Exercices récursivité
Correction

Christophe Viroulaud

Terminale - NSI

Lang 06

Sommaire

Exercice 1

```
1  COMPTEUR = 0
2
3  def puissance_perso(x: int, n: int) -> int:
4      global COMPTEUR
5      res = 1
6      for i in range(n):
7          COMPTEUR += 1
8          res *= x
9      return res
10
11 puissance_perso(2701, 19406)
12 print("perso: ", COMPTEUR)
```

Code 1 – La fonction effectue 19406 itérations.

```python
def puissance_recursif(x: int, n: int) -> int:
    global COMPTEUR
    if n == 0:
        return 1
    else:
        COMPTEUR += 1
        return x*puissance_recursif(x, n-1)
```

Code 2 – La fonction effectue également 19406 itérations.

Remarque

Ne pas oublier de remettre `COMPTEUR` à 0.

```
1  def puissance_recursif_rapide(x, n):
2      global COMPTEUR
3      if n == 0:
4          return 1
5      elif n % 2 == 0:
6          COMPTEUR += 1
7          return puissance_recursif_rapide(x*x, n
       //2)
8      else:
9          COMPTEUR += 1
10         return x*puissance_recursif_rapide(x*x, n
       //2)
```

Code 3 – Il n'y a que 15 itérations.

```
1  def puissance_iteratif_rapide(x, n):
2      global COMPTEUR
3      res = 1
4      while n > 0:
5          COMPTEUR += 1
6          if n % 2 != 0: #impair
7              res = res * x
8          x = x*x
9          n = n//2
10     return res
```

Code 4 – Version itérative

Sommaire

Exercice 2

$$somme(n) = \begin{cases} 0 & \text{si } n = 0 \\ \text{n} + \text{somme(n-1)} & \text{si } n > 0 \end{cases}$$

```
1 def somme(n: int) -> int:
2     if n == 0:
3         return 0
4     else:
5         return n + somme(n-1)
```

Code 5 – Somme des entiers

Hors programme

Une fonction à récursivité terminale est une fonction où l'appel récursif est la dernière instruction à être évaluée.

```
1 def somme_terminale(n: int, res: int) -> int:
2     if n == 0:
3         return res
4     else:
5         return somme_terminale(n-1, res+n)
```

Code 6 – Version *terminale*

Sommaire

Exercice 3

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ \text{n} \times (\text{n-1})\,! & \text{si } n > 0 \end{cases}$$

```
1 def factorielle(n: int)->int:
2     if n == 0:
3         return 1
4     else:
5         return n * factorielle(n-1)
```

Sommaire

Exercice 4

```
1  def syracuse(u: int) -> None:
2      print(u, end=" ")
3      if u > 1: # cas limite
4          if u % 2 == 0:
5              syracuse(u // 2)
6          else:
7              syracuse(3 * u + 1)
8
9  print(syracuse(5))
```

```python
def syracuse2(u: int, l: list) -> list:
    l.append(u)
    if u > 1:
        if u % 2 == 0:
            syracuse2(u // 2, l)
        else:
            syracuse2(3 * u + 1, l)
    return l

print(syracuse2(5, []))
```

Code 7 – Version avec renvoi des valeurs dans un tableau.

Sommaire

Exercice 5

```python
def entiers(i: int, k: int) -> None:
    if i <= k:
        print(i, end=" ")
        entiers(i+1, k)
```

```
1  def impairs(i: int, k: int) -> None:
2      if i <= k:
3          if i % 2 == 1:
4              print(i, end=" ")
5          impairs(i+1, k)
```

Sommaire

Exercice 6

```
1 from random import randint
2
3 t = [randint(1, 100) for _ in range(10)]
```

```
1 def somme(tab: list) -> int:
2     s = 0
3     for i in range(len(tab)):
4         s += tab[i]
5     return s
6
7 somme(t)
```

```python
def somme_rec(tab: list, i: int) -> int:
    """
    calcule la somme des éléments du tableau
    Args:
        tab (list): le tableau
        deb (int): indice de l'élément en cours

    Returns:
        int: la somme
    """
    if i == len(tab):
        return 0
    else:
        return tab[i] + somme_rec(tab, i+1)

somme_rec(t, 0)
```

Code 8 – Version récursive

```python
def somme_rec_term(tab: list, i: int, s: int) -> int:
    """
    version terminale
    Args:
        tab (list): le tableau
        i (int): indice de l'élément en cours
        s (int): somme

    Returns:
        int: la somme
    """
    if i == len(tab):
        return s
    else:
        return somme_rec_term(tab, i+1, s+tab[i])

somme_rec_term(t, 0, 0)
```

Code 9 – Version récursive terminale

```
1 def somme_rec(tab: list, s: int) -> int:
2     if len(tab) == 0:
3         return s
4     else:
5         tete = tab[0]
6         queue = tab[1:] # slice
7         return somme_rec(queue, s+tete)
8
9 somme_rec(t, 0)
```

Code 10 – Autre version récursive avec slice

Remarque

Le *slice* est hors programme.

Sommaire

Exercice 7

```
1 t = [randint(1, 1000) for _ in range(30)]
```

```
1  def mini(tab: list) -> int:
2      m = float("inf")
3      for i in range(len(tab)):
4          if tab[i] < m:
5              m = tab[i]
6      return m
7
8  mini(t)
```

```python
def mini_rec(tab: list, i: int, m: int) -> int:
    """
    cherche le plus petit élément du tableau

    Args:
        tab (list): le tableau
        i (int): indice de l'élément en cours
        m (int): l'élément mini
    """
    if i == len(tab):
        return m
    else:
        if tab[i] < m:
            m = tab[i]
        return mini_rec(tab, i+1, m)

mini_rec(t, 0, float("inf"))
```

```python
def mini_rec2(tab: list, m: int) -> int:
    """
    avec slice
    """
    if len(tab) == 0:
        return m
    else:
        if tab[0] < m:
            m = tab[0]
        return mini_rec2(tab[1:], m)

mini_rec2(t, float("inf"))
```

Code 11 – Avec slice

Sommaire

Exercice 8

$a = 20, b = 35$

$$35 = \overbrace{20}^{a \to b} \times 1 + \overbrace{15}^{b\%a \to a}$$

$20 = 15 \times 1 + 5$

$15 = 5 \times 3 + 0$

$\text{pgcd} = 5$

```python
def pgcd(a: int, b: int) -> int:
    while a != 0:
        a, b = b % a, a
    return b
```

Code 12 – Version itérative

```
1 def pgcd_rec(a: int, b: int) -> int:
2     if a == 0:
3         return b
4     else:
5         return pgcd_rec(b % a, a)
```

Code 13 – Version récursive

Sommaire

Exercice 9

```python
def nombre_chiffres(n: int) -> int:
    if n < 10:
        return 1
    else:
        return 1 + nombre_chiffres(n//10)
```

Code 14

```
1  def nombre_chiffres_terminal(n: int, acc: int
       ) -> int:
2      if n < 10:
3          return acc
4      else:
5          return nombre_chiffres_terminal(n
       //10, acc+1)
```

Code 15 – Version *terminale*

Sommaire

Exercice 10

```
1  def C(n: int, p: int) -> int:
2      if p == 0 or n == p:
3          return 1
4      else:
5          return C(n-1, p-1) + C(n-1, p)
```

Exercice 10

```
1 for n in range(10): # chaque ligne
2     for p in range(n+1): # chaque élément de
    la ligne
3         print(C(n, p), end=" ")
4     print() # saut de ligne
```