Exercices d'informatique - Corrigé de la série n° 2

Cours 3IN OC01

1. Les fonctions *divmod* et *pow*.
La fonction `divmod(a,b)` donne le quotient (entier) de a par b (c'est-à-dire `a//b`) et a modulo b (c'est-à-dire `a%b`). La fonction prédéfinie `pow(a,b)` renvoie le même résultat que `a**b` et `pow(a,b,3)` renvoie le même résultat que `(a**b)%3` mais plus rapidement.

2. Opérations sur les listes de nombres.
La fonction `min(l)` renvoie le plus petit nombre dans la liste `l`, `max(l)` donne le nombre le plus grand et `sum(l)` calcule la somme des nombres contenus dans la liste `l`.

3. Opérations sur les listes.
La fonction `len(l1)` donne le nombre d'éléments contenus dans la liste `l1`. L'opération `l1+l2` donne la concaténation de `l1` et `l2` et `2*l1` donne la concaténation de la liste `l1` avec elle-même.

4. Modification d'une liste.
Si `l` est une liste, l'instruction `l[i:j]=x` remplace dans `l` les éléments dont l'indice est $\geq i$ et $< j$ par `x`. L'instruction `l[1:1]= list ("texte")` insère les éléments de la liste `list ("texte")` avant l'élément qui se trouvait à l'indice `i` dans `l` avant l'affectation.

5. Modification d'une liste.
Nous constatons que

(1) `l.count("e")`: renvoie le nombre d'occurrences de `"e"` dans `l`,
(2) `l.index("t")`: renvoie l'indice de la première occurrence de `"t"` dans `l`,
(3) `l.append("vraiment petit")`: ajoute l'élément `"vraiment petit"` à la fin de `l`,
(4) `l.extend(list ("vraiment petit"))`: ajoute tous les éléments de la liste `list ("vraiment petit")` à la fin de `l`,
(5) `l.insert (3,"Z")`: insère l'élément `"Z"` à l'indice 3 dans `l`,
(6) `l.remove("e")`: supprime la première occurrence de `"e"` dans `l`,
(7) `l.pop(5)`: renvoie la valeur de l'élément à l'indice `5` et l'ôte de `l`; si `5` est omis, supprime et renvoie le dernier élément de `l`
(8) `l.reverse()`: renverse l'ordre de `l`.

6. La méthode *sort*.
La méthode `sort` d'une liste réordonne les éléments de cette liste. Cette méthode reconnaît trois paramètres optionnels.

7. Exceptions.

(1) Une erreur se produit.
(2) Le bloc d'instructions qui suit directement une instruction **try** est exécuté par Python sous réserve. Si une erreur survient pendant l'exécution de l'une de ces instructions, alors Python annule cette instruction fautive et exécute à sa place le code inclus dans le bloc qui suit l'instruction **except**.
(3) On peut, par exemple, utiliser le script suivant:

```
while True:
    n=raw_input('Tapez un nombre: ')
    try:
        print float(n)
        break
    except:
        print "Ce n'est pas un nombre !"
```

8. Monnaie optimale.

On peut, par exemple, utiliser le script suivant:

```
c=[1,2,5,10,20,50,100]
x=input('Tapez une somme:')
v=0
m=6
while v<x:
    while x-v<c[m]:
        m=m-1
    v=v+c[m]
    print c[m]
```

ou

```
c=[100,50,20,10,5,2,1]
x=input('Tapez une somme:')
r=x
for p in c:
    n=r/p
    if n!=0:
        print n,'X', p
        r=r%p
```

9. La factorielle.

On peut, par exemple, utiliser le script suivant

```
def fact(n):
    try:
        n=int(n)
        x=1
        for k in range(1,n+1,1):
            x=x*k
        print "n!=",x
    except:
        print "Vous devez entrer un nombre entier !"
while True:
    n=raw_input("n=")
    fact(n)
```

10. L'instruction *def*.

L'instruction **def** permet de définir une fonction. Elle s'exécute si et seulement si elle est appelée (dans notre exemple par la ligne bernard()).

11. L'instruction *def*.

Les termes en parenthèses dans la ligne de l'instruction **def**, s'il y en a, sont les paramètres utilisés par la fonction. Ces paramètres sont obligatoires. Chaque appel à la fonction doit fournir une valeur pour chaque paramètre obligatoire. Dans notre exemple, la fonction bernard requiert un paramètre. L'appel de la fonction par la ligne bernard() provoque une erreur lors de l'exécution du script.

12. L'instruction *return*.

L'instruction permet de définir une fonction qui retourne une ou plusieurs valeurs.

13. Les paramètres facultatifs d'une fonction.

Dans cet exemple, le paramètre est facultatif. Sa valeur par défaut est 0.

14. Attributs d'une fonction.

L'attribut .func_name donne le nom de la fonction, .func_defaults le tuple contenant les valeurs par défaut et .func_doc la chaîne de documentation de la fonction. Toute fonction devrait contenir une chaîne de documentation pour faciliter son utilisation.

15. Attributs d'une fonction.

En plus de ses attributs prédéfinis, on peut doter une fonction d'attributs quelconques. Dans cet exemple, la fonction *essai* a un attribut .compteur qui contient le nombre d'appel à la fonction dans le script.

16. Variables locales.

Les variables a et y sont définies en dehors de la fonction. La valeur de a est accessible dans le corps de la fonction. Une variable y est également définie dans le corps de la fonction. Les modifications apportées à la valeur de y dans la fonction n'affectent pas la valeur de la variable y hors de la fonction. On dit que la variable y définie dans le corps de la fonction est une variable locale.

17. Variables locales.

L'affectation y=2 dans le corps de la fonction ne change pas la valeur de la variable y dans le corps du programme.

18. Suites de Syracuse.

(1) La suite de Syracuse de 1 est donnée par: $1\ 4\ 2\ 1\ 4\ 2\ 1\ \cdots$

(2) On peut utiliser le script suivant:

```
1  def f(n):
2      if n%2==0:
3          a=n/2
4      else:
5          a=3*n+1
6      return a
7  print "Tapez un nombre entier >0: ",
8  n=input()
9  print "u_ 0  = ",n
10 k=0
11 while n>1:
12     k=k+1
13     n=f(n)
14     print "u_",k," = ",n
```

19. Suites de Syracuse.

On peut, par exemple, utiliser le script suivant:

```
1  def f(n):
2      f.k=f.k+1
3      if n%2==0:
4          a=n/2
5      else:
6          a=3*n+1
7      if a>f.max:
8          f.max=a
9      if a<n0 and f.vol==0:
```

```
10          f.vol=f.k-1
11      return a
12  print "Tapez un nombre entier >0: ",
13  n=input()
14  n0=n
15  f.k=0
16  f.max=0
17  f.vol=0
18  while n>1:
19      n=f(n)
20  print "Altitude maximale=",f.max
21  print "Temps de vol=",f.k
22  print "Temps de vol en altitude=",f.vol
```

20. L'instruction *global*.

Dans le premier script, la ligne `a=2` dans le corps de la fonction ne change pas la valeur de la variable `a` du corps du programme. Cette ligne permet simplement de déclarer une variable locale `a` dans le corps de la fonction. L'instruction **global** au début de la fonction `gnu` du second script permet de modifier dans le corps de la fonction la valeur de la variable `a` du programme.

21. L'instruction *import*.

Le résultat est le même dans les trois cas.

22. *Lambda*-expressions.

Les résultats sont identiques. Quand le corps d'une fonction contient seulement une instruction **return**, on peut remplacer cette fonction par la forme spéciale **lambda**.

23. La factorielle.

On peut, par exemple, utiliser le script suivant:

```
1  def fact(n):
2      if n>1:
3          m=fact(n-1)
4          return n*m
5      else:
6          return n
7  n=input('n=')
8  print fact(n)
```

24. Fonction récursive.

On peut, par exemple, utiliser le script suivant:

```
1  def f(n):
2      if n==0:
3          return 1.
4      else:
5          return 1./2**n+f(n-1)
6  n=input("Tapez un nombre entier: n=")
7  print "1+1/2+1/4+...+1/n^2=",f(n)
```

25. Fonction récursive.

On peut, par exemple, utiliser le script suivant:

```
from math import *
def f(n):
    if n==0:
        return 1.
    else:
        return 1./(2*n+1)**2+f(n-1)
n=input("Tapez un nombre entier: n=")
print "1+1/9+1/25+...+1/(2n+1)^2=",f(n)
print "pi^2/8=",pi**2/8
```

26. Fonction récursive.

On peut, par exemple, utiliser le script suivant:

```
def f(n):
    if n==1:
        return 1./2
    else:
        return 1./(n*(n+1))+f(n-1)
n=input("Tapez un nombre entier: n=")
print "1/(1*2)+1/(2*3)+...+1/(n*(n+1))=",f(n)
```