Algorithmique et Programmation : Fonctions et Procédures

E. M. Souidi

Faculté des Sciences - Rabat
SVI4 –STU4 2013-14

Introduction

En algorithmique il existe deux types de sous-programmes : les fonctions et les procédures.

Introduction

En algorithmique il existe deux types de sous-programmes : les fonctions et les procédures. Ces notions sont présentes dans tout les langages de programmation.

L'objectif de ces notions est de structurer son programme en un ensemble de sous-programmes résolvant chacun un sous-problème particulier.

Par exemples

Il arrive souvent dans un programme d'avoir besoin d'une tâche à appliquer plusieurs fois.

Par exemples

Il arrive souvent dans un programme d'avoir besoin d'une tâche à appliquer plusieurs fois.
Par exemples :

Par exemples

Il arrive souvent dans un programme d'avoir besoin d'une tâche à appliquer plusieurs fois.
Par exemples :
- calculer la moyenne de chaque étudiant.

Par exemples

Il arrive souvent dans un programme d'avoir besoin d'une tâche à appliquer plusieurs fois.
Par exemples :
- calculer la moyenne de chaque étudiant.
- attribuer la mention etc...

Ce serai très long d'écrire pour chaque étudiant un sous-programme qui calcule sa moyenne.

Ce serai très long d'écrire pour chaque étudiant un sous-programme qui calcule sa moyenne.
D'autant plus que c'est le même sous programme.

Ce serai très long d'écrire pour chaque étudiant un sous-programme qui calcule sa moyenne.
D'autant plus que c'est le même sous programme.
Solution :
écrire ce sous-programme une seule fois et le réutiliser autant de fois que c'est nécessaire.

Définition d'une fonction

Une fonction est un petit sous-programme qui :

Définition d'une fonction

Une fonction est un petit sous-programme qui :
- accepte éventuellement des arguments ;

Définition d'une fonction

Une fonction est un petit sous-programme qui :
- accepte éventuellement des arguments ;
- effectue une séquence d'instructions utilisant ces arguments ;

- retourne une valeur simple calculée par cette séquence d'instructions.

- retourne une valeur simple calculée par cette séquence d'instructions.
- utilisable autant de fois que nécessaire.

Deux types de fonctions

Tout langage de programmation propose un certain nombre de fonctions prédéfinies.

Deux types de fonctions

Tout langage de programmation propose un certain nombre de fonctions prédéfinies.
Exemple de fonctions prédéfinies en Python : range(), len(), type() etc..

Les fonctions définies par l'utilisateur.

Les fonctions définies par l'utilisateur.
C'est ce type va nous intéresser dans ce chapitre.

Pourquoi les fonctions ?

Les fonctions sont utilisées pour :

Pourquoi les fonctions ?

Les fonctions sont utilisées pour :
- rendre le programme mieux structuré et plus lisible ;

Pourquoi les fonctions ?

Les fonctions sont utilisées pour :
- rendre le programme mieux structuré et plus lisible ;
- éviter d'écrire plusieurs fois les mêmes portions de code ;

- être éventuellement sauvegardées dans des bibliothèques et à utiliser dans d'autres programmes.

- être éventuellement sauvegardées dans des bibliothèques et à utiliser dans d'autres programmes.
Il est toujours conseillé d'écrire son programme avec plusieurs fonctions (simples).

Fonctions en pseudo-code

fonction NomFonction(arg1, arg2, ...)
commentaires décrivant la fonction
Début
séquence d'instructions
valeur à retourner
Fin

Appel de fonctions

Les fonctions sont généralement définies au début du programme (principal).

Appel de fonctions

Les fonctions sont généralement définies au début du programme (principal). Il est possible de les appeler par leurs noms n'importe où dans le programme en spécifiant leurs arguments (et leurs types).

Les valeurs retournées par les fonctions peuvent être utilisées dans le reste des instructions ou expressions du programme.

Remarque

L'appel de fonctions n'est pas limite au programme principal, il est possible d'appeler une fonction depuis une autre fonction.

Variables dans une fonction

Plusieurs fonctions peuvent partager des variables communes qu'on qualifie alors de variables globales.

Variables dans une fonction

Plusieurs fonctions peuvent partager des variables communes qu'on qualifie alors de variables globales.
Une variable est dite locale si elle est définie au sein d'une fonction et n'est pas reconnue à l'extérieur de cette fonction. Sa portée est donc limitée à cette fonction.

Procédures

Il se peut que dans un programme, on ait à réaliser des tâches répétitives, mais que ces tâches n'aient pas pour rôle de générer une valeur particulière (effacer l'écran, afficher un logo), ou qu'elles aient pour rôle de générer plusieurs valeurs.

Une fonction ne peut ni effacer l'écran ni afficher un logo. Une fonction retourne une seule valeur.

Une fonction ne peut ni effacer l'écran ni afficher un logo. Une fonction retourne une seule valeur.
La notion de procédure (on parle aussi de sous-programme.) généralise celle de fonction.

Définition de procédure

Une procédure est une suite d'actions désignée par un nom, définie une fois dans un programme et pouvant être appelée plusieurs fois dans ce programme, ou sauvegardée dans une bibliothèque pour l'utiliser dans d'autres algorithmes.

Rôle des procédures

- éviter les répétitions inutiles de code.

Rôle des procédures

- éviter les répétitions inutiles de code.
- structurer le programme en isolant des fonctionnalités spécifiques dans des procédures séparées.

- partager le travail entre programmeurs lors de la résolution d'un problème informatique complexe.

Procédure en pseudo-code

commentaires décrivant la fonction
Pour définir une procédure
Procédure maprocédure(...)
    suite d'actions
Fin Procédure
Pour l'utiliser l'appeler par

Procédure en pseudo-code

commentaires décrivant la fonction
Pour définir une procédure
Procédure maprocédure(...)
    suite d'actions
Fin Procédure
Pour l'utiliser l'appeler par
maprocédure (...)

Remarque

La valeur retournée par une fonction est toujours affectée à une variable.

Remarque

La valeur retournée par une fonction est toujours affectée à une variable.
L'appel d'une procédure, lui, est au contraire toujours une instruction autonome. C'est un ordre qui se suffit à lui-même.

Paramètres d'une procédure

Pour une fonction on parle d'arguments et pour une procédure on parle de paramètres en entrée et de paramètres en sortie.

Paramètres d'une procédure

Pour une fonction on parle d'arguments et pour une procédure on parle de paramètres en entrée et de paramètres en sortie. Toute procédure possédant un et un seul paramètre en sortie peut également être écrite sous forme d'une fonction.

Dans la plupart des langages, on ne parlera pas de paramètres "en entrée" ou "en sortie", mais de paramètres transmis :

Dans la plupart des langages, on ne parlera pas de paramètres "en entrée" ou "en sortie", mais de paramètres transmis :
- par valeur (correspondant à un paramètre en entrée)

Dans la plupart des langages, on ne parlera pas de paramètres "en entrée" ou "en sortie", mais de paramètres transmis :
- par valeur (correspondant à un paramètre en entrée)
- par référence (correspondant à un paramètre en sortie)

Dans la plupart des langages, on ne parlera pas de paramètres "en entrée" ou "en sortie", mais de paramètres transmis :
- par valeur (correspondant à un paramètre en entrée)
- par référence (correspondant à un paramètre en sortie)

Conclusion

Une procédure est un morceau de programme qui comporte éventuellement des paramètres en entrée (transmis par valeur) et des paramètres en sortie (transmis par référence).

Portée de variables

Une variable peut être déclarée :

Portée de variables

Une variable peut être déclarée :
Comme locale (le cas le plus utilisé). Une telle variable disparaît (et sa valeur avec) dès que prend fin la procédure ou elle a été créée.

Comme globale. une telle variable est conservée intacte pour toute l'application, et conserve sa valeur et peut être traitée par différentes procédures du programme.

En pseudo-code

On peut utiliser le mot-clé global pour déclarer une variable globale :

En pseudo-code

On peut utiliser le mot-clé global pour déclarer une variable globale :
Variable globale x.

En pseudo-code

On peut utiliser le mot-clé global pour déclarer une variable globale :
Variable globale x.
Une variable globale consomme énormément de ressources en mémoire.

En pseudo-code

On peut utiliser le mot-clé global pour déclarer une variable globale :
Variable globale x.
Une variable globale consomme énormément de ressources en mémoire.
On ne déclare alors comme global qu'une variable qui doit l'être absolument .

Structure générale d'une application

Une application est couramment formée d'un programme principal, de fonctions et de procédures généralement placées au début du programme.

Structure générale d'une application

Une application est couramment formée d'un programme principal, de fonctions et de procédures généralement placées au début du programme.
L'exemple typique est celui d'un menu, ou chaque traitement est réalisé par une ou plusieurs procédures ou fonctions.

Structure d'un programme

Un programme est en général structuré de la façon suivante :

```
Nom du programme
Définition des constantes et types
Déclaration des variables globales
Fonctions et procédures
début
  instructions du programme principal
  (Appels des fonctions et procédures)
fin
```

Fonctions et Procédure en Python

Sous Python, le terme de fonction est utilisé indifféremment pour désigner à la fois des fonctions mais également des procédures. Python utilise la même instruction def pour définir les unes et les autres.

Dans certains langages de programmation, les fonctions et les procédures sont définies à l'aide d'instructions différentes.

En Python la description du rôle d'une fonction est mis entre """ ..."""". Ne jamais oublier cette description.

Fonction en Python

```
def NomFonction(arg1, arg2, ..., argN) :
""" Description de cette fonction """
    instructions
    return valeur
```

Exemple

Construisons une fonction avec Python qui détermine le minimum de deux éléments :

Exemple

```
def min(x,y) :
""" retourne le min des nombres x et y """  \par  \pause
if ( x $<$ y ) :
   return x
else :
   return y
```

Comment utiliser une fonction ?

il suffit de l'appeler par son nom en indiquant entre parenthèses ses arguments.

Comment utiliser une fonction ?

il suffit de l'appeler par son nom en indiquant entre parenthèses ses arguments.

```
>>> min(45.0,0)
```

Exemple

```
def intersect(L1, L2):
"""Retourne l'intersection de listes"""
  resultat = [ ]    # une liste vide
  for x in L1:
        if x in L2:
           resultat.append(x)
  return resultat
```

Fonction imbriquées

L'appel de fonctions n'est pas limité au programme principal, il est possible d'appeler une fonction depuis une autre fonction. Par exemple :

Exemple

```
def min3(x,y,z) :
"""retourne le min de 3 nombres """
    return min(min(x,y),z)
```

Variable globale ou locale

Les variables affectées hors de toute fonction sont globales, c'est à dire qu'elles sont reconnues aussi à l'intérieur des fonctions :

Variable globale ou locale

Les variables affectées hors de toute fonction sont globales, c'est à dire qu'elles sont reconnues aussi à l'intérieur des fonctions :

Variable globale ou locale

Les variables affectées hors de toute fonction sont globales, c'est à dire qu'elles sont reconnues aussi à l'intérieur des fonctions :

```
>>> def test(x) :
```

Variable globale ou locale

Les variables affectées hors de toute fonction sont globales, c'est à dire qu'elles sont reconnues aussi à l'intérieur des fonctions :

```
>>> def test(x) :
        return x*i
```

Variable globale ou locale

Les variables affectées hors de toute fonction sont globales, c'est à dire qu'elles sont reconnues aussi à l'intérieur des fonctions :

```
>>> def test(x) :
        return x*i
>>> i=9
```

Variable globale ou locale

Les variables affectées hors de toute fonction sont globales, c'est à dire qu'elles sont reconnues aussi à l'intérieur des fonctions :

```
>>> def test(x) :
        return x*i
>>> i=9
>>> test(8)
```

Variable globale ou locale

Les variables affectées hors de toute fonction sont globales, c'est à dire qu'elles sont reconnues aussi à l'intérieur des fonctions :

```
>>> def test(x) :
        return x*i
>>> i=9
>>> test(8)
```

Il est toutefois possible d'affecter une variable dans une fonction, tout en précisant que cette variable sera connue à l'extérieur de la fonction.

Il est toutefois possible d'affecter une variable dans une fonction, tout en précisant que cette variable sera connue à l'extérieur de la fonction.
Pour cela, il suffit, avant la première utilisation de la variable, d'annoncer qu'elle sera globale :

Il est toutefois possible d'affecter une variable dans une fonction, tout en précisant que cette variable sera connue à l'extérieur de la fonction.
Pour cela, il suffit, avant la première utilisation de la variable, d'annoncer qu'elle sera globale :

Exemple

```
def fonction(x) :
"""fonction avec variable globale i"""
global i
i=2
return x+2
print fonction(4)
print i
```

Fonctions avec un nombre variable d'arguments

En Python, il est possible de préciser qu'une fonction accepte plusieurs arguments sans en préciser nécessairement le nombre à l'avance.

Il suffit pour cela d'utiliser le caractère * suivi du nom d'un argument, qui sera dans la fonction considéré comme un tuple :

Exemple

```
def moyenne(*arguments) :
"""calcule la moyenne"""
somme=0
for nbre in arguments:
    somme=somme+nbre
return somme/float(len(arguments))
moyenne(3,5,7)
moyenne(2,10,13,17,20)
```

Fonctions récursives

On peut définir une fonction de façon récursive (elle fait appel à elle même). Factorielle est une fonction récursive :

$$Fact(n) = n * fact(n-1)$$

En Python

```
def fact(n) :
    if n<2 :
    return 1
else :
    return n*fact(n-1)
```

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

```
>>> map(fonction,liste)
```

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

```
>>> map(fonction,liste)
```

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

```
>>> map(fonction,liste)
```

et pour afficher le resultat :

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

```
>>> map(fonction,liste)
```

et pour afficher le resultat :

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

```
>>> map(fonction,liste)
```

et pour afficher le resultat :

```
>>> L = map(fonction,liste)
```

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

```
>>> map(fonction,liste)
```

et pour afficher le resultat :

```
>>> L = map(fonction,liste)
>>> list(L)
```

Application d'une fonction à une liste

Pour appliquer une fonctions à tous les éléments d'une liste on utilise la fonction prédéfinie map dont la syntaxe est :

```
>>> map(fonction,liste)
```

et pour afficher le resultat :

```
>>> L = map(fonction,liste)
>>> list(L)
```

Cette fonction doit être définie avant de faire appel à la fonction map.

```
def carre(i) :
```

```
def carre(i) :
        return i*i
```

```
def carre(i) :
        return i*i
>>> map(carre,[2,4,5])
```

```
def carre(i) :
        return i*i
>>> map(carre,[2,4,5])
```

Applications de fonctions à des listes : le filtrage

La fonction prédéfinie filter dont la syntaxe est `filter(fonction, Liste)` retourne une liste formée des éléments `x` de `Liste` pour lesquels `fonction(x)` est vrai.

exemple

```
def f(x) : return x % 2 != 0 and x % 3 != 0

filter(f, range(2,25))
```

Applications des fonctions à des listes : la réduction

Syntaxe : reduce(fonction,liste)

Applications des fonctions à des listes : la réduction

Syntaxe : reduce(fonction,liste)
La fonction prédéfinie reduce permet d'appliquer fonction aux éléments de liste, suivant le schéma suivant :

on calcule

on calcule

on calcule
val0=fonction(liste[0],liste[1]),

on calcule
val0=fonction(liste[0],liste[1]),
val1=fonction(val0,liste[2]),

on calcule
val0=fonction(liste[0],liste[1]),
val1=fonction(val0,liste[2]),
val2=fonction(val1,liste[3]), ...

on calcule
val0=fonction(liste[0],liste[1]),
val1=fonction(val0,liste[2]),
val2=fonction(val1,liste[3]), ...

Exemple

```
def minimum(i,j) :
    if i $<$ j
    return i
else :
    return j
```

Exemple

```
def maximum(i,j) :
if i > j :
  return i
else :
return j
Liste=[8, 2, 6, 3, 1, 5, 13, -8, 15, 9]
reduce(minimum,liste)
```

Exercice

En utilisant la fonction reduce, réécrivez la fonction moyenne qui calcule la moyenne d'un nombre quelconque d'éléments d'une liste.

Documentation sur une fonction

Le nom d'une fonction n'est pas suffisant pour la décrire ainsi que le type et le nombre d'arguments auxquels elle s'applique.

Documentation sur une fonction

Le nom d'une fonction n'est pas suffisant pour la décrire ainsi que le type et le nombre d'arguments auxquels elle s'applique. Pour cela en définissant une fonction on doit inclure l'aide complète entre """ """ juste après la ligne contenant def

Exemple

```
def mafonction():
"""cette fonction est bien documentée mais ne dit que bonjou
  print " bonjour "
```

Aide sur une fonction

Pour accéder à l'aide d'une fonction prédéfinie ou définie par l'utilisateur ou qu'on vient de définir, il suffit de tapper.

Aide sur une fonction

Pour accéder à l'aide d'une fonction prédéfinie ou définie par l'utilisateur ou qu'on vient de définir, il suffit de tapper.

```
print mafonction.__doc__
```

Module

Un module en Python est un ensemble de fonctions rangées dans un fichier d'extension .py concernant un domaine précis des sciences (maths, électricité, etc) que l'on peut réutiliser souvent.

La plupart des modules de Python sont installés avec les versions standards de Python. D'autres sont téléchargeables de l'internet. Les modules sont écrits par des bénivoles et sont publiés sous la licence GPL (libre).

Exemple de modules

- le module math défini les fonctions mathématiques sin, cos, ln etc.

Exemple de modules

- le module math défini les fonctions mathématiques sin, cos, ln etc.
- le module os (operating system, système d'expoitation) défini des fonctions pour créer des répertoires, les supprimer, etc

- le module turtle défini des fonctions pour dessiner des formes géométriques : droites, rectangle, ellipse etc.

Il y a plusieurs modules. Voir l'aide de Python pour la liste complète des modules : lancer Python IDLE (GUI) et dans menu ouvrir Help puis Python, Docs et Global Module Index.

La liste actualisée se trouve sur :

`http://www.python.org/doc/current/ modindex.html`

La liste actualisée se trouve sur :
`http://www.python.org/doc/current/ modindex.html`
L'utilisateur peut définir d'autres modules et les publier sur le net.

Au lancement de Python, les modules et les fonctions qu'ils définissent ne sont pas chargés automatiquement.

Comment charger un module ?

Il suffit de le charger à l'aide de la fonction import.

Comment charger un module ?

Il suffit de le charger à l'aide de la fonction import.

```
import math
```

Comment charger un module ?

Il suffit de le charger à l'aide de la fonction import.

```
import math
```

Et pour charger une fonction précise d'un module :

Comment charger un module ?

Il suffit de le charger à l'aide de la fonction import.

```
import math
```

Et pour charger une fonction précise d'un module :

```
from module import fonction
```

Comment charger un module ?

Il suffit de le charger à l'aide de la fonction import.

```
import math
```

Et pour charger une fonction précise d'un module :

```
from module import fonction
from math import cos
```

Comment charger un module ?

Et pour charger toutes les fonctions d'un module :

Comment charger un module ?

Et pour charger toutes les fonctions d'un module :

```
from module import *
```

Comment charger un module ?

Et pour charger toutes les fonctions d'un module :

```
from module import *
from math import *
```

Comment décharger un module ?

Pour vider de la mémoire un module déjà chargé, on utilise l'instruction del :

Comment décharger un module ?

Pour vider de la mémoire un module déjà chargé, on utilise l'instruction del :

```
del module
```

Comment décharger un module ?

Pour vider de la mémoire un module déjà chargé, on utilise l'instruction del :

```
del module
del math
```

Aide sur un module

Pour avoir la description d'un module : les fonctions qu'il contient, la syntaxe, et l'aide sur chacune d'entre elles, on charge le module et on utilise help.

Aide sur un module

Pour avoir la description d'un module : les fonctions qu'il contient, la syntaxe, et l'aide sur chacune d'entre elles, on charge le module et on utilise help.

```
import module
```

Aide sur un module

Pour avoir la description d'un module : les fonctions qu'il contient, la syntaxe, et l'aide sur chacune d'entre elles, on charge le module et on utilise help.

```
import module
help(module)
```

Aide sur un module

Pour avoir la description d'un module : les fonctions qu'il contient, la syntaxe, et l'aide sur chacune d'entre elles, on charge le module et on utilise help.

```
import module
help(module)
import math
```

Aide sur un module

Pour avoir la description d'un module : les fonctions qu'il contient, la syntaxe, et l'aide sur chacune d'entre elles, on charge le module et on utilise help.

```
import module
help(module)
import math
help(math)
```