)
- **les fonctions de texte**, qui manipulent les chaînes de caractères ( len(), et tout une artillerie selon le langage)
- **les fonctions de conversion** (pour convertir un type en un autre)

## 2 - Fonctions personnalisées

Une application, surtout si elle est longue, a toutes les chances de devoir procéder aux mêmes traitements, ou à des traitements similaires, à plusieurs endroits de son déroulement. Par exemple, la saisie d'une réponse par oui ou par non (et le contrôle qu'elle implique), peuvent être répétés dix fois à des moments différents de la même application, pour dix questions différentes.

Il faut séparer ce traitement du corps du programme et regrouper les instructions qui le composent en un module séparé. Il ne restera alors plus qu'à appeler ce groupe d'instructions (qui n'existe donc désormais qu'en un exemplaire unique) à chaque fois qu'on en a besoin.

Le corps du programme s'appelle alors la **procédure principale**, et ces groupes d'instructions auxquels on a recours s'appellent des **fonctions** et des **sous-procédures**.

Prenons un exemple de question à laquelle l'utilisateur doit répondre par oui ou par non :

*Mauvaise Structure :*

```
...
Ecrire "Etes-vous marié ?"
Lire Rep1
TantQue Rep1 <> "Oui" et Rep1 <> "Non"
  Ecrire "Tapez Oui ou Non"
  Lire Rep1
FinTantQue
...
Ecrire "Avez-vous des enfants ?"
Lire Rep2
TantQue Rep2 <> "Oui" et Rep2 <> "Non"
  Ecrire "Tapez Oui ou Non"
  Lire Rep2
FinTantQue
...
```

*Bonne Structure :*

```
Fonction RepOuiNon() en caractère
Lire Truc
TantQue Truc <> "Oui" et Truc <> "Non"
  Ecrire "Tapez Oui ou Non"
  Lire Truc
FinTantQue
Renvoyer Truc
Fin

...
Ecrire "Etes-vous marié ?"
Rep1 ← RepOuiNon()
...
Ecrire "Avez-vous des enfants ?"
Rep2 ← RepOuiNon()
...
```

## Passage d'arguments

On va améliorer notre fonction en lui « passant » le libellé à afficher. Il s'agira ici de lui passer cette chaîne de caractères en argument :

```
Fonction RepOuiNon(Msg en Caractère) en Caractère
Ecrire Msg
Truc ← ""
TantQue Truc <> "Oui" et Truc <> "Non"
  Ecrire "Tapez Oui ou Non"
  Lire Truc
FinTantQue
Renvoyer Truc
Fin Fonction
```

*RQ : On n'a passé qu'un seul argument en entrée. Mais bien entendu, on peut en passer autant qu'on veut, et créer des fonctions avec deux, trois, quatre, etc. arguments*

```
...
Rep1 ← RepOuiNon("Etes-vous marié ?")
...
Rep2 ← RepOuiNon("Avez-vous des enfants ?")
...
```

## 3- Sous-Procédures

Une sous-procédure est un genre de fonction qui ne **renvoie rien**. C'est un bout de programme qu'on peut appeler dans la procédure principale et qui fera son petit boulot dans son coin sans avoir besoin d'informer la procédure principale de ce qu'elle a accompli. Exemple : afficher un texte, rafraîchir un écran...

Voici comment se présente une sous-procédure :

```
Procédure Bidule( ... )
...
Fin Procédure
```

Dans la procédure principale, l'appel à la sous-procédure Bidule devient quant à lui :

```
Appeler Bidule(...)
```

## 4 - Portée d'une variable, passage par valeur.

**! ATTENTION : PARTIE PEU SIMPLE A COMPRENDRE...MAIS ESSENTIEL A COMPRENDRE !**

Lors de l'appel d'une fonction, les arguments transmis ne sont jamais des variables mais sont **les valeurs de ces variables**.

Exemple : Voici ci-dessous une fonction *Plus4* qui fait un + 4.

```
Fonction Plus4(a en Numérique) en Numérique
a = a+4
Renvoyer a
Fin Fonction

...

Nbr = 6
Rep1 ← Plus4(Nbr)
Ecrire Rep1
Ecrire "la valeur de a est : "
Ecrire a
...
```

Dans la définition de ma fonction, j'ai nommé cet argument a. Ce qu'il faut comprendre, c'est que je n'ai nullement besoin de lui envoyer ma valeur numérique via une variable du même nom !  
Lorsque j'appelle ma fonction, ce n'est pas Nbr que je lui envoie, c'est la **valeur de Nbr** !  
De la même manière, lorsque ma fonction renvoie le résultat, ce n'est pas a qu'elle renvoie, mais la **valeur de a** !

**! LE PASSAGE D'ARGUMENT SE FAIT PAR VALEUR !**

### Portée d'une variable :

Regardons ce programme où nous avons ajouté une ligne dans la procédure principale et intéressons nous plus particulièrement à la variable a

```
Fonction Plus4(a en Numérique) en Numérique
a = a+4
Renvoyer a
Fin Fonction

...

Nbr = 6
Rep1 ← Plus4(Nbr)
Ecrire Rep1
Ecrire "la valeur de a est : "
Ecrire a
...
```

Si on a compris ce qui précédait dans le cours, on comprend sans peine que **Rep1** va prendre la valeur **10**, et qu'on va afficher sans peine cette valeur à l'écran. Quand est-il de **a** ? Va-t-on afficher la **valeur de a est : 10** ?

**NON**, car **a** n'est pas à la portée de la procédure principale ! Elle est localisée dans la fonction, elle n'est connue QUE **localement** dans la fonction !

*Rq : en Python, une fonction est capable d'aller lire la valeur d'une variable qui est définie dans la procédure principale. Elle est par contre incapable de la modifier !*

Exemple :

**Ceci fonctionne :**

```
def toto():
    print(b)
...
b=4
toto()
```

**Ceci ne fonctionne pas :**

```
def toto():
    b=b+4
...
b=4
toto()
```

Nous verrons plus tard qu'il est possible de faire ce qu'on appelle un **passage par référence**, qui permettra de manipuler une variable qui n'est pas locale à la fonction.

## Exercice 7.2

Écrivez une fonction qui renvoie la somme de cinq nombres fournis en argument.

```
Sum(a, b, c, d, e)  
a + b + c + d + e
```

## Exercice 7.3

Ecrire un traitement qui informe si un tableau envoyé en argument est formé ou non d'éléments tous rangés en ordre croissant.

```
TableauCroissant(T() en Numérique, n en  
Numérique)  
i  
Flag  
  
Flag ← Vrai  
i ← 0  
Flag i < n-1  
Flag ← T(i) < T(i+1)  
i ← i+1  
  
Flag
```

## Exercice 7.4

Ecrire un traitement qui inverse le contenu de deux valeurs passées en argument.