Institut National Universitaire **Champollion**

Chapitre 5

Fonctions et procédures

Plan du chapitre

Fonctions et procédures

Fonctions et procédures

Exemple introductif :

On cherche à élaborer une moyenne générale d'un étudiant tout en calculant ses moyennes dans chaque UE.

On stocke pour un étudiant dans une liste de liste l'ensemble de ses notes, sachant que chaque premier élément d'une ligne contient la chaîne de caractère correspondant, puis ensuite les notes obtenues dans cette UE.

On suppose que dans chaque UE toutes les notes ont même coefficient, et que toutes les UE ont même coefficient.

Fonctions et procédures

Voilà le code correspondant :

Entrée [14]:

```
notes=[["math",12,9,15],["prog", 13, 15],["Numération",8,12,11,5,17]]
moyennesUE = []

for i in range(len(notes)):
    moyenneUE = 0.0
    for j in range(1,len(notes[i])):
        moyenneUE = moyenneUE + notes[i][j]
    moyenneUE = moyenneUE/(len(notes[i])-1)
    print("moyenne de ",notes[i][0]," : ",moyenneUE)
    moyennesUE.append(moyenneUE)

moyenneGenerale = 0.0
for i in range(len(moyennesUE)):
    moyenneGenerale = moyenneGenerale + moyennesUE[i]
moyenneGenerale = moyenneGenerale/len(moyennesUE)

print("moyenne générale : ",moyenneGenerale)
```

```
moyenne de  math  :  12.0
moyenne de  prog  :  14.0
moyenne de  Numération  :  10.6
moyenne générale :  12.200000000000001
```

Fonctions et procédures

On voit dans ce code, que le calcul de la moyenne est répété deux fois.
On peut éviter cela en définissant une fonction moyenne …

Ainsi les fonctions (et procédures) permettent de « factoriser » du code qui sera utilisé plusieurs fois dans un programme.

Cela donne aussi une lecture et une compréhension plus aisées.

Fonctions et procédures

Voici une version utilisant une fonction moyenne :

Entrée [15]:

```
notes=[["math",12,9,15],["prog", 13, 15],["Numération",8,12,11,5,17]]
moyennesUE = []

def moyenne(L,iMin):
    moy = 0.0
    for i in range(iMin,len(L)):
        moy = moy + L[i]
    return moy / (len(L)-iMin)

for i in range(len(notes)):
    moyenneUE = moyenne(notes[i],1)
    print("moyenne de ",notes[i][0]," : ",moyenneUE)
    moyennesUE.append(moyenneUE)

moyenneGenerale = moyenne(moyennesUE,0)
print("moyenne générale : ",moyenneGenerale)


```

```
moyenne de  math  :  12.0
moyenne de  prog  :  14.0
moyenne de  Numération  :  10.6
moyenne générale :  12.200000000000001
```

Fonctions et procédures

Le code du calcul de la moyenne est mis dans la fonction au lieu de le répéter deux fois dans le programme.

Repérez comment on déclare une fonction, mot clé **def**, avec ses **paramètres** : ici L et iMin, L étant une liste et iMin un entier correspondant à l'indice à partir duquel la moyenne est calculée dans la liste L.

Le mot clé **return** permet de renvoyer le résultat de la fonction.

Une fonction peut prendre en argument plusieurs paramètres mais ne renvoie qu'un seul résultat.

Fonctions et procédures

Mais : puisqu'on peut choisir le type retour celui-ci peut être une liste …

Entrée [16]:

```
1  # fonction qui retourne le minimum d'une liste et son indice
2  def minimum(L):
3      min = L[0]
4      iMin = 0
5      for i in range(1,len(L)):
6          if (L[i] < min):
7              min = L[i]
8              iMin = i
9      return [iMin, min]
10
11 L=[6,8,4,1,9,8]
12 print(minimum(L))
13
```

```
[3, 1]
```

Fonctions et procédures

Il peut y avoir plusieurs instructions return dans une fonction

Entrée [18]:

```
1  # fonction parité
2  def parite1(n):
3      if (n%2 == 0):
4          return "pair"
5      else:
6          return "impair"
7
8  print("5 est ",parite1(5))
```

```
5 est  impair
```

Cependant on évitera de programmer comme cela pour plus de lisibilité (même si cela se discute ...)

Fonctions et procédures

Autre solution sans plusieurs return :

Entrée [20]:

```
1  def parite2(n):
2      retour = "impair"
3      if (n%2 == 0):
4          retour = "pair"
5      return retour
6
7  print("5 est ",parite2(5))
```

```
5 est  impair
```

vous noterez même que cela permet de mettre des valeurs par défaut au résultat de la fonction.
Cela peut s'avérer très pratique ….

Fonctions et procédures

Et une fonction sans return ça existe ?

Et bien … oui : cela s'appelle une procédure !

C'est typiquement le rôle d'une procédure d'affichage :

Entrée [21]:

```
# affichage des moyennes des UE
def afficheMoyUE(moyennesUE):
    for i in range(len(moyennesUE)):
        print("moyenne en ",moyennesUE[i][0]," ",moyennesUE[i][1])

moyennesUE = [["math", 12], ["prog",11], ["Numeration", 9]]
afficheMoyUE(moyennesUE)
```

```
moyenne en  math   12
moyenne en  prog   11
moyenne en  Numeration   9
```

Fonctions et procédures

Notez que l'appel de la procédure se fait directement : il n'y a pas de stockage dans une variable du résultat d'une procédure puisqu'il n'y a pas de « résultat » …

Les procédures ont bien d'autres utilisations :

quand un traitement demande beaucoup d'actions ou de calculs on les découpe pour plus de clarté, en sous actions et parties de calculs, les effets de ces procédures peuvent très bien modifier uniquement des variables globales.

(en POO, on parle d'ailleurs plutôt de **méthodes** qui regroupe les termes fonctions et procédures).

Nous y reviendrons ...

Le modèle mémoire en Python

Le modèle mémoire en Python

nous avons abordé les variables en se les représentant comme des boîtes dans lesquelles on peut stocker des valeurs comme des nombres. En gros, cette représentation laisse pour le moment entendre que lors d'une définition de variable telle que a = 2, Python va réserver un espace dans la mémoire vive semblable à une boîte dans lequel le nombre 2 sera stocké. La variable a joue alors le rôle de symbole que l'on utilise à la place de la valeur 2.

Cette vision n'est pas correcte.

En effet Python voit toutes les données, y compris les entiers, réels, booléens, … comme des objets.

Le modèle mémoire en Python

Les objets sont des structures contenant à la fois des données mais aussi des comportements via des fonctions s'appliquant sur eux.

Par exemple les listes :

création d'une liste : L = [2,7,4]

et vous avez invoqués des méthodes s'appliquant aux objets de type liste. Par exemple L.append(8)

Au fait, que contient L ?

en fait L contient une adresse à partir de laquelle les données de la liste sont accessibles.

Le modèle mémoire en Python

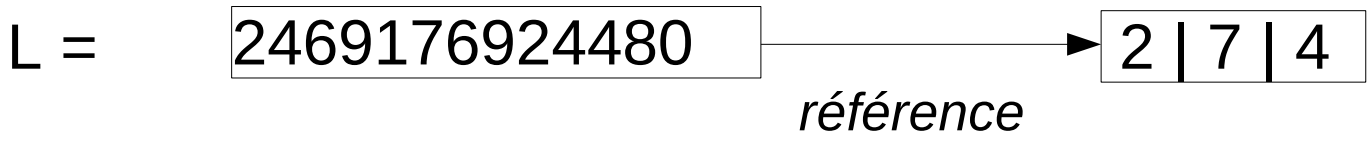
Pour obtenir une traduction de cette adresse on peut utiliser la fonction générique id(variable) :

```
Entrée [2]:
1 L =[2,7,4]
2 print("contenu de la liste L : ",L)
3 print("contenu de la variable L : ",id(L))
```

```
contenu de la liste L :  [2, 7, 4]
contenu de la variable L :  2469176924480
```

l'instruction print(L) affiche bien l'état de l'objet L, c'est à dire l'ensemble des valeurs des variables qui caractérisent L.

l'instruction id(L) renvoie un entier qui est une traduction de l'adresse qui référence L.



Le modèle mémoire en Python

Essayons de comprendre les instructions suivantes :

Entrée [9]:

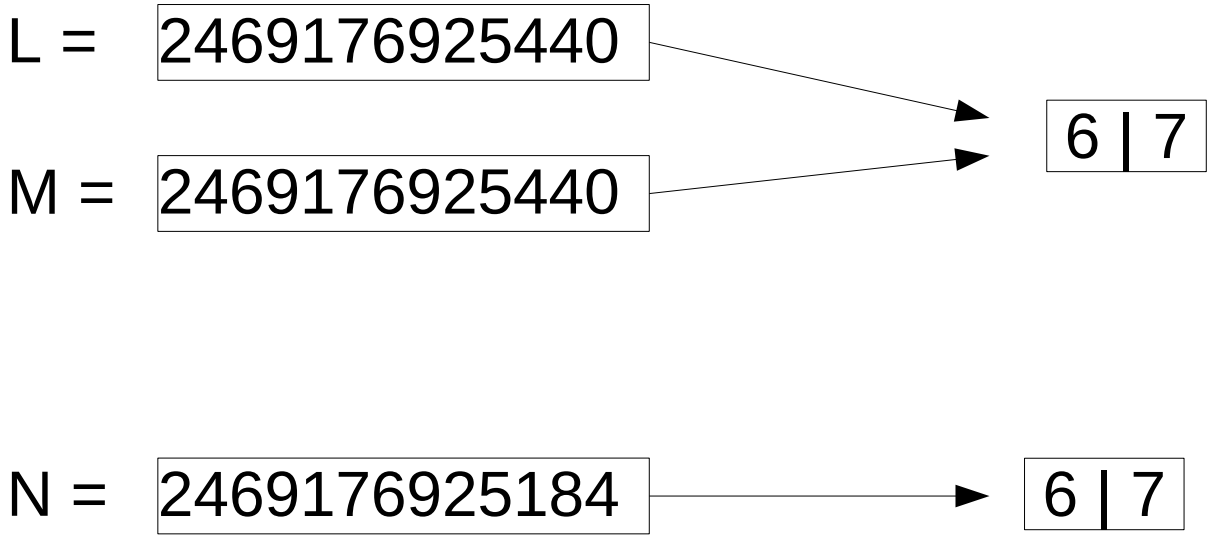
```
1 L = [6,7]
2 M = L
3 print("L == M ? ",L==M)
4 print("contenu de la variable L : ",id(L))
5 print("contenu de la variable M : ",id(M))
6 N = [6,7]
7 print("L == N ? ",L==N)
8 print("contenu de la variable N : ",id(N))
```

```
L == M ?  True
contenu de la variable L :  2469176916864
contenu de la variable M :  2469176916864
L == N ?  True
contenu de la variable N :  2469176925440
```

- on voit que l'instruction M = L met la référence contenue dans M dans L. (lignes 2-4-5)
- l'instruction N = [6,7] créé un **clone** de l'objet L (lignes 6-8)
- le test L==M ou L==N lui compare bien l'état des objets : le contenu des listes. (lignes 3 - 7)

Le modèle mémoire en Python

schématiquement :



Le modèle mémoire en Python

Regardons ce qui se passe avec les chaînes de caractères :

Entrée [10]:

```
1 s1 = "abc"
2 s2 = s1
3 print("s1 == 2 ? ",s1==s2)
4 print("contenu de la variable s1 : ",id(s1))
5 print("contenu de la variable s2 : ",id(s2))
6 s3 = "abc"
7 print("s1 == s3 ? ",s1==s3)
8 print("contenu de la variable s3 : ",id(s3))
```

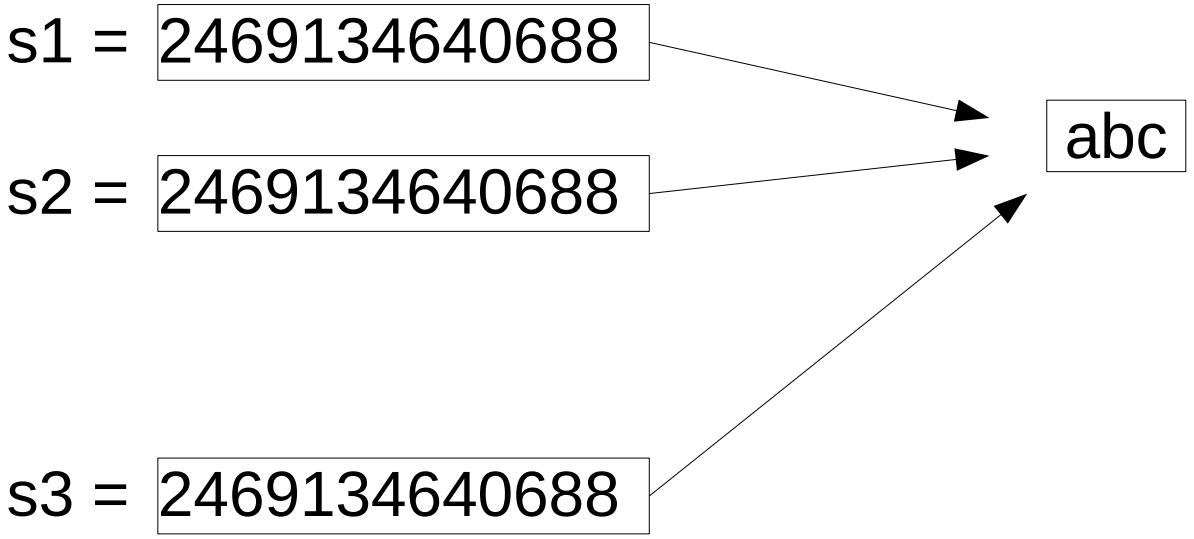
```
s1 == 2 ?  True
contenu de la variable s1 :  2469134640688
contenu de la variable s2 :  2469134640688
s1 == s3 ?  True
contenu de la variable s3 :  2469134640688
```

même comportement sur la première partie du code, **mais** :

l'instruction s3 = 'abc' n'a pas créé de nouvel objet

Le modèle mémoire en Python

schématiquement :



Les objets de type String sont **immuables** : deux objets différents ne peuvent avoir le même état.
Autrement dit deux chaînes de caractères différentes ne peuvent pas contenir les mêmes caractères.

Le modèle mémoire en Python

Et que se passe t-il pour les entiers ?

Entrée [14]:

```
1 a = 2
2 b = a
3 print("a == b ? ",a==b)
4 print("contenu de la variable a : ",id(a))
5 print("contenu de la variable b : ",id(b))
6 c = 2
7 print("a == c ? ",a==c)
8 print("contenu de la variable c : ",id(c))
```

```
a == b ?  True
contenu de la variable a :  140724492711744
contenu de la variable b :  140724492711744
a == c ?  True
contenu de la variable c :  140724492711744
```

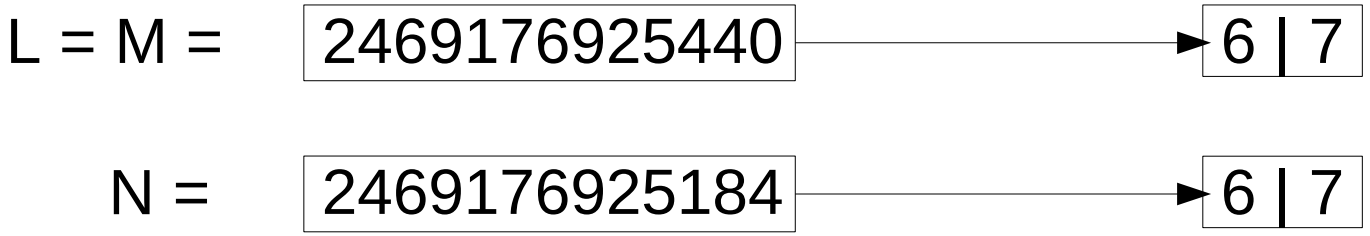
Même comportement que pour les objets String !!!

Les objets de type int, float, boolean sont immuables

Le modèle mémoire en Python

Du coup nos schémas sont à corriger !!

pour les listes (diapo 19) :



pour les String (diapo 21) :

Les paramètres

Les paramètres

Portée des variables :

Quand on définit une variable dans une fonction celle-ci n'est connue que dans la fonction ou elle est déclarée : on parle de **variable locale :**

```
  File "D:\cleF\francois_Cle\ALBI\2021\S1_S3_S5\L1_python
\cours5_TP5_Procédures_PortéePassageVariables\sanstitre1.py", line
11, in <module>
    print(y)

NameError: name 'y' is not defined
```

```
1
2
3   def f(x) :
4       y = 3
5       return x+y
6
7   # programme principal
8
9   a = 2
10  print(f(a))
11  print(y)
12
```

Les paramètres

Mais :

```
1   #
2
3   def f(x) :
4       print("a = ",a)
5       return x+3
6
7   # programme principal
8
9   a = 2
10  print("f(a) = ",f(a))
11
```

```
In [13]: runfile('D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables/sanstitre1.py',
wdir='D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables')
a =  2
f(a) =  5
```

dans ce cas-ci la référence a est connue via le paramètre x

Les paramètres

Passage des paramètres :

```
1   #
2
3   def fonction1(unEntier, uneListe):
4       retour = []
5       for i in range(len(uneListe)):
6           retour.append (uneListe[i] + unEntier)
7       unEntier = unEntier + 1
8       return retour
9
10  # Programme principal
11
12  a = 2
13  L1 = [3,5]
14  L2 = fonction1(a,L1)
15  print("après fonction1")
16  print("L1 = ",L1)
17  print("L2 = ",L2)
18  print("a = ",a)
```

```
In [19]: runfile('D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables/im13.py', wdir='D:/cleF/
francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables')
après fonction1
L1 =  [3, 5]
L2 =  [5, 7]
a =  2
```

version élémentaire : on créé une var locale que l'on retourne.

Les paramètres

Or, les listes sont mutables :

```
1   #
2
3   def fonction2(unEntier,uneListe):
4       for i in range(len(uneListe)):
5           uneListe[i] = uneListe[i] + unEntier
6
7   # Programme principal
8
9   a = 2
10  L1 = [3,5]
11  fonction2(a,L1)
12  print(L1)
```

```
In [18]: runfile('D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables/sanstitre2.py',
wdir='D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables')
[5, 7]
```

on peut donc faire non pas une fonction mais une procédure

on ne pourrai pas faire cela avec les chaînes ...

Les paramètres

on ne pourrai pas faire cela avec les chaînes :

```
1   #
2
3   def fonction2(uneC,uneChaine):
4       uneChaine = uneChaine + uneC
5
6   # Programme principal
7
8   uneC = "d"
9   chaine = "abc"
10  fonction2(uneC,chaine)
11  print(chaine)
```

```
In [22]: runfile('D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables/sanstitre3.py',
wdir='D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables')
abc
```

car elles sont imutables ...

Les variables globales

Les variables globales

Ou comment partager quoiqu'il arrive des variables.

On a vu (diapo 25) que les variables ont une portée, et que l'on peut modifier des variables mutables (mais pas les variables immuables) d'un pgm principal via une procédure en les passant en paramètre (diapos 28,29).

Or dans un programme on a besoin de partager des variables entre différentes fonctions , procédures ou pgm principal, et on aimerai pouvoir les modifier à loisir.

C'est ce que l'on va faire avec des **variables globales**.

Les variables globales

Cette fois-ci (voir diapo 25) la variable y est partagée :

```
1   #
2
3   def f(x):
4       global y
5       y = 3
6       return x+y
7
8   # programme principal
9
10  global y
11  a = 2
12  print("f(2) = ",f(a))
13  print("y = ",y)
14
```

```
In [26]: runfile('D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/
L1_python/cours5_TP5_Procédures_PortéePassageVariables/
sanstitre4.py', wdir='D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/
L1_python/cours5_TP5_Procédures_PortéePassageVariables')
f(2) =  5
y =  3
```

Les variables globales

On peut toujours partager des types mutables :

```
1   #
2
3   def fonction2(unEntier):
4       global L1
5       for i in range(len(L1)):
6           L1[i] = L1[i] + unEntier
7
8   # Programme principal
9
10  a = 2
11  global L1
12  L1 = [3,5]
13  fonction2(a)
14  print("L1 = ",L1)
```

```
In [29]: runfile('D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/
L1_python/cours5_TP5_Procédures_PortéePassageVariables/im17.py',
wdir='D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables')
L1 =  [5, 7]
```

plus besoin de passer la liste en paramètre !!

Les variables globales

Et on peut partager des types immuables :

```
1   #
2   def fonction2(uneC):
3       global chaine
4       chaine = chaine + uneC
5
6   # Programme principal
7
8   global chaine
9   uneC = "d"
10  chaine = "abc"
11  fonction2(uneC)
12  print(chaine)
```

```
In [34]: runfile('D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/
L1_python/cours5_TP5_Procédures_PortéePassageVariables/im18.py',
wdir='D:/cleF/francois_Cle/ALBI/2021/S1_S3_S5/L1_python/
cours5_TP5_Procédures_PortéePassageVariables')
abcd
```

Exercices

Exercice 1 : implémenter une fonction max(a,b) qui retourne la valeur maximum ente a et b.

Exercice 2 : implémenter une fonction abs(x) qui renvoie la valeurs absolue de x.

Exercice 3 : implémenter une fonction max(L), qui renvoie l'élément maximum de la liste L.

Exercice 4 : refaites l'exercice 3 en utilisant la fonction max de l'exercice 1.

Exercice 5 : implémentez une fonction addition(L1,L2) qui renvoie une nouvelle liste dont les éléments i, sont la somme de L1[i] et L2[i]