جامعة مولاي إسماعيل
UNIVERSITE MOULAY ISMAIL
كلية العلوم
FACULTE DES SCIENCES

Filière M.I.P – S2
Année Universitaire : 2024 – 2025

Chapitre 1
Les fonctions et les procédures

Filière MIP – S2

Module : Informatique 2 : Algorithmique 2/Python

Pr. Badraddine AGHOUTANE
b.aghoutane@umi.ac.ma

Plan du cours

- Rappel
- **Les fonctions et les procédures en python.**
- La récursivité et son application dans des algorithmes.
- Les enregistrements et les fichiers en Python.
- La complexité des algorithmes et ses principaux types.
- Les preuves de correction et de terminaison d'un algorithme.

Plan du chapitre 1

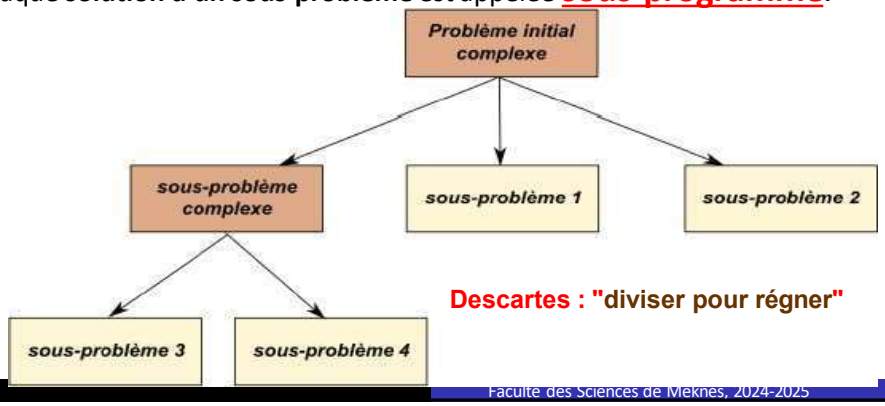
Introduction: programmation modulaire

Problème : Comment traiter les **problèmes complexes**?

Décomposer un problème donné en un ensemble fini des **sous-problèmes.**

- Pour chaque sous-problème *i* , on cherche une solution s_i .
- On regroupe l'ensemble des sous solutions pour aboutir à la résolution du problème initial complexe.

Chaque **solution d'un sous-problème** est appelée **sous-programme**.



Descartes : "diviser pour régner"

Fonctions en algorithmique

Fonction: syntaxe

Une **fonction** est un **sous-programme** qui permet d'apporter une solution à un **sous-problème.** Une **fonction** :

- A un nom
- Peut avoir des paramètres (arguments)
- Peut retourner une valeur d'un certain type (résultat)
- Peut avoir besoin de variables locales
- Est composé d'instructions (traitement)

Fonctions en algorithmique

Une **fonction** est un **sous-programme** qui permet d'apporter une solution à un **sous-problème.** Une **fonction** :

- A un nom
- Peut avoir des paramètres
- Peut retourner une valeur d'un certain type
- Peut avoir besoin de variables
- Est composé d'instructions

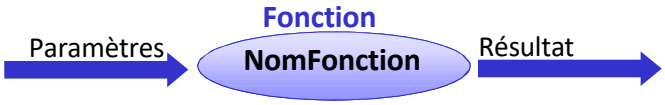
Syntaxe algorithmique d'une fonction :

```
Fonction   nomFonction (Paramètre1 : type1, ... ) :  typeRetour
    Variables   variableLocale : type
Début
        Instructions …
        retourner résultat
Fin
```

Fonctions en algorithmique

Fonction: syntaxe

Une **fonction** est un **sous-programme** qui admet des **paramètres** et retourne un **résultat :**



On distingue **3 étapes** importantes :

1. On envoie des **informations en entrée** à la fonction (paramètres)
2. La fonction exécute un **traitement (instructions)** à l'aide des informations qu'elle a reçu
3. Enfin, la fonction **renvoie un résultat en sortie**, on parle aussi de valeur de retour (return en anglais)

Pour appeler une **fonction**, il suffit d'écrire le nom de cette fonction :

NomFonction (liste paramètres)

Fonctions en algorithmique

La **définition** d'une fonction se fait au moyen de :

1. **En-tête de la fonction**
2. **Déclaration des variables**
3. **Corps de la fonction**

De la manière suivante :

```
Fonction nomFonction (Paramètre1 : type1, ... ) : typeRetour
    Variables  variableLocale : type
Début
        Instructions ...
        retourner résultat
Fin
```

En-tête

Déclarations des variables

Corps de la fonction

Fonctions en algorithmique : Définition

- **L'En-tête** (*appelé aussi* ***prototype*** *ou* ***signature***) est écrit ainsi :

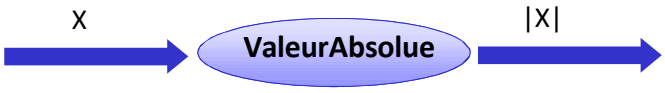
  fonction NomFonction (**NomArg_1:TypeArg_1,..., NomArg_k:TypeArg_k**) : **TypeRetour**

  Il est composé de :
  - **Nom de la fonction** (*NomFonction*), après le mot-clé **fonction**.
  - **Liste des paramètres** (*ou arguments*): **Optionnelle.** Si la liste est vide, on peut ne rien écrire ().
  - **Type du résultat** (*TypeRetour*). **Vide,** si la fonction ne retourne pas de résultat (dans ce cas, on parle de **procédure)**.

- **Déclarations des variables :** elles sont déclarées à l'intérieur de la fonction ➔ **Variables locales**
- **Corps de la fonction** : suite d'instructions, mises entre les mots-clé : **Début** et **Fin**.

Fonctions en algorithmique : Exemples

Définition d'une fonction (Valeur absolue) :




```
Fonction valeurAbsolue (X : Entier) : Entier
# Pas de variales locales
Début
        Si (X ≥ 0) alors
            retourner  X
        Sinon
            retourner  -X
        FinSi
Fin
```

En-tête

Déclarations des variables

Corps de la fonction

Fonctions en algorithmique : Exemples

Définition et appel d'une fonction

```
Algorithme Abs
Variable a : Entier
Fonction valeurAbsolue (X : Entier) : Entier
    Variable valeurAbsolue : Entier
    Début
        Si (X ≥ 0) alors
            valeurAbsolue ← X
        Sinon
            valeurAbsolue ← -X
        FinSi
        retourner valeurAbsolue
Fin
Début
a ← valeurAbsolue(-3)
Ecrire ("la valeur absolue de -3 est ", a)
Fin
```

Appel de la fonction

Fonctions en algorithmique : Exemples

Fonctions : appel de la fonction valeurAbsolue

```
Algorithme Abs
  Variable a, b : Entier
  Fonction valeurAbsolue (X : Entier) : Entier
    Variable va : Entier
    Début
        Si X ≥ 0 alors
            va ← X
        Sinon
            va ← -X
        FinSi
        retourner va
    Fin
Début
#valeur entrée au clavier
Ecrire ("Entrez un entier :")
Lire(a)
  b ← valeurAbsolue(a)
Ecrire ("la valeur absolue de ", a," est ", b)
Fin
```

Appel de la fonction

Fonctions en algorithmique : Exemples

Fonctions : exemple 2 (fonction minimum de deux nombres)

```
Algorithme min
variable  mini : Entier
Fonction min2 (a , b : Entier) : Entier
Début
        Si (a ≥ b) alors
            retourner b
        Sinon
            retourner a
        FinSi
Fin

Début
  mini ← min2(5,8)
  Ecrire ("Le minimum est", mini)
Fin
```


Fonctions en algorithmique : Exemples

Fonctions : exemple 2 (fonction minimum de trois nombres)

```
Algorithme min
variable  mini : Entier
Fonction min2 (a , b : Entier) : Entier
Début
        Si (a ≥ b) alors
            retourner b
        Sinon
            retourner a
        FinSi
Fin

#fonction min3 utilisant la fonction min2
Fonction min3 (a , b, c : Entier) : Entier
Début
        retourner min2(c, min2(a,b))
Fin
Début
  mini ← min3(3,5,8)
  Ecrire ("Le minimum est", mini)
Fin
```

Fonctions en algorithmique : Exemples

Fonctions : exemple 2 (fonction minimum de trois nombres)

```
Algorithme Min
variable a,b, c, mini : Entier
Fonction min2 (a , b : Entier) : Entier
Début
        Si a ≥ b alors
            retourner b
        Sinon
            retourner a
        FinSi
Fin

Fonction min3 (a , b, c : Entier) :
Entier
Début
        retourner Min2(c, Min2(a,b))
Fin
Début
    Ecrire ("Saisir trois entier")
    Lire(a,b,c)
    mini ← min3(a,b,c)
    Ecrire ("Le minimum est", mini)
Fin
```


Qu'est qu'une fonction (en python)?

⇒Qu'est qu'une fonction ?

Une **fonction en langage Python :**

- **fait un traitement**
- **a un identificateur**
- **reçoit des paramètres en cas de besoin et retourne un résultat**

La définition d'une fonction en python se fait à l'aide du mot clé **def** :

⇒Syntaxe

```
def NomFonction(parametre1, parametre2, parametre3, ...) :
    # Attention à l'indentation
    Instruction 1
    Instruction 2
    ...
    Instruction n
    return resultat
```

instruction **return**

- La **valeur de l'expression** après **return** est la valeur de retour de la **fonction (résultat)**
- Le **programme s'arrête** après l'exécution du **1er return rencontré**
- L'**instruction return** provoque:
  - ➢ la **sortie** immédiate de la fonction ;
  - ➢ le retour dans le programme appelant;
  - ➢ les instructions qui suivent, dans la fonction, ne sont pas exécutées,

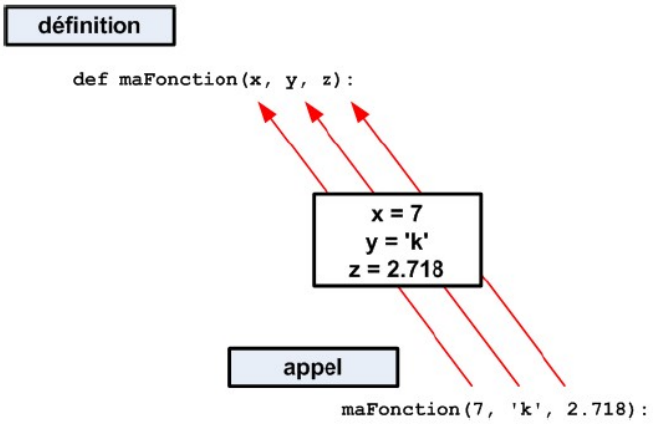
Ne pas confondre **return** et **print**

- **print** affiche un texte ou la valeur d'une variable à l'écran mais ne retourne pas de valeur,
- **return** retourne la valeur de la fonction mais n'affiche rien sur l'écran

Fonctions : Appel

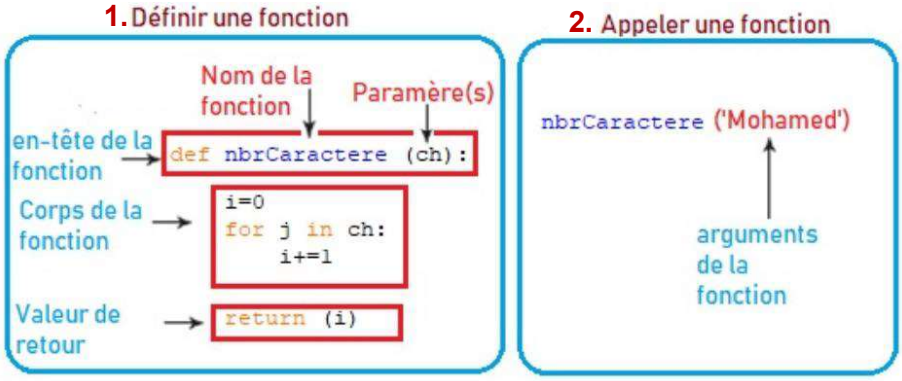
Appel d'une fonction :

- ❖ Pour appeler une fonction, il suffit d'utiliser son nom dans l'instruction **print** ou dans une **expression arithmétique.**
- ❖ Pour appeler une fonction, il suffit d'utiliser son nom dans la fonction **principale**.



Fonctions (en python)

- Les fonctions représentent une sorte de **"sous-programme dans le programme"**.
- On utilise des fonctions pour regrouper des instructions et les appeler sur demande. Cela permet de simplifier le programme principal.
- L'utilisation de fonctions se compose de deux phases :



Fonctions (en python)

Exemples

Exemple : $f(x) = x^2 + 1$

- **Le modèle mathématique :**

$$\mathbf{fct}: \mathbf{R} \rightarrow \mathbf{R}$$
$$y \rightarrow x^2 + 1$$

⇒**Code python**

```
##Définition de la fonction f()##
def fct(x) :
    y=x*x+1
    return y

## Programme principal ##
a=10
## Appel de la fonction ##
print(fct(a))
b=fct(0)+fct(2)
print(b)
print(fct(6))
```

⇒**Résultat d'exécution (écran)**

```
101
6
37
```

Fonctions (en python)

```
Algorithme Abs
   Variable  a, b : Entier
   Fonction ValeurAbsolue (X : Entier) : Entier
      Variable valeurAbsolue : Entier
      Début
            Si X ≥ 0 alors
                valeurAbsolue ← X
            Sinon
                valeurAbsolue ← -X
            FinSi
            retourner valeurAbsolue
      Fin
Début
   Ecrire ("Entrez un entier :")
   Lire(a)
      b ← valeurAbsolue(a)
   Ecrire ("la valeur absolue de ", a," est ", b)
Fin
```

Appel de la fonction

main.py | fct.py | ValeurAbsolue.py

```
1  def valeurAbsolue (X : int) :
2      if (X >= 0) :
3          return X
4      else:
5          return -X
6
7  a=int(input("Entrez un entier : "))
8  b=valeurAbsolue(a)
9  print ("la valeur absolue de ", a," est ", b)
```

⇒**Résultat d'exécution (écran)**

```
Entrez un entier :
-20
la valeur absolue de  -20  est  2
```


Fonctions (en python)

```
def min2 (a, b : int):
    if (a>=b) :
        return b
    else :
        return a
def min3 (a, b, c : int):
    return min2(c, min2(a,b))

a = int(input("Saisir a "))
b = int(input("Saisir b "))
c = int(input("Saisir c "))
mini=min3(a, b, c)
print ("La valeur minimale de a=", a," b=", b, " c=", c, "est : ", mini)
```

Fonctions : exemple 2 (fonction minimum de deux nombres)

```
Algorithme Min
variable a,b, c, mini : Entier
Fonction min2 (a , b : Entier) : Entier
Début
      Si a ≥ b alors
          retourner b
      Sinon
          retourner a
      FinSi
Fin
Fonction min3 (a , b, c : Entier) :
Entier
Début
      retourner Min2(c, Min2(a,b))
Fin
Début
   Ecrire ("Saisir trois entier")
   Lire(a,b,c)
   mini ← min3(a,b,c)
   Ecrire ("Le minimum est", mini)
Fin
```

⇒ **Résultat d'exécution (écran)**

```
Saisir a
15
Saisir b
12
Saisir c
30
La valeur minimale de a= 15  b= 12  c= 30 est :  12
```

Fonctions (en python)

Exemple 3 : $f(x) = x^2 + 1$

- **Le modèle mathématique :**

$$f: \mathbb{R} \rightarrow \mathbb{R}$$
$$x' \rightarrow x^2 + 1$$

⇒**Code python**

main.py | Exemple3.py | +

```
1  ##Définition de la fonction f()##
2  def f(x) :
3      y=x*x+1
4      return y
5
6  ## Programme principal ##
7  a=10
8  ## Appel ##
9  print(f(a))
10 b=f(0)+f(2)
11 print(b)
12 print(f(6))
```

⇒**Résultat d'exécution (écran)**

```
101
6
37
```


Procédures en algorithmique : Définition

Procédure : syntaxe

Une **procédure** est un **sous-programme** qui effectue un traitement (suite d'instructions) admettant des **paramètres** et **ne retourne aucun résultat.**

Procédure

Paramètres → **NomProécédure**

Syntaxe :

Une **procédure** se définit par la construction :

Procédure **NomProcédure** (NomArg_1:TypeArg_1,..., NomArg_k:TypeArg_k)

Déclarations des variables

Début

Corps de la procédure (Bloc d'instruction(s))

Fin

Procédures en algorithmique : Exemples

Procédure (exemple) : incrémentation)

```
Algorithme Incrémentation
    Variable x : entier
    Procédure Incrémenter(y : entier)
        Début
            Ecrire("la valeur avant:", y)
            y← y+1
            Ecrire("la valeur après:", y)
        Fin
```


Procédures (en python)

Qu'est qu'une procédure en python ?

Une **procédure** est un sous programme qui **ne retourne pas une valeur**. Généralement, une procédure est utilisée soit pour lire des objets ou soit pour les afficher.

```
def NomProcedure(param1, param2, param3, ...) :
    # attention à l'indentation
    Instruction 1
    Instruction 2
    ....
    Instruction n
```

L'appel d'une procédure

En **python**, L'appel d'une procédure se fait dans le programme principal ou dans une autre fonction ou procédure (même syntaxe que celle d'une fonction).

NomProcedure (liste paramètres)

Procédures en algorithmique : Exemples

Procédure (exemple) : incrémentation)

```
Algorithme Incrémentation
Variable x : entier
Procédure Incrémenter(y : entier)
Début
        Ecrire("la valeur avant:", y)
        y← y+1
        Ecrire("la valeur après:", y)
        Fin
Début
        X← 10
        Ecrire(x)
        Incrémenter(x)
        Ecrire("la valeur initiale :", x)
Fin
```

```
def incrémenter (y: int):
    print("la valeur avant:", y)
    y=y+1
    print("la valeur après:", y)

x=10
print(x)
incrémenter (x)
print("la valeur initiale :", x)
```

⇒**Résultat d'exécution (écran)**

```
10
la valeur avant: 10
la valeur après: 11
la valeur initiale : 10
```


Procédures (en python)

Exemples

Exemple 4 :

```
## Définition de la procédure ##
def Afficher(msg) :
        print(msg)
## Programme principal ##
ch1="Salut"
ch2="Mohamed Amine"
## Appel de la procédure ##
Afficher(ch1)
Afficher(ch2)
```

⇒**Résultat d'exécution (écran)**

Procédures (en python)

Exemple 4 :

main.py | Exemple4.py

```
1  ## Définition de la procédure ##
2  def Afficher(msg) :
3      print(msg)
4
5  ## Programme principal ##
6  ch1="Salut"
7  ch2="Mohamed Amine"
8
9  ## Appel de la procédure ##
10 Afficher(ch1)
11 Afficher(ch2)
```

⇒**Résultat d'exécution (écran)**

```
Salut
Mohamed Amine
```


Procédures (en python)

Exemple 5 :

```
##Définition de la procédure ##
def T() :
    print("Bonjour")
##programme principal##
##Appel de la procedure ##
T()
```

⇒**Résultat d'exécution (écran)**

Procédures (en python)

Exemple 5 :

main.py | Exemple4.py | Exemple5.py | +

```
1  ##Définition de la procédure ##
2  def T() :
3      print("Bonjour")
4
5  ##programme principal##
6  ##Appel de la procedure ##
7  T()
```

⇒**Résultat d'exécution (écran)**

```
Bonjour
```


Procédures (en python)

Exemple 6 :

```
##Définition des procédures ##
def Ali() :
        print("ali")
def Bonjour() :
        print("Bonjour")
        Ali()
##Programme principal##
##Appel ##
Bonjour()
```

⇒**Résultat d'exécution (écran)**

Procédures (en python)

Exemple 6 :

main.py | Exemple6.py

```
1  ##Définition des procédures ##
2  def Ali() :
3      print("ali")
4
5  def Bonjour() :
6      print("Bonjour")
7      Ali()
8
9  ##Programme principal##
10 ##Appel ##
11 Bonjour()
```

⇒**Résultat d'exécution (écran)**

```
Bonjour
ali
```


Fonctions : variables locales et variables globales

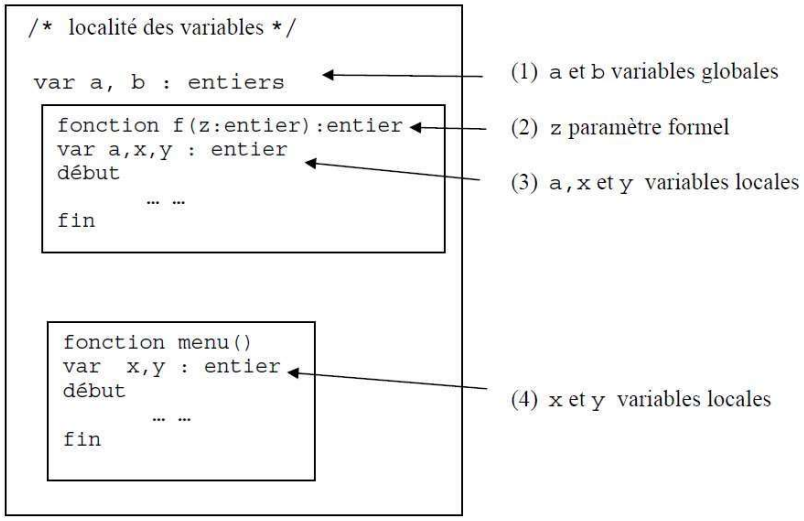
Il peut y avoir différents niveaux de déclaration de variables.

- Les **variables** définies dans le **programme** hors des fonctions sont dites **variables globales**.

➔ les **variables globales** sont connues de **tout le programme**.

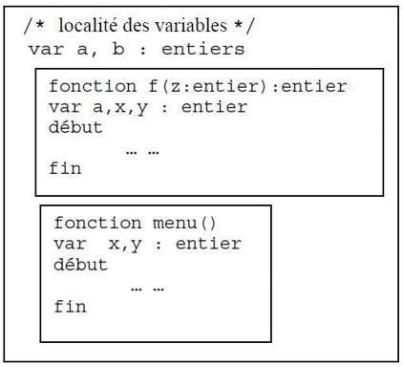
- Les **variables déclarées** dans le corps d'une fonction sont dites variables **locales** ou variables **automatiques**:
  - Connues **uniquement à l'intérieur de la fonction**,
  - Propres à la fonction et invisibles de l'extérieur.
  - Appelées aussi **automatiques** car elles sont **automatiquement détruites** après l'exécution de la fonction et leurs espaces mémoire libérés.
  - Il est possible d'utiliser des variables de même nom dans des fonctions différentes sans craintes d'interférences déplaisantes .

Fonctions : variables locales et variables globales



Fonctions : variables locales et variables globales

- Les **variables a et b**, (1) sont des **variables globales**: elles seront connues et utilisées dans toutes les parties du programme.
- La fonction **menu()** peut:
  - utiliser les variables globales a et b
  - appeler la **fonction f()**



- Une **fonction** connait:
  - ses paramètres formels,
  - ses propres variables,
  - toutes les fonctions définies avant elle
  - elle se connait elle-même **(récursivité).**
  - les variables globales dont le nom n'est pas redéfini dans la fonction,

  ➔ Conflit locale – globale (même nom) : la variable locale l'emporte

Fonctions : variables locales et variables globales

- La **fonction f()** peut utiliser les **variables locales a, x, y, z** et la **variable globale b**.
- Le **paramètre d'appel (z)** est considéré comme une **variable locale**,
- La **variable globale a** est **inaccessible** de l'intérieur de f(), toute utilisation de **a** fait référence à la **variable locale**.
- La **fonction f()** pourra s'appeler elle-même.
- La **fonction menu()** peut travailler sur les **variables locales x, y et les variables globales a et b**
- Une **fonction** peut être utilisée (appelée) par une autre fonction: la fonction **menu()** peut appeler f() *(elle est définie avant elle)*
- Une **fonction** ne peut pas être définie à l'intérieure d'une autre fonction.

```
/* localité des variables */
var a, b : entiers
   fonction f(z:entier):entier
   var a,x,y : entier
   début
         … …
   fin
   fonction menu()
   var  x,y : entier
   début
         … …
   fin
```