Fonctions / procédures

Programmation en Python–MPSI–

MPSI

mazza8azzouz@gmail.com

16 décembre 2024

Plan

Introduction

Exercice

Écrire un algorithme qui demande à l'utilisateur de lire deux entiers p et n et qui calcule et affiche le nombre de combinaisons de p parmi n. Le calcul se fait de la manière suivante :

$$\binom{n}{p} = \frac{n!}{p! * (n-p)!}$$

⇒**Trace d'exécution :**

```
Donnez un entier n :3
Donnez un entier p :1
Le nombre est 3
```

Introduction

⇒ **Solution de l'exercice**

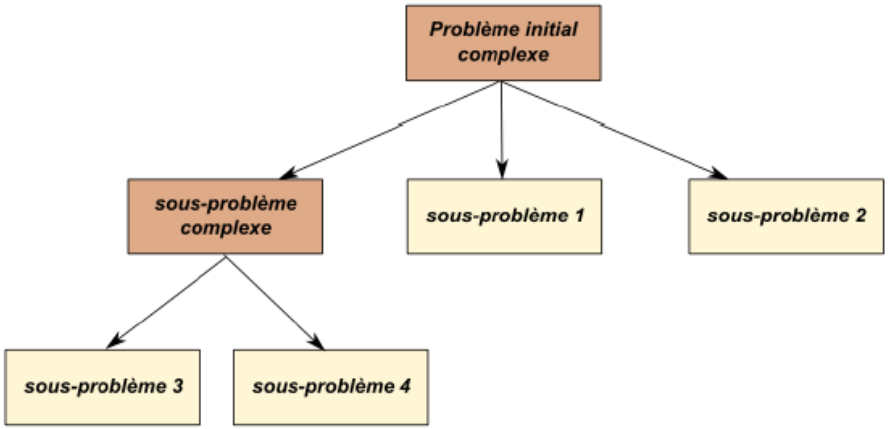
```
Algorithme nbr_combinaisons ;
variables n,p,i,FN,FP,FNP,CPN :entiers
début
Ecrire("Donnez un entier n :")
lire(n)
Ecrire("Donnez un entier p :")
lire(p)
##Calcul de n !##
FN←1
Pour i←1 jusqu'à n faire
    FN←FN*i
FinPour
##Calcul de p !##
FP←1
Pour i←1 jusqu'à p faire
    FP←FP*i
FinPour
##Calcul de (n-p) !##
FNP←1
Pour i←1 jusqu'à n-p faire
    FNP←FNP*i
FinPour
##Calcul de CPN##
CPN←(FN*FP)/FNP
Ecrire("le nombre est :",CPN)
Fin
```

Introduction

- Dans cet algorithme, nous avons calculé la factorielle de 3 nombres :**n,p et n-p** chacun à part. Pour chaque calcul, nous avons utilisé un certain nombre de lignes (bloc d'instructions) qui ce sont répétées 3 fois en modifiant uniquement la valeur. Cela implique les inconvénients suivants :
  - Le code devient lourd
  - Inélégant
  - Source de propagation d'erreurs
- Par conséquent, cet algorithme n'est pas bien structuré (mauvaise programmation).
- Pour résoudre ce problème on introduit **le concept de la programmation modulaire**

Le concept de la programmation modulaire

La programmation modulaire consiste à décomposer un problème donné en ensemble fini des sous problèmes. Ensuite, pour chaque sous-problème i, on cherche de trouver la solution i. Enfin on regroupe l'ensemble des sous solultions pour aboutir à la résolution du problème de départ. Chaque solution d'un sous problème est appellée un module. Un module est une entité qui contient les données et les instructions. Cette entité est appellée aussi un sous programme.



Qu'est-ce qu'un sous programme ?

Définition

- Un sous programme est une partie du programme principal, qui porte un nom donné et qui peut être appelé selon les besoins pour exécuter une tâche donnée.
- En programmation, on distingue deux types d'un sous programme :
  1. Les fonctions
  2. Les procédures

⇒**Objectif**

- Il existe déjà des fonctions prédéfinies comme : len, sqrt, pow, abs, randint,etc.
- Notre objectif consiste à developper nos propres fonctions.

Syntaxe algorithmique

⇒Définition :

En programmation, la définition d'une fonction est similaire à la définition basique d'une fonction mathématique ($f(x) = x^2 + 1,\ g(x) = \sqrt{x}, ...$). De plus, une fonction est un sous programme qui **retourne une valeur**
Retourner une valeur signifie que lorsqu'on calcule l'image d'un point donné en utilisant une fonction donnée, alors on obtient un résultat (valeur unique).

Syntaxe

La définition d'une fonction est composée :

- du mot clé `def` suivi de l'identificateur de la fonction, de parenthèses entourant les paramètres de la fonction séparés par des virgules, et du caractère ≪ deux points ≫ qui termine toujours une instruction composée ;
- d'une chaîne de documentation indentée comme le corps de la fonction ;
- du bloc d'instructions indenté par rapport à la ligne de définition, et qui constitue le corps de la fonction.

Le bloc d'instructions est obligatoire. S'il est vide, on emploie l'instruction `pass`. La documentation (facultative) est fortement conseillée.

Exemples : :exemple 1

```
def afficheAddMul(a, b) :
"""Calcule et affiche la somme et le produit de a et b."""
    somme = a + b
    produit = a * b
    print("La somme de", a, " et", b, " est", somme, " et
```

Exemples : :exemple 2

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

- Le modèle mathématique :

$$\begin{array}{rccc} f & : & \mathbb{R} & \rightarrow & \mathbb{R} \\ & & x & \mapsto & x^2+1 \end{array}$$

- Analyse du problème :

- **Prorgamme :**

```
def f(x):
    return x**2 + 1
```

Exemples : :exemple 3

Exemple3 : $fact(n) = n!$

- Le modèle mathématique :

$$\begin{array}{rcccc} fact & : & \mathbb{N} & \rightarrow & \mathbb{N} \\ & & n & \mapsto & n! \end{array}$$

- Analyse du problème :

- **Prorgamme :**

```
???
```

Exemples : :exemple 4

Exemple4 : Calcul de 2^n

- Le modèle mathématique :

$$\begin{array}{rcccc} DeuxPuiss & : & \mathbb{N} & \rightarrow & \mathbb{N} \\ & & n & \mapsto & 2^n \end{array}$$

- **Prorgamme :**

```
???
```

Exemples : :exemple 5

Exemple 5 : calcul de $g(x)$

- Le modèle mathématique :

$$g(x) = \begin{cases} \sqrt{x} + 2 & \text{si } x \geqslant 0 \\ (-x)^5 & \text{sinon} \end{cases}$$

- **Prorgamme :**

```
???
```

Comment appeler une fonction ?

Pour appeler une fonction, il suffit d'utiliser son nom.

Exemple

```
def cube(a):
'''une fonction qui calcule le cube d'un entier en paramèt
    return a**3
##programme principal##
n=int(input("Saisir un entier:"))
## Appel de la fonction
print("le cube de ",n,"est :",cube(n))
```

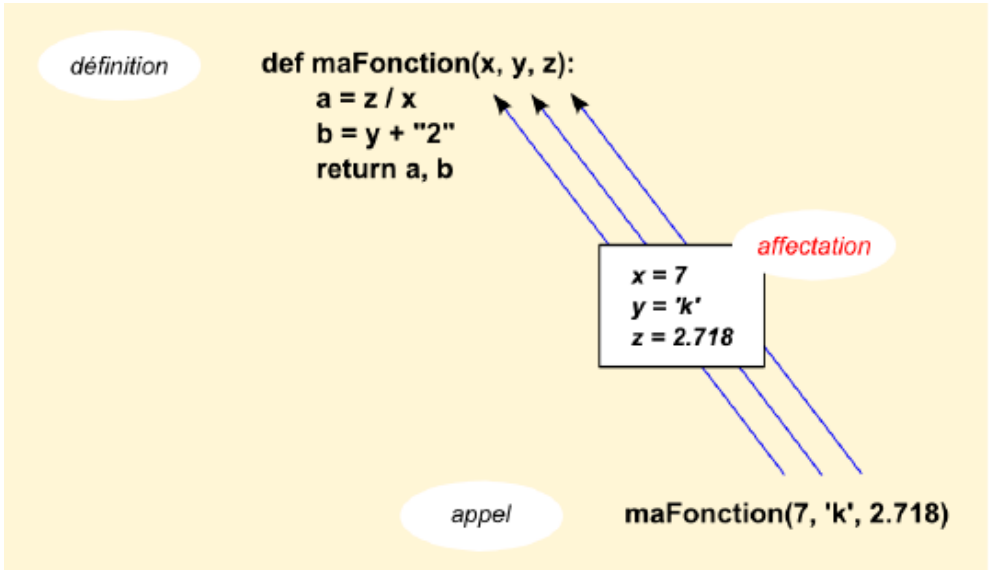
⇒**Trace d'exécution :**

```
Saisir un entier n :3
le cube de 3 est : 27
```

Comment appeler une fonction ?

Remarque

L'exécution d'un programme se fait à partir du programme principal, ensuite lorsque l'interpréteur python rencontre une fonction (où une procédure) dans le programme principal, il exécute le code de cette fonction, la correspondance se fait par affectation des paramètres aux arguments, puis il retourne la valeur dans le programme principal. A la fin de l'exécution de cette fonction, l'interpréteur continu l'exécution des instructions qui viennent après le code de cette fonction.



Exercices

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

- Le modèle mathématique :

$$\begin{array}{rccc} f & : & \mathbb{R} & \rightarrow & \mathbb{R} \\ & & x & \mapsto & x^2+1 \end{array}$$

⇒Code python

⇒Résultat d'exécution (écran)

Exercices

Exemple2 : $fact(n) = n!$

- Le modèle mathématique :

$$\begin{array}{rcccc} fact & : & \mathbb{N} & \rightarrow & \mathbb{N} \\ & & n & \mapsto & n! \end{array}$$

⇒Code python

⇒Résultat d'exécution (écran)

Exercices

Exemple3 : $g(x) = \sqrt{x} + 2$

- Le modèle mathématique :

$$g(x) = \sqrt{x} + 2$$

⇒Code python

⇒Résultat d'exécution (écran)

Exercices

Exemple 4 : calcul de $h(x)$

- Le modèle mathématique :

$$h(x) = \frac{\sqrt{x} + 2}{x^2 + 1}$$

⇒Code python

⇒Résultat d'exécution (écran)

Syntaxe algorithmique

⇒**Définition :**

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.

Syntaxe

Même syntaxe qu'**une fonction**, sauf qu'une procédure ne contient pas l'instruction `return`

Comment appeler une procédure ?

L'appel d'une procédure se fait dans le programme principal ou dans une autre procédure par une instruction indiquant le nom de la procédure.

Exemples

Exemple1 :

```
##sous programme##
def Afficher(msg):
    print(msg)
##programme principal##
ch1="salut"
ch2="sami"
##Appel ##
Afficher(ch1)
Afficher(ch2)
```

⇒Résultat d'exécution (écran)

Exemples

Exemple 2 :

```
##sous programme##
def T() :
    print("Bonjour")
##programme principal##
##Appel ##
T()
```

⇒Résultat d'exécution (écran)

Portée d'une variable locale

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",
    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**

```
>>> 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 juste après avoir effectué l'exécution du code, nous obtenons un message d'erreur.

Portée d'une variable globale

Définition

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

⇒**Exemple**

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

N a été déclarée à l'extérieur de toute fonction, directement dans

Portée d'une variable globale

Portée d'une variable globale

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) :
```

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**

```
def mask() :
    p = 20
    print(p,q)
#programme principal
p, q = 15, 38 #affectation parallèle
mask() #Appel
20 38
print(p, q)
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**

```
def monter() :
    global a
    a = a+1
    print(a)
##Programme principal
a = 15
monter()
16
```

Exemples

⇒Exemple1

```
# x et fonc sont affectés dans le module : globaux
def fonc(y) :
# y et z sont affectés dans fonc : locaux
    global x
    x += 2
    z = x + y
    return z
#Programme principale
x = 99
print(fonc(1))
print(x)
```

Exemples

⇒Exemple2

```
def fonc(y) :
# y et z sont affectés dans fonc : locaux
    z = x + y
    return z
#Programme principale
x = 99
print(fonc(1))
print(x)
```

Exemples

⇒Exemple3

```
def fonc(y) :
    x = 3
    z = x + y
    return z
#Programme principal
x = 99
print(fonc(1))
print(x)
```

L'instruction from et import

En Python, il existe de nombreux modules additionnels dont le plus connu est le module math qui définit une vingtaine de constantes et fonctions mathématiques usuelles. On peut importer ce module en utilisant deux manières l'instruction from et 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
>>> sin(pi/2)
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
>>> math.sin(math.pi/2)
1.0
```

⇒**Remarque**

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

```
>>> import math
>>> 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. d'abord on fait 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 fichier combinaison.py ####
from progfact import fact
def comb(n,p) :
        y=fact(n)/(fact(p)*fact(n-p))
        return y
>>> comb(3,1)
3
```

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

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)
```

Donne :
Help on function max_2_entiers in module __main__ :
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 (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 répertoire de travail

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 ≫