JEAN-NOËL BEURY

INFORMATIQUE AVEC PYTHON

PRÉPAS SCIENTIFIQUES

EXERCICES INCONTOURNABLES

j'intègre

DUNOD

Conception graphique de la couverture : Hokus Pokus Créations

Retirage corrigé avec nouvelle présentation (2019)
11 rue Paul Bert, 92240 Malakoff
www.dunod.com

ISBN 978-2-10-078005-1

Table des matières

Partie 1
Prise en main de Python et quelques pièges à éviter

1. Prise en main de Python **3**
2. Quelques pièges à éviter **21**

Partie 2
Algorithmes

3. Algorithmes à connaître **33**
4. Extraits de sujets de concours **45**

Partie 3
Simulation numérique

5. Méthode d'Euler – Problème de Cauchy **61**
6. Méthode des rectangles – Méthode des trapèzes **97**
7. Méthode de Gauss avec recherche partielle du pivot **103**
8. Traitement numérique du signal **113**

Partie 4
Fichiers

9. Gestion des fichiers **129**

Partie 5
Récursivité et piles

10. Récursivité **137**
11. Piles **157**

Partie 6
TRI

12. Algorithmes de tri **173**

Partie 7
APPLICATIONS

13. Traitement d'images **189**
14. Algorithmique des graphes **193**
15. Codage **201**

Partie 8
BASES DE DONNÉES

16. Requêtes SQL **217**

Index **231**

Les exercices sont destinés aux étudiants de Sup et Spé,
sauf ceux dont le titre porte la mention « (SPÉ) »,
qui sont réservés aux étudiants de Spé.

Partie 1

Prise en main de Python et quelques pièges à éviter

Plan

1. Prise en main de Python **3**
1.1 : Boucles, test, fonctions (banque PT 2015) 3
1.2 : Manipulations de listes et de tableaux numpy 7
1.3 : Tracé d'une fonction avec matplotlib 10
1.4 : Matrices 14
2. Quelques pièges à éviter **21**
2.1 : Variables locales, variables globales 21
2.2 : Copies de listes et de tableaux 22
2.3 : Passage par référence pour les listes et les tableaux numpy 24
2.4 : Slicing, ou découpage en tranches, et range, np.arange, np.linspace 27

1

Prise en main de Python

Exercice 1.1 : Boucles, test, fonctions (banque PT 2015)

1. Soit l'entier $n = 1\,234$. Quel est le quotient, noté q, de la division euclidienne de n par 10 ? Quel est le reste ? Que se passe-t-il si on recommence la division euclidienne par 10 à partir de q ?

Écrire une fonction `calcul_base10` d'argument `n`, renvoyant une liste `L` contenant les restes des divisions euclidiennes successives.

Écrire le programme principal demandant à l'utilisateur de saisir un entier `n` strictement positif et renvoyant la décomposition en base 10 de l'entier `n`.

2. Écrire une fonction `somcube`, d'argument `n`, renvoyant la somme des cubes des chiffres du nombre entier `n`. On pourra utiliser la fonction `calcul_base10`.

3. Écrire une fonction permettant de trouver tous les nombres entiers strictement inférieurs à 1 000 égaux à la somme des cubes de leurs chiffres.

4. Écrire une fonction `somcube2` qui convertit l'entier n en une chaîne de caractères permettant ainsi la récupération de ses chiffres sous forme de caractères. Cette nouvelle fonction renvoie la chaîne de caractères ainsi que la somme des cubes des chiffres de l'entier n. On pourra utiliser la fonction `str` et manipuler les chaînes de caractères.

Analyse du problème

Cet exercice est extrait du sujet de concours 0 de la banque PT 2015. Il permet de s'entraîner à manipuler les fonctions, les boucles, les tests et les différents types rencontrés dans Python 3.

Cours :

L'installation de Python 3 peut se faire très facilement avec la suite Anaconda 3 (recherche Google : « anaconda 3 python 3 »). On pourra utiliser Spyder comme éditeur qui s'installe automatiquement avec Anaconda.

Dans Python 3, 4/3 renvoie 1.333 alors que, dans Python 2, 4/3 renvoie 1, c'est-à-dire le quotient de la division euclidienne de 4 par 3 ! Les programmes seront réalisés exclusivement avec Python 3 dans cet ouvrage.

Une affectation dans Python se fait avec la syntaxe `n = 3` : n prend la valeur 3.

Il est préférable de ne pas utiliser d'accent pour les noms de variables. On peut utiliser « _ » dans le nom des variables mais pas « - ».

On peut ajouter des commentaires dans les programmes Python avec le symbole dièse : `# commentaire sur le programme`.

Le type d'une variable s'obtient avec la syntaxe `type(n)`.

Les types de base des variables dans Python sont :

- Entiers : `int`
  `n // 10` : quotient de la division euclidienne de *n* par 10
  `n % 10` : reste de la division euclidienne de *n* par 10
  `n ** 3` : *n* puissance 3
- Nombres à virgule flottante : `float`
- Opérations de base : `+, -, *, /`
- Booléens : `bool`. Les variables booléennes sont égales à `True` (vrai) ou `False` (faux).
- Opérateurs :
  `a == b` : cet opérateur compare *a* et *b*. Si *a=b*, Python retourne `True` sinon `False`
  `a != b` : *a* différent de *b*
  `a > b, a < b, a > = b, a < = b` : strictement supérieur, strictement inférieur, supérieur ou égal, inférieur ou égal
  `or` : ou
  `and` : et
  `not` : non

Les types de base des conteneurs dans Python sont :

- Chaînes de caractères : `str`

Exemple

```
s = "abcdef"   # on peut écrire également s = 'abcdef'
s[0]   # affiche le premier caractère, ici 'a'
s[-1]   # affiche le dernier caractère, ici 'f'
```

Pour extraire des caractères, voir exercice 2.4 « Slicing, ou découpage en tranches, et range, np.arange, np.linspace ».

- Listes : `list`

Exemple

```
L = [ ]   # création d'une liste vide
L.append(3)   # ajoute 3 dans la liste L
L.append(2)   # ajoute 2 dans la liste L
L.pop()   # efface le dernier élément (ici 2) de la liste L et retourne
          # sa valeur
L1 = ['r',3,'te']   # création d'une liste contenant des caractères et des
                    # entiers
len(L)   # affiche la longueur de la liste L
```

Pour extraire des éléments d'une liste, voir exercice 2.4 « Slicing, ou découpage en tranches, et range, np.arange, np.linspace ».

- Tuples : `tuple`. Une fois le tuple créé, il ne peut pas être modifié.
  `M = (2,3,8)` : création du tuple. On met des parenthèses. Ne pas confondre avec les listes, dans lesquelles on met des crochets.

- Tableaux numpy : pour leur utilisation, voir exercice 1.2 « Manipulations de listes et de tableaux numpy ».

Quelques fonctions intrinsèques :

`abs(x)` : renvoie la valeur absolue

`int(x)` : convertit *x* en entier

`float(x)` : convertit *x* en flottant

`str(x)` : convertit *x* en chaîne de caractères

`bool(x)` : convertit *x* en booléen

Première utilisation de la boucle `for` :

```
for i in range(n): # boucle faisant varier i de 0 à n−1
                   # ne pas oublier ':' à la fin de la ligne
        # voir exercice 2.4 "Slicing, ou découpage en tranches,
        # et range, np.arange, np.linspace"
    x = x + i   # ce code ajoute i à x à chaque passage dans la boucle
                # attention à l'indentation
```

Deuxième utilisation de la boucle `for` :

```
for elt in chaine:  # elt prend successivement les éléments de chaine
```

Pour l'utilisation de la boucle `while`, ne pas oublier :

```
i = 0
while i < 3:   # boucle exécutée tant que i < 3
    x = x + i  # attention à l'indentation
    i = i + 1
```

Définition d'une fonction :

```
def f(x):   # définition de la fonction f ayant pour argument d'entrée x.
            # ne pas oublier ':' à la fin de la ligne
    y = x + 3   # y est une variable locale : elle est créée à l'appel
                # de la fonction et est détruite à la fin de la fonction
            # voir exercice 2.1 "Variables locales, variables globales"
    return y   # fin de la fonction et retourne la valeur y
            # attention à l'indentation
```

Structure conditionnelle :

```
if x == 3:   # teste si x = 3
    y = 3 * x
elif x > 3 and x <= 4:   # si le test précédent n'est pas vérifié,
                         # alors teste si x > 3 et si x <= 4
    y = x + 2
elif x > 4 and x < 5:   # si le test précédent n'est pas vérifié,
                        # alors teste si x > 4 et si x < 5
    y = x − 2
else:   # sinon (les tests précédents ne sont pas vérifiés)
    y = 0
```

1. $n = 1\,234 = 123 \times 10 + 4$. Le reste vaut 4.

Si on recommence la division euclidienne de 123 par 10 : $123 = 12 \times 10 + 3$. Le reste vaut 3.

Si on recommence la division euclidienne de 12 par 10 : $12 = 1 \times 10 + 2$. Le reste vaut 2.

Si on recommence la division euclidienne de 1 par 10 : $1 = 0 \times 10 + 1$. Le reste vaut 1.

On obtient la décomposition en base 10 de n : $1, 2, 3, 4$.

```
def calcul_base10(n):
    L=[ ]   # création d'une liste vide
    while n > 0:   # boucle tant que n > 0
        q=n//10   # quotient de la division euclidienne de n par 10
        r=n%10   # reste de la division euclidienne de n par 10
        L.append(r)   # on ajoute le reste dans la liste L
        n=q
    L.reverse()   # renverse la liste L
    return L   # on retourne la liste L

# programme principal
n=int(input('Taper un entier strictement positif : '))
# conversion en entier du résultat de la saisie
print('Décomposition en base 10 : ',calcul_base10(n))
```

2.

```
def somcube(n):
    somme=0   # initialisation de la variable somme
    L=calcul_base10(n)   # on récupère la liste donnant
                         # la décomposition en base 10 de n
    for i in range(len(L)):  # i varie de 0 à len(L)
        somme=somme+L[i]**3  # on ajoute L[i] à la puissance 3
                             # on peut écrire somme+=L[i]**3
    return somme

# programme principal
n=1234
print(somcube(n))   # affiche 100 pour n = 1234
```

3.

```
def affiche_liste_entier_cube():   # pas d'argument d'entrée
    L=[ ]   # création d'une liste vide
    for i in range(1000):   # i varie entre 0 et 999
        if i==somcube(i):   # teste si i est égal à la somme des
                            # cubes de ses chiffres
           L.append(i)    # ajoute i dans la liste L
    return L  # fin de la fonction et renvoi de la liste L

# programme principal
print(affiche_liste_entier_cube())   # affiche
                                     # [0, 1, 153, 370, 371, 407]
```

4.

```
def somcube2(n):
    somme=0
    chaine=str(n)   # convertit n en une chaîne de caractères
    L=[ ]
    for elt in chaine:   # elt prend successivement les éléments
                         # de chaine
          L.append(elt)  # elt est un caractère que l'on ajoute
                         # dans L
          somme=somme+(int(elt))**3   # il faut convertir elt
                                      # en entier
    return L,somme

# programme principal
n=int(input('Taper un entier strictement positif : '))
L1,res=somcube2(n)   # L1 contient la liste des chiffres de n
                     # res = somme des cubes des chiffres de n
```

Exercice 1.2 : Manipulations de listes et de tableaux numpy

On considère la fonction f définie par : $f(x) = \sin(x) - x$.

1. Définir la fonction f dans Python en utilisant la bibliothèque `numpy`.

2. Définir une liste `X1` contenant 20 valeurs régulièrement espacées entre 10 et 30. Définir une liste `Y1` contenant 20 valeurs définies par $Y1[i] = f(X1[i])$. On utilisera la syntaxe `range`.

3. Définir un tableau numpy `X2` contenant 20 valeurs régulièrement espacés entre 10 et 30. Définir un tableau `Y2` contenant 20 valeurs définies par $Y[i] = f(X[i])$. On utilisera la syntaxe `range`.

4. Définir un tableau numpy `X3` contenant 20 valeurs régulièrement espacés entre 10 et 30 en utilisant la syntaxe `np.linspace`. Définir un tableau `Y3` contenant 11 valeurs définies par $Y[i] = f(X[i])$ sans utiliser la syntaxe `range`.

Analyse du problème

On va étudier dans cet exercice la différence entre les listes et les tableaux numpy. Il faut savoir manipuler aussi bien les listes que les tableaux numpy.

Voir exercice 13.1 « Traitement d'images et filtrage passe-bas » pour l'utilisation des matrices.

Cours :

Manipulation de listes avec Python

Exemple

```
L1 = [ ]   # création d'une liste vide
L2 = [1,2,'a','b']   # création d'une liste avec 4 éléments
L2.append(3)   # on ajoute 3 dans la liste L2. On a alors
               # L2 = [1,2,'a','b',3]
len(L2)   # renvoie la longueur de la liste L2, ici 5
x=L2.pop()    # efface le dernier élément de la liste et retourne sa valeur
              # on obtient L2 = [1,2,'a','b']
```