Cours Info - 8

Fonctions & Modularité

D.Malka

MPSI 2015-2016

Sommaire

Sommaire

Intérêt des fonctions

Calcul de factorielle n

On veut calculer factorielle 10! dans un programme :

```
1 n=10;fact=1;c=1
2 while c<=n:
3     fact=fact*c
4     c+=1
5
6 n=100;fact=1;c=1
```

Intérêt des fonctions

Calcul de factorielle n

Dans le même programme, on veut aussi calculer 100! :

```
1  n=10;fact=1;c=1
2  while c<=n:
3      fact=fact*c
4      c+=1
5
6  n=100;fact=1;c=1
7  while c<=n:
8      fact=fact*c
9      c+=1
```

Intérêt des fonctions

Calcul de factorielle n

Dans l'exemple précédent :

- **On a copié-collé du code**
- **Lourd**
- **Inélégant**
- **Source de propagation d'erreurs**

Idée : factoriser du code ! $\Rightarrow$ Notion de fonction

Sommaire

Déclaration d'une fonction

En Python

En Python :

```
1 def factorielle(n):#declaration
2     fact=1
3     c=1
4     while c<=n:
5       fact=fact*c
6       c+=1
7     return fact
8
9 fact_100=factorielle(100)#appel
```

```
1 def factorielle(n):#declaration
2     fact=1
3     c=1
4     while c<=n:
5       fact=fact*c
6       c+=1
7     return fact
```

Déclaration d'une fonction

En Python

En Python :

- **déclaration : mot clef `def`**
- **indentation signifiante $\Rightarrow$ corps de la fonction**
- **n : argument formel**
- **return : instruction de retour de la valeur de la fonction**
- **fact : valeur renvoyée par la fonction = valeur de la fonction**

Déclaration d'une fonction

Généralisation

En pseudo-code :

Fonction *nom_fonction(arguments_formels) : type de la valeur de retour*
    corps de la fonction
    retourner *valeur de retour*

Algorithm 1: Déclaration générique d'une fonction

Déclaration d'une fonction

Qu'est-ce qu'une fonction ?

Qu'est-ce qu'une fonction ?

Une fonction est une suite d'instructions qui dépend de paramètres (= arguments de la fonction).

Déclaration d'une fonction

Arguments formels

Arguments formels

Les arguments formels d'une fonction n'ont pas de valeur à la déclaration. Ils ne prendront une valeur que lors de l'appel de la fonction.

Il existe des fonctions ne prenant pas d'arguments. En pseudo-code :

fonction *politesse() : chaine de caractères*
  retourner *« Salut les amis ! ça va ? »*

Algorithm 2: Fonction politesse

Déclaration d'une fonction

Instruction `return`

Instruction `Renvoyer`, soit `return` en Python :

Valeur de retour : instruction `return`

- **La valeur de l'expression suivant `return` est la valeur de la fonction**
- **Le programme s'arrête après exécution du 1er `return` rencontré**
- **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.**
- **L'instruction `return` est facultative. En son absence la valeur de la fonction est `None`**

Déclaration d'une fonction

Déclaration en algorithmique

En pseudo-code :

```
/*Déclaration de la fonction*/
Fonction factorielle(n : entier) : entier
    fact=1
    c=1
    tant que c<=n faire
        fact=fact*c
        c=c+1
    retourner fact
```

Algorithm 3: Fonction factorielle

Déclaration d'une fonction

Intérêt des fonctions

Pourquoi déclarer des fonctions ?

- **factorisation, réutilisabilité du code**
- **lisibilité & maintenance du code**
- **boite noire : pas besoin de connaître l'implémentation pour l'utilisateur**
- **modularité du code**
- **structuration du programme $\Rightarrow$ facilité de conception**

Sommaire

Appel d'une fonction

Algorithmique

En pseudo-code :

```
/*Déclaration de la fonction*/
Fonction factorielle(n : entier) : entier
    fact=1
    c=1
    tant que c<=n faire
        fact=fact*c
        c=c+1
    retourner fact
/*Appel de la fonction*/
factorielle(10)
a=15
x=factorielle(a)
```

Algorithm 4: Fonction factorielle

Appel d'une fonction

En Python

Appel d'une fonction en Python :

```
1 def factorielle(n):#declaration
2     fact=1
3     c=1
4     while c<=n:
5       fact=fact*c
6       c+=1
7     return fact
8
9 fact_100=factorielle(100)#appel
```

Appel d'une fonction

Arguments effectifs

Arguments effectifs

Les arguments passés lors de l'appel de la fonction sont appelés *arguments effectifs* ou *arguments fonctionnels*.

Appel d'une fonction

Que se passe-t-il en mémoire ?

Que se passe-t-il en mémoire au cours de l'appel d'une fonction ?

Passage d'arguments par valeur

Au moment de l'appel de la fonction :

- **des variables correspondant aux arguments formels sont créés en mémoire,**
- **les valeurs des arguments effectifs sont copiés dans les arguments formels,**
- **à la fin de l'appel de la fonction, les arguments formels sont détruits.**
- **à la fin de l'appel de la fonction, les arguments fonctionnels n'ont pas été modifiés.**
- **On appelle ce mode de passage : passage d'argument par valeur. Seul la valeur de l'argument est transmise à la fonction, pas l'argument lui-même**

Appel d'une fonction

Que se passe-t-il en mémoire ? Cas d'un liste

Cas d'une liste : valeur passé en argument = adresse de la liste.

L'argument formel pointe vers la même liste que l'argument fonctionnel.

A la fin de l'appel, la liste passée en argument a pu être modifiée.

Appel d'une fonction

Que se passe-t-il en mémoire ? Cas d'un liste

Exemple

Tester le code suivant :

```
1  l=[1,2,3,4,5]
2
3  def nullify_list(l):
4      n=len(l)
5      for i in range(n):
6          l[i]=0
7      return None
8
9  nullify_list(l)
10 print(l)
```

Sommaire

Variables locales et globales

Variable locale

Dans la fonction factorielle définie ainsi :

```
1 def factorielle(n):#declaration
2     fact=1
3     c=1
4     while c<=n:
5       fact=fact*c
6       c+=1
7     return fact
```

on déclare et initialise deux variables `fact` et `c` : ce sont deux *variables locales*.

Variables locales et globales

Portée d'une variable locale

Variable locale

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

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.

Variables locales et globales

Gestion mémoire des variables locales

Gestion mémoire des variables locales

Tout comme les arguments formels, les variables locales sont créées au moment de l'appel de la fonction, utilisées/modifiées pendant l'exécution de la fonction et détruite à la fin de l'appel.

Exemple

Tester sur Python Tutor l'exécution de `factorielle(5)` avec la fonction `factorielle` précédemment définie.

Variables locales et globales

Variable globale

Dans le programme suivant :

```
1 N=100
2
3 def ma_fonction(a):
4     return N+a
5
6 resultat=ma_fonction(10)
7
8 print(resultat)
```

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

Variables locales et globales

Portée d'une variable globale

Portée d'une variable globale

Une variable globale peut-être lue et modifiée depuis n'importe quel endroit du programme.

Variables locales et globales

Variable locale vs variable globale

Que se passe-t-il si une variable locale et une variable globale ont même nom ?

```
1  N=100
2
3  def ma_fonction(a):
4      N=10
5      return N+a
6
7  resultat=ma_fonction(10)
8
9  print(resultat)
10 print(N)
```

Variables locales et globales

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.

De la bonne utilisation des variables globales

- Les variables globales doivent être très peu utilisées car modifiables n'importe où dans le programme.
- On les utilisera uniquement pour définir des paramètres constants dans tout le programme.
- Dans toute autre situation, on préfère utiliser un passage d'argument par valeur.

Application 1

Ecrire une fonction qui prend en arguments deux entiers n et m et qui :

- **renvoie -1 si** $n < m$
- **renvoie 0 si** $n = m$
- **renvoie 1 si** $n > m$

Fonction comparaison

```
/*Appel de la fonction*/
fonction comparaison(n : entier, m :entier) : entier
    si n<m alors
        retourner -1
    si n==m alors
        retourner 0
    si n>m alors
        retourner 1
```

Algorithm 5: Fonction comparaison

Application 2

Ecrire une fonction renvoyant la norme d'un vecteur du plan de coordonnées (x, y).

```
1 import math
2
3 def norme_vect(x,y):
4     return math.sqrt(x**2+y**2)
```

Sommaire

Fonction passée en argument

En Python, une fonction est une valeur comme une autre, on peur la passer en argument. Voici un exemple :

```
1  def somme_fonction(f,N):
2      somme=0
3      for i in range(N+1):
4          somme=somme+f(i)
5      return somme
6
7  def carre(x):
8      return x**2
9
10 somme_carre=somme_fonction(carre,3)
```

Fonction passée en argument

Autre exemple avec une fonction anonyme de syntaxe :

```
lambda x:x**2+x+1
```

très pratique pour les fonctions se réduisant à une expression.

```
1  def somme_fonction(f,N):
2      somme=0
3      for i in range(N+1):
4          somme=somme+f(i)
5      return somme
6
7  somme_carre=somme_fonction(lambda x:x**2+x+1,3)
8
9  print(somme_carre)
```

Sommaire

Fonctions prédéfinies

Il existe de nombreuses fonctions pré-écrites contenues dans des modules et bibliothèque natifs de Python ou bien à ajouter : `math`, `random`, `numpy`, `scipy`...

Exemple

```
1 import math
2 import numpy
3
4 print(numpy.cos(3))
5
6 print(math.exp(3))
```

Sommaire

Documentation d'une fonction

Documenter une fonction en Python

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

```
1  def retourne_liste(liste):
2      """
3      appel : retourne_liste(liste)
4      inverse l'ordre des elements de la liste en argument
5      liste : liste d'elements quelconques
6      Valeur de retour : None
7      """
8      for i in range(len(liste)/2):
9          (liste[i],liste[-i-1])=(liste[-i-1],liste[i])
```

Documentation d'une fonction

Documenter une fonction en Python

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

Exemple

```
1 help(retourne_liste)
```

Donne :
Help on function retourne_liste in module __main__ :

retourne_liste(liste)
appel : retourne_liste(liste)
inverse le classement des elements de la liste en argument
liste : liste d'elements quelconques
valeur de retour : None

Les fonctions prédéfinies sont souvent bien documentées.