Devoir sur table n°2

MATHÉMATIQUES

Samedi 19 Novembre

Rappel des consignes

Lorsqu'on écrit un code Python, :

- faire attention à ce que les indentations soient visibles sur la copie ;
- commenter le code de façon à expliquer les grandes étapes de l'algorithme en ajoutant un commentaire en fin de ligne de code après le symbole ♯.

Exercice 1

Ecrire une fonction `renverser` qui à une liste renvoie la liste renversée.
Remarque : On n'utilisera **pas** la méthode `reverse` déjà implémentée dans Python.

```
>>>L=[2,8,-1,7]
>>>renverser(L)
[7,-1,8,2]
```

Exercice 2

L'objectif de cet exercice est de faire une liste des triangles qui vérifient les trois conditions suivantes :

- les côtés des triangles sont de mesure entière ;
- les triangles sont rectangles ;
- les triangles sont de périmètre p (la valeur de p étant fixée).

1) Une fonction rectangle :

   (a) Ecrire une fonction `rectangle(a,b,c)` qui prend comme entrée trois entiers positifs et renvoie `True` si le triangle dont les côtés de mesures a, b et c est un triangle rectangle et `False` sinon.
   Exemple :

   ```
   rectangle(4,3,5)
   >>> True
   rectangle(2,7,1)
   >>> False
   ```

   Solution.

```
def rectangle(a,b,c):
    A=a**2
    B=b**2
    C=c**2
    if A==B+C or B==A+C or C==A+B:
        return(True)
    else:
        return(False)
```

(b) On note N_{rect} le nombre d'opérations de cet algorithme. Calculer N_{rect}.

Solution.
- Trois affectations, trois multiplications
- trois tests avec trois additions
- un return

$N_{rect} = 13$

2) Une première fonction :

(a) On considère la fonction `triangle` suivante. Que renvoie-t-elle ?

```
def triangle(p):
   Liste=[]
   for a in range(1,p+1):
      for b in range(1,p+1):
         for c in range(1,p+1):
            if rectangle(a,b,c):
               Liste.append((a,b,c))
   return(Liste)
```

Solution. La fonction construit tous les triplets (a, b, c) d'entiers compris entre 1 et p. Elle ne garde que ceux qui sont les mesures d'un triangle rectangle. Le périmètre de ces triangles par contre n'est pas fixé. Il peut varier entre 3 et $3p$.

(b) Modifier la fonction `triangle` en une fonction `triangle2` pour qu'elle renvoie la liste des triangles qui vérifient les trois conditions énoncées au début de l'exercice.

Solution.
```
def triangle2(p):
   Liste=[]
   for a in range(1,p+1):
      for b in range(1,p+1):
         for c in range(1,p+1):
            if a+b+c==p:
               if rectangle(a,b,c):
                  Liste.append((a,b,c))
   return(Liste)
```

(c) Déterminer le nombre d'opérations effectuées dans la fonction `triangle2`.

Solution. Pour la fonction `triangle2` :

- une assignation
- p itérations de chaque boucle, donc p^3 itérations de :
  - 2 additions
  - 1 test
  - fonction rectangle : N_{rect}=13
  - 1 test
  - un append
- un return

Le nombre d'opérations du premier algorithme est : $\boxed{N_{op} = 2 + 18p^3}$

3) Une deuxième fonction :

   (a) On introduit la fonction suivante :

```
def triangle3(p):
   Liste=[]
   for a in range(1,p//3+1):
      for b in range(a,(p-a)//2+1):
            if rectangle(a,b,p-a-b):
               Liste.append((a,b,p-a-b))
   return(Liste)
```

Expliquer pourquoi cette fonction renvoie aussi la liste des triangles cherchés.

Solution. La liste renvoyée contient des triplets (a, b, c) tels que (a, b, c) vérifient la condition `rectangle` et $c = p-a-b$. Donc ces triplets sont associés à des triangles rectangles de périmètre p.
Vérifions qu'on obtient bien tous les triangles cherchés avec cette fonction :
`a,b` et `c` sont les trois longueurs des côtés. On choisit qu'ils seront dans l'ordre croissant : $a \leqslant b \leqslant c$.
Le plus petit des côtés est nécessairement inférieur à $\frac{p}{3}$. En effet, si $a > \frac{p}{3}$, alors le périmètre est : $a + b + c > 3 \times p/3 = p$. Le périmètre du triangle ne peut pas être p.
Comme a est un entier : $a \leqslant \lfloor \frac{p}{3} \rfloor$. Donc a parcourt : $1, 2, \cdots \lfloor \frac{p}{3} \rfloor$.
`p//3` renvoie le quotient dans la division euclidienne de p par 3, c'est donc $\lfloor \frac{p}{3} \rfloor$. Alors `range(1,p//3+1)` est la liste $[1, 2, 3, \cdots, \lfloor \frac{p}{3} \rfloor]$.
Maintenant que a est fixé, les deux autres longueurs vérifient : $b \geqslant a$ et $b+c = p-a$. Encore une fois, l'une des deux longueurs est nécessairement inférieures à $\frac{p-a}{2}$ (si b et c sont strictement plus grand que $\frac{p-a}{2}$, alors $a + b + c > p$.) On choisit : $b \leqslant \frac{p-a}{2}$. Donc `b in range(a,(p-a)//2+1)`.
Enfin, comme la longueur du périmètre est fixée : $c = p - a - b$.
On obtient donc bien la liste cherchée.

   (b) Montrer que le nombre d'opérations N_{op} de la fonction `triangle3` vérifie :

$$N_{op} \leqslant 2 + 17 \left(\frac{p^2}{12} + \frac{p}{12} \right)$$

Solution. Second algorithme :

- une assignation
- $\lfloor \frac{p}{3} \rfloor$ itérations de la première boucle,
- $\lfloor \frac{p-a}{2} \rfloor$ itérations de la deuxième boucle,
  - 2 soustractions
  - fonction rectangle : N_{rect}=13
  - 1 test
  - un append
- un return

La complexité du premier algorithme est : $N_{op} = 2 + \sum_{a=1}^{\lfloor \frac{p}{3} \rfloor} \sum_{b=a}^{\lfloor \frac{p-a}{2} \rfloor} 17.$

On utilisera l'encadrement : $x - 1 \leqslant \lfloor x \rfloor \leqslant x$.

$$N_{op} = 2 + 17 \sum_{a=1}^{\lfloor \frac{p}{3} \rfloor} (\lfloor \frac{p-a}{2} \rfloor - a + 1) \leqslant 2 + 17 \sum_{a=1}^{\lfloor \frac{p}{3} \rfloor} (\frac{p-a}{2} + 1 - a)$$

$$N_{op} \leqslant 2 + 17 \sum_{a=1}^{\lfloor \frac{p}{3} \rfloor} (\frac{p}{2} + 1 - \frac{3}{2}a) = 2 + 17 \left[(\frac{p}{2} + 1)(\lfloor \frac{p}{3} \rfloor) - \frac{3}{2} \frac{\lfloor \frac{p}{3} \rfloor (\lfloor \frac{p}{3} \rfloor + 1)}{2} \right].$$

$$N_{op} \leqslant 2 + 17\frac{p}{3}\left[(\tfrac{p}{2}+1) - \tfrac{3}{2}\frac{(\frac{p}{3}+1)}{2}\right] = 2 + 17(\frac{p^2}{12} + \frac{p}{12}).$$

$$\boxed{N_{op} \leqslant 2 + 17\left((\frac{p^2}{12} + \frac{p}{12}\right)}$$

(c) Comparer la complexité des deux algorithmes. Lequel est le plus rapide pour des grandes valeurs de p ?

Solution. Pour le premier, $N_{op} = 2 + 18p^3$. Donc la complexité est en $O(p^3)$.

Pour le second, $N_{op} \leqslant 2 + 17\left((\frac{p^2}{12} + \frac{p}{12}\right)$. Donc la complexité est en $O(p^2)$.

Quand p est grand, le deuxième algorithme est le plus rapide.