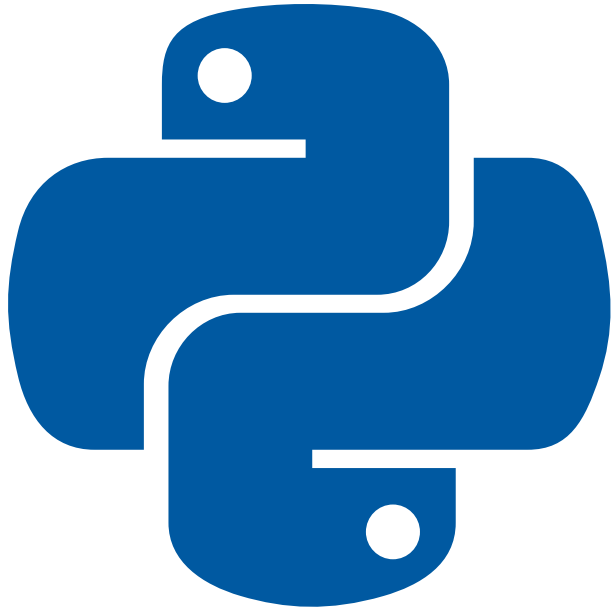
Les fonctions et les procédures

Les bases de l'algorithmiques

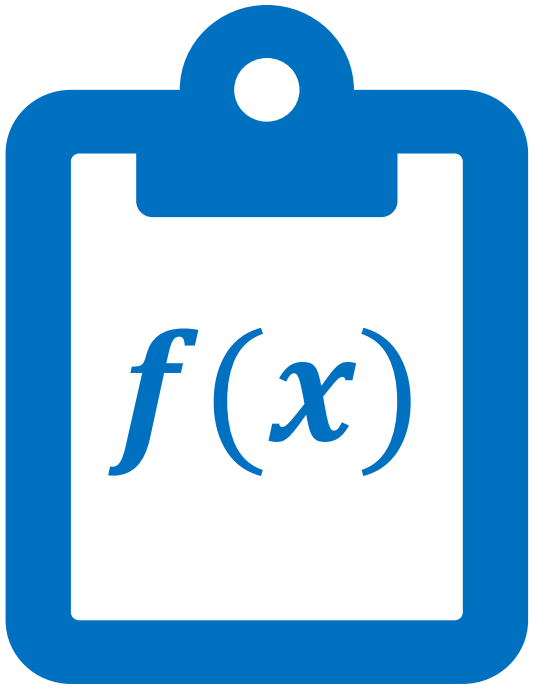
Réalisation : Omar OUGHZAL



Fonctions prédéfinies (Standards)

Tout langage de programmation dispose d'un ensemble de fonctions prédéfinies qui permettent de réaliser tout sorte d'opération :

- Conversion de type
- Calcul mathématique
- Manipulation de date et heure
- Manipulation de chaînes de caractères
- Et bien d'autres fonctions utiles



Fonctions mathématiques

- Les fonctions mathématiques prédéfinies permettent la réalisation d'un traitement mathématique sur des données numériques

```
Algorithme MathFonctions
Const pi = 3.14159
Début
    Ecrireln("sin(pi/6) = ",sin(pi/6))
    Ecrireln("cos(pi/3) = ",cos(pi/3))
    Ecrireln("tan(pi/3) = ",tan(pi/4))
    Ecrireln("|-6| = ", abs(-6))
    Ecrireln("Arrondi(3.22) = ",arrondi(3.22))
    Ecrireln("Racine(9) = ", racine(9))
    Ecrireln("nombre Aléatoire entre 1,9 : ", Alea(1,9))
Fin
```

Fonction de chaîne de caractères

- Une chaîne est une séquence de caractères dont la longueur correspond au nombre de caractères qu'elle contient. Si une chaîne est vide, sa longueur est égale à zéro
- **Long(CH)** : retourne le nombre de caractères de la chaîne CH
  Long("Maroc") → 5
- **Convch(nb)** : retourne la conversion du nombre nb en chaîne
  Convch(123) → "123"
- **Val(CH)** : retourne la conversion da la chaîne CH à un nombre
  Val("456") → 456
- **Pos(CH1,CH2)** : retourne la première position de CH1 dans CH2
  Pos("ar","Maroc") → 1
- **Chr(N)** : retourne le caractère du code ASCII N
  Ch(65) →'A'
- **Asc(C)** : retourne le code ASCII du caractère C
  Asc('C') → 67

Programmation structurée

Un long algorithme de plus deux pages est difficile à comprendre et à gérer :

- Difficile à écrire et interpréter
- Difficile à maintenir
- Des séquences de code qui se répètent à plusieurs endroits

La solution est d'utiliser la programmation structurée :

- L'idée découpage d'un problème en des sous-problèmes moins complexes
- Un sous-problème peut être découper en sous-problème si nécessaire
- Chaque sous-problème est résolu avec un sous-algorithme (sous-programme)
- Il existe deux sortes de sous algorithmes : Les procédures et les fonctions

Avantages :

- Clarté de l'algorithme.
- Facilité de maintenance.
- Réutilisation des sous-algorithmes

Les procédures

- Une procédure est une série d'instructions regroupées sous un seul nom, qui permet d'effectuer des actions par un simple appel de la procédure dans un algorithme ou dans un autre sous algorithme.

Syntaxe :

```
Procédure Nom_Proce(Liste des paramètres)
Var nom: Type
Début
    Instructions
Fin
```

Après le nom de la procédure, il faut donner la liste des paramètres (s'il y en a) avec leurs types respectifs. Ces paramètres sont appelés **paramètres formels**. Leur valeur n'est pas connue lors de la création de la procédure.

Exemple

Ecrire une procédure qui qui affiche à l'écran une ligne de N étoiles.

```
procedure etoiles(N : Entier)
var i : Entier // variable locale
début
    pour i de 0 à N-1 Faire
        Ecrire("*")
    finpour
fin
```

L'appel d'une procédure

- Pour déclencher l'exécution d'une procédure dans un programme, il suffit de l'appeler (l'invoquer) avec ses paramètres entre parenthèses

```
Algorithme LesProcedures
var i: Entier
procedure etoiles(N : Entier)
début
    pour i de 0 à N-1 Faire
        Ecrire("*")
    finpour
    Ecrireln()
fin
Début
    etoiles(8)
    etoiles(10)
Fin
```

Exemple

- En utilisant la procédure Etoiles déclarée dans l'exemple précédent, Ecrire un algorithme permettant de dessiner un triangle d'étoiles de 10 lignes.

```
Algorithme LesProcedures
var i: Entier
procedure etoiles(N : Entier)
début
    pour i de 0 à 9 Faire
         etoiles(i)
     finpour
Fin
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Remarques

- Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de la partie principale (algorithme principal)
- Lors de la conception d'un algorithme deux aspects apparaissent :
    - **La définition** (déclaration) de la procédure ou fonction
    - **L'appel** (l'invocation) de la procédure ou la fonction
- Un sous-algorithme (procédure ou fonction) prend des données par l'intermédiaire de ses paramètres:
    - **Paramètres formels** : objets utilisés pour la description d'un sous-algorithme
    - **Paramètres effectifs** : objets utilisés lors de l'appel d'un sous-algorithme
    - Un paramètre formel est toujours une variable
    - Un paramètre effectif peut être : une variable, une constante, une expression, un appel de fonction

Passage des paramètres

Les échanges d'informations entre une procédure et le sous algorithme appelant se fait par l'intermédiaire de paramètres.

Il existe deux principaux types de passage de paramètres qui permettent des usages différents :

- **Par valeur** : le paramètre formel reçoit uniquement une copie de la valeur du paramètre effectif. La valeur du paramètre effectif ne sera jamais modifiée.
- **Par référence (adresse)** : le paramètre formel reçoit l'adresse du paramètre effectif. La valeur de la variable effective sera modifiée
- Les paramètres passés par adresse son précédés par le mot-clé **Var**.

Exemple

- Ecrire une procédure qui échange les valeurs de deux variables passées en paramètres

```
Algorithme EchangeValeurs
var a,b : Entier
procedure swap(var x :
Entier, var y:Entier)
var t : Entier
début
    t = a
    a = b
    b  = t
fin
Début
    a = 8
    b = 2
    swap(a,b)
    Ecrire("a :",a," ;
b:",b)
Fin
```

Les Fonctions

- Les fonctions sont des sous algorithmes admettant des paramètres et retournant un seul résultat (une seule valeur)

Syntaxe :

```
Fonction Nom_Fonc(Liste des paramètres) : Type
Var nom : type
Début
Instruction(s)
Retourner expression
Fin
```

La syntaxe de la déclaration d'une fonction est assez proche de celle à laquelle on ajoute un type qui représente le type de la valeur retournée par la fonction est instruction **Retourner expression**

Exemple

Définir une fonction qui revoie le plus grand entre deux nombres différents.

```
fonction max(a:Entier,b:Entier) : entier
début
    si a>b alors
        retourner a
    sinon
        retourner b
    FinSi
fin
```

L’appel d’une fonction

- Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivi des paramètres effectifs, c’est la même syntaxe qu’une procédure.
- A la différence d’une procédure, la fonction retourne une valeur. L’appel d’une fonction pourra être utilisé dans une instruction (Affichage, affectation, …) qui utilise sa valeur
- Syntaxe :

```
Nom_Fonc (Liste des paramètres)
```

Les Fonctions en python

```python
#suntaxe de défintion d'un fonction
def nom_de_la_fonction(param1, param2, ..., paramN):
    # corps de la fonction
    return resultat # éventuellement
```

Une fonction en Python est une séquence d'instructions nommée et qui peut être appelée dans le programme principal.

Paramètres de fonction

```
def ma_fonction(parametre1, parametre2, ...):
    # corps de la fonction
```

Les paramètres de fonction sont des valeurs qui peuvent être utilisées à l'intérieur d'une fonction. Les paramètres permettent à une fonction de recevoir des données en entrée, qui peuvent être utilisées pour effectuer des calculs.

Arguments de fonction

```
ma_fonction(argument1, argument2, ...)
```

Les arguments de fonction sont les valeurs réelles passées à une fonction lorsqu'elle est appelée. Les arguments peuvent être des constantes, des variables, des expressions, ou même d'autres fonctions.

Types de paramètres

```
def somme(a,b=3) : return a+b
somme(3,4) #positionnels
somme(b=4,a=2)  #keyword
somme(5) # b=3 par défaut
```

- **Paramètres positionnels** : ces paramètres sont passés à la fonction dans l'ordre dans lequel ils sont définis.
- **Paramètres par mot-clé** : ces paramètres sont identifiés par leur nom lorsqu'ils sont passés à la fonction. L'ordre dans lequel ils sont passés n'a pas d'importance.
- **Paramètres par défaut** : ces paramètres ont une valeur par défaut qui est utilisée si aucun argument n'est passé pour ce paramètre.

Exemple

```
def somme(a, b):
    return a + b
print(somme(3, 5)) # affiche 8
def afficher_infos(nom, age=30, ville="Paris"):
    print("Nom :", nom)
    print("Age :", age)
    print("Ville :", ville)
afficher_infos("Jean", 25) # informations de Jean
afficher_infos("Marie", ville="Lyon") # informations
de Marie
```