

Exercices récursivité Correction

Christophe Viroulaud

Terminale - NSI

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

6. Exercice 6

7. Exercice 7

8. Exercice 8

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

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.

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

```
1 def puissance_recuratif(x: int, n: int) -> int:  
2     global COMPTEUR  
3     if n == 0:  
4         return 1  
5     else:  
6         COMPTEUR += 1  
7         return x*puissance_recuratif(x, n-1)
```

Code 2 – La fonction effectue également 19406 itérations.

Remarque

Ne pas oublier de remettre **COMPTEUR** à 0.

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

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

Code 3 – Il n'y a que 15 itérations.

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

```
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

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

6. Exercice 6

7. Exercice 7

8. Exercice 8

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 2

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

```
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*

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

6. Exercice 6

7. Exercice 7

8. Exercice 8

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 3

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

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

Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

6. Exercice 6

7. Exercice 7

8. Exercice 8

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercise 4

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

```
1 def syracuse2(u: int, l: list) -> list:
2     l.append(u)
3     if u > 1:
4         if u % 2 == 0:
5             syracuse2(u // 2, l)
6         else:
7             syracuse2(3 * u + 1, l)
8     return l
9
10 print(syracuse2(5, []))
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

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

Sommaire

1. Exercice 1

Exercice 1

2. Exercice 2

Exercice 2

3. Exercice 3

Exercice 3

4. Exercice 4

Exercice 4

5. Exercice 5

Exercice 5

6. Exercice 6

Exercice 6

7. Exercice 7

Exercice 7

8. Exercice 8

Exercice 8

Exercice 9

Exercice 10

Exercice 5

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)

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

Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

6. Exercice 6

7. Exercice 7

8. Exercice 8

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 6

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 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)
```

```
1 def somme_rec(tab: list, deb: int, s: int) -> int:
2     """
3     calcule la somme des éléments du tableau
4
5     Args:
6         tab (list): le tableau
7         deb (int): indice de l'élément en cours
8         s (int): somme
9     """
10    if deb == len(tab):
11        return s
12    else:
13        return somme_rec(tab, deb+1, s+tab[deb])
14
15    somme_rec(t, 0, 0)
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Code 8 – Version récursive

```
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 9 – Autre version récursive avec slice

Remarque

Le *slice* est hors programme.

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

6. Exercice 6

7. Exercice 7

8. Exercice 8

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 7

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

```
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)
```

```
1 def mini_rec(tab: list, deb: int, m: int) -> int:
2     """
3     cherche le plus petit élément du tableau
4
5     Args:
6         tab (list): le tableau
7         deb (int): indice de l'élément en cours
8         m (int): l'élément mini
9     """
10    if deb == len(tab):
11        return m
12    else:
13        if tab[deb] < m:
14            m = tab[deb]
15        return mini_rec(tab, deb+1, m)
16
17 mini_rec(t, 0, float("inf"))
```

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)

```
1 def mini_rec2(tab: list, m: int) -> int:
2     """
3     avec slice
4     """
5     if len(tab) == 0:
6         return m
7     else:
8         if tab[0] < m:
9             m = tab[0]
10            return mini_rec2(tab[1:], m)
11
12 mini_rec2(t, float("inf"))
```

Code 10 – Avec slice

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

6. Exercice 6

7. Exercice 7

8. Exercice 8

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercise 8

$$a = 20, b = 35$$

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

$$20 = 15 \times 1 + 5$$

$$15 = 5 \times 3 + 0$$

$$\text{pgcd} = 5$$

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

```
1 def pgcd(a: int, b: int) -> int:  
2     while a != 0:  
3         a, b = b % a, a  
4     return b
```

Code 11 – Version itérative

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

```
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 12 – Version récursive

Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

5. Exercice 5

6. Exercice 6

7. Exercice 7

8. Exercice 8

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercise 9

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

Code 13

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

```
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 14 – Version *terminale*

Sommaire

1. Exercice 1

Exercice 1

2. Exercice 2

Exercice 2

3. Exercice 3

Exercice 3

4. Exercice 4

Exercice 4

5. Exercice 5

Exercice 5

6. Exercice 6

Exercice 6

7. Exercice 7

Exercice 7

8. Exercice 8

Exercice 8

Exercice 9

Exercice 10

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 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

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
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10