جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

كلية العلوم
FACULTÉ DES SCIENCES

Filière M.I.P – S2 : Tronc Commun
Année Universitaire : 2023 – 2024

Chapitre 1
Les fonctions et les procédures

Filière MIP – S2

Module : Informatique 2 : Algorithmique 2/Python

Pr. Badraddine AGHOUTANE
b.aghoutane@gmail.com

Plan

- 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

- Introdution
- Fonctions en algorithmique
- Procédures en algorithmique
- Fonctions en python: Définition & Appel
- Procédures en python : Définition & Appel
- La portée des variables (variables locales, variables globales)
- Les modules en python

Fonctions et procédures : introduction

Problème :

Comment traiter les **problèmes complexes**?

Solution :

On peut appliquer la méthode "Diviser pour régner":
Décomposer la résolution du **problème initial** en une suite de **"sous problèmes"** que l'on considère comme résolus.

Réussir à trouver et à **combiner** les solutions élémentaires des **"sous-problèmes"** pour résoudre le **problème initial**.

➔ **C'est l'analyse descendante**

Fonctions et procédures en algorithmique

Fonctions : syntaxe

Une **fonction** est un **sous-programme** admettant des **paramètres** et retournant un **résultat** (comme *les fonctions mathématiques* **y=f(x,y,..)**)

Syntaxe :

```
Fonction nomFonction (Paramètre1 : type1, ... ) : typeRetour
    variable variableLocale : type
    ...
Début
        instructions
        retourne résultat
Fin
```


Fonctions et procédure en algorithmique

➔ La **définition** se fait au moyen de :

1. **Entête de la fonction**
2. **Déclaration des variables**
3. **Corps de la fonction**

De la manière suivante:

Entête

déclarations des variables

Début

...

Corps de la fonction

...

Fin

Fonctions en algorithmique : Définition

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

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

  est composé de:
  - **nom de la fonction** (*NomFonction*), identificateur standard.
  - **liste des paramètres** (*ou arguments*) d'appel: **Optionnelle**, si la liste est vide, on peut ne rien écrire.
  - **Du type du résultat** (*TypeRetour*), **éventuellement vide**. Si la fonction ne rend pas de résultat, 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

Fonctions : exemple 1 (valeur absolue)

```
Fonction ValeurAbsolue (X : Entier) : Entier
Début
        Si X ≥ 0 alors
            retourner X
        Sinon
            retourner -X
        FinSi
Fin
```

Fonctions en algorithmique : Exemples

Fonctions : appel de 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
```


Fonctions en algorithmique : Exemples

Fonctions : appel de la fonction valeurAbsolue

```
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
```

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
```

```
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 deux nombres)

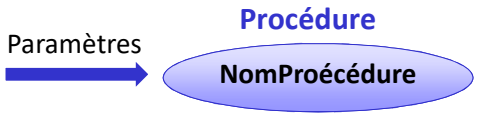
```
Algorithme min
variable a,b, c, mini : Entier
Fonction min2 (a , b : Entier) : Entier
Début
        Si a ≥ b alors
            retourner b
        Sinon
            retourner a
        FinSi
Fin
```

```
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
```

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.**



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(y)
                y←y+1
                Ecrire(y)
        Fin
Début
        x←0
        Ecrire(x)
        Incrémenter(x)
        Ecrire(x)
Fin
```

Fonctions : variables locales et variables globales

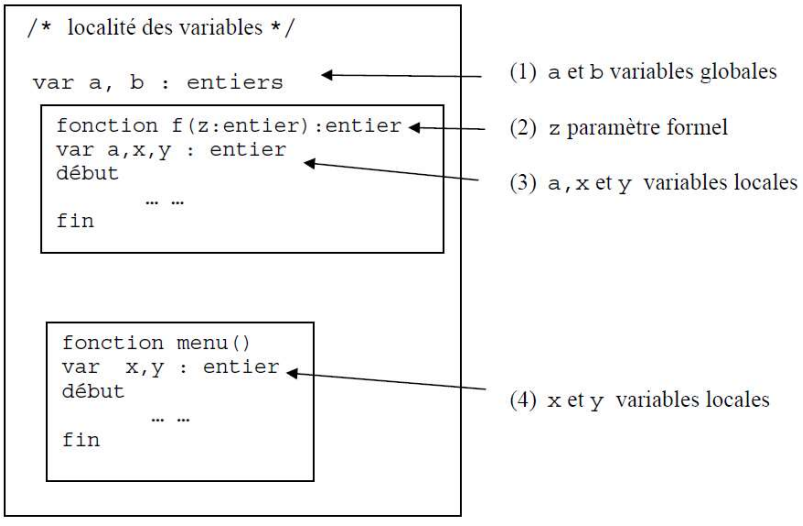
Il peut y avoir différents niveaux de déclaration de variables.

- 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.
  - Possibilité 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 utilisables dans toutes les parties du programme.
- La fonction **menu()** ne peut que:
  - travailler sur les variables globales a et b
  - appeler la **fonction f()**
- Une fonction connait:
  - les 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

17

Fonctions : variables locales et variables globales

- La **fonction f()** pourra travailler avec 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()** pourra travailler sur les **variables locales x, y et les variables globales a et b**
- la fonction **menu()** pourra appeler f() *(elle est définie avant elle)*

18

Fonctions : variables locales et variables globales

- La **fonction f()** pourra travailler avec 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()** pourra 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()** pourra appeler f() *(elle est définie avant elle)*
- Une fonction ne peut pas être définie à l'intérieure d'une autre fonction.

19

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

⇒Qu'est qu'une fonction ?

Une **fonction en langage Python :**

- fait une tâche
- a un identificateur
- reçoit des paramètres en cas de besoin et retourne un résultat si nécessaire

⇒Syntaxe

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

20

instruction **return**

- La **valeur de l'expression** suivant **return** est la valeur de la **fonction**
- 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,**

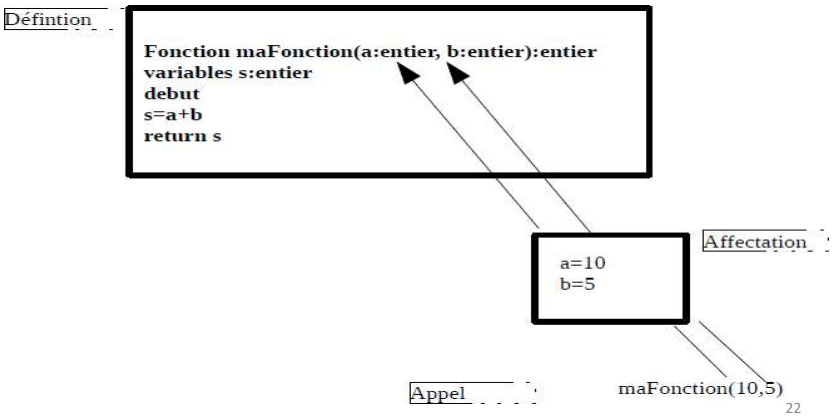
Ne pas confondre **return** et **print**

- **print** affiche un texte à l'écran mais n'a pas de valeur,
- **return** décide de la valeur de la fonction mais n'affiche rien sur l'écran

Fonctions

Appel de fonction :

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



Fonctions (en python)

Exemples

Exemple1 : $f(x) = x^2 + 1$

- **Le modèle mathématique :**

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

⇒**Code python**

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

## Programme principal ##
a=10
## Appel ##
print(f(a))
b=f(0)+f(2)
print(b)
print(f(6))
```

⇒**Résultat d'exécution (écran)**

Fonctions (en python)

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

- **Le modèle mathématique :**

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

⇒**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 python)

Qu'est qu'une procédure ?

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 function ou procédure par une instruction indiquant le nom de la procédure.

Remarque :

Dans le cas d'une **procédure**, on peut utiliser : **return ;**

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
```


Portée d'une variable

Définition :

Une **variable déclarée** dans le **corps d'une fonction** est une **variable locale** à cette fonction.

⇒ **Exemple :**

```
def tableMulti(base, debut, fin) :
    print('Affichage de la table de multiplication par', base, ' :')
    n = debut
    while n <= fin :
        print(n, 'x', base, '=', n * base)
        n=n+1
```

Dans cet exemple, les variables **base, debut, fin** et **n** représentent des **variables locales**.

Portée d'une variable locale

Portée d'une variable locale

Une **variable locale** ne peut-être lue et modifiée que dans le corps de la fonction dans laquelle elle a été déclarée.

⇒**Exemple :**

```
#####Programme  principal#####
print(base)
Traceback (most recent call last) :
File "< stdin >", line 1, in < module >
NameError : name 'base' is not defined
```

D'après cet exemple, si nous essayons d'afficher le contenu de la variable base dans le **programme principal**, nous obtenons un **message d'erreur.**

Portée d'une variable locale

Définition

Les variables définies à **l'extérieur d'une fonction** sont des **variables globales.**

⇒**Exemple :**

```
##Déclaration d'une varibale globale
N=100
def maFonction(a) :
        return N+a
##Programme principal
resultat=maFonction(10)
print(resultat)
```

N a été déclarée à l'extérieur de toute fonction, directement dans le programme principal : c'est une **variable globale**.

Portée d'une variable globale

Défintion :

Le contenu d'une variable globale est **visible** de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier.

⇒ Exemple :

```
a=10
#####Sous programme #####
def incrementer() :
        a=a+1
        return a
#####Programme  principal#####
resultat=incrementer(a)
print(resultat)
```

```
Traceback (most recent call last) :
File "C :\Users\saadi\Desktop\tp1.py", line 7, in < module >
result=incrementer()
File "C :\Users\saadi\Desktop\tp1.py", line 4, in incrementer
UnboundLocalError : local variable 'a' referenced before assignment
```


Variable locale **vs** variable globale

Variable locale vs variable globale

Dans une **fonction**, si deux variables **ont même nom**, c'est la **variable locale qui est prioritaire.**

⇒ **Exemple 7 :**

```
def mask() :
        p = 20          # variable locale
        print(p,q)
#programme principal
p= 15                   # variables globales
q = 38
mask()                  # Appel de la function mask()
print(p,q)
```

⇒Résultat d'exécution (écran)

```
20 38
15 38
```

L'instruction global

L'instruction global :

Cette instruction permet d'indiquer – à l'intérieur de la définition d'une fonction – quelles sont les variables à traiter globalement.

⇒**Exemple 8:**

```
def monter() :
        global a
        a = a+1
        print("à l'intérieur de la la fonction a=", a)
a = 15
monter()
print("à l'extérieur de la fonction a=", a)
```

⇒**Résultat d'exécution (écran)**

```
à l'intérieur de la la fonction a= 16
à l'extérieur de la fonction a= 16
```


L'instruction from et import

En **Python**, il existe de nombreux modules additionnels dont le plus connu est le module **math**, ce dernier définit une vingtaine de constantes et fonctions mathématiques usuelles.

On peut importer ce module en utilisant deux manières : l'instruction **from** ou l'instruction **import** :

L'instruction from

Cette instruction consiste à charger toutes les fonctions d'un module dans la mémoire.

⇒**Exemple**

```
from math import sin, pi
A=sin(pi/2)
Print(A)
1.0
```

Il est possible également d'importer toutes les fonctions d'un module. Pour ce faire on utilise l'astérisque (*) qui signifie « tout ».

```
from math import *
 sqrt(tan(log(pi)))
1.484345173593278
```

L'instruction from et import

L'instruction import

Cette instruction consiste à utiliser un module directement sans le charger dans la mémoire.

⇒**Exemple**

```
import math
A=math.sin(math.pi/2) 1.0
print(A)
1.0
```

⇒**Remarque**

il faut utiliser le nom du module comme préfixe, sinon on obtient une erreur.

```
import math
B=sqrt(2)
Traceback (most recent call last) :
File " < stdin > ", line 1, in < module >
NameError : name 'sqrt' is not defined
```


Comment créer un nouveau module

Un **module** est un fichier dont l'extension est **.py** et qui contient un ensemble des procédures ou des fonctions.

Pour créer un nouveau module, on applique les étapes suivantes :

1. Tout d'abord on crée le code d'une fonction nommée **fact** qui permet de calculer la factorielle d'un nombre n.
2. Ensuite on met le code de cette fonction dans un fichier appelé **progfact.py** (ce fichier est un module).
3. Enfin on peut utiliser ce nouveau module afin de calculer le nombre de combinaisons.

Comment créer un nouveau module

⇒**Exemple (solution 1)**

```
####le fichier progfact.py####
def fact(n) :
        f=1
        for i in range(1,n+1) :
                f=f*i
        return f

####le programme principal ####

from progfact import fact
f10=fact(10)
print(f10)
```


Comment créer un nouveau module

⇒**Exemple (solution 2)**

```
####le fichier progfact.py####
def fact(n) :
        f=1
        for i in range(1,n+1) :
                f=f*i
        return f

####le fichier combinaison.py ####

from progfact import fact
def comb(n,p) :
        y=fact(n)/(fact(p)*fact(n-p))
        return y
print (comb(3,1))
```

Comment créer un nouveau module

⇒**Exemple (solution 3)**

```
####le fichier progfact.py####
def fact(n) :
        f=1
        for i in range(1,n+1) :
                f=f*i
        return f
####le fichier combinaison.py ####
import progfact
def comb(n,p) :
        y=progfact.fact(n)/(progfact.fact(p)*progfact.fact(n-p))
        return y
print(comb(3,1))
```


Documentation d'une fonction

Lorsqu'on écrit une fonction, il convient de la documenter pour expliquer comment l'utiliser et éventuellement comment elle a été implémentée. C'est la doc string de la fonction à placer en tête : entre le symbole """....""".

Exemple :

```
def max_2_entiers(a,b)
"""appel : max_2_entiers(a,b)
    cette fonction retourne la valeur maximale de
    deux entiers a,b : sont deux entiers
    valeur de retour : a ou b
"""
    if a>b :
        return a
    else :
        return b
```

Documentation d'une fonction

L'accès à la documentation d'une fonction se fait via l'appel de la fonction **help(fonction).**

Exemple

```
help(max_2_entiers)
```

Cette instruction nous donne :

Help on function max_2_entiers in module max_2_ entiers(a,b)

appel : max_2_entiers(a,b)

cette fonction retourne la valeur maximale de deux entiers

a,b : sont deux entiers

valeur de retour : a ou b

⇒ Les fonctions prédéfinies sont souvent bien documentées.

Compléments sur les modules en Python

Module (numpy : gestion des tableaux).

import numpy as np	Les fonctions seront préfixées par np.
np.array(liste)	Création d'un tableau à partir d'une liste
np.linspace(min,max],nbPoints)	Création d'un tableau 1D
np.arange(min,max[,pas)	Création d'un tableau 1D
tableau.reshape(nbElts sur axe 0,...)	Permet de reformer un tableau
tableau.shape	Nombre d'éléments sur chaque axe

Module (copy : gestion des « copies »).

from copy import deepcopy	importation de la fonction deepcopy
deepcopy(var)	Création d'une copie « indépendante »

Module (os : gestion des répertoires).

import os	Les fonctions seront préfixées par os.
os.getcwd()	Quel est le répertoire de travail ?
os.chdir('chemin')	Changement du répertoire de travail
os.listdir()	Liste des fichiers dans le repertoire de travail