

Exercices récursivité correction

Christophe Viroulaud

Terminale - NSI

Lang 08



Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

- 1. Exercice 1
- 2. Exercice 2
- 3. Exercice 3
- 4. Exercice 4
- 5. Exercice 5
- 6. Exercice 6
- 7. Exercice 7
- 8. Exercice 8
- 9. Exercice 9
- 10. Exercice 10
- 11. Exercice 11
- 12. Exercice 12
- 13. Exercice 13
- 14. Exercice 14

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Exercice 1

```
1 def somme(n: int) -> int:  
2     s = 0  
3     while n > 0:  
4         s = s + n  
5         n = n-1  
6     return s
```

Code 1 – Version itérative

La complexité est linéaire : elle est proportionnelle à n .

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

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

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

Code 2 – Version récursive

La complexité est linéaire : elle est proportionnelle à n .

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

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 3 – Version *terminale*

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

```
1 def somme_rapide(n: int) -> int:  
2     return n*(n+1)//2
```

Code 4 – Version rapide

La complexité est constante : on effectue toujours le même nombre d'opérations.

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

Exercice 2

```
1 def factorielle(n: int) -> int:  
2     total = 1  
3     while n > 0:  
4         total = total*n  
5         n = n-1  
6     return total
```

Code 5 – Version itérative

La complexité est linéaire : elle est proportionnelle à n .

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

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

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

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

Code 6 – Version récursive

La complexité est linéaire : elle est proportionnelle à n.

Exercice 3

```
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 11

Exercice 12

Exercice 13

Exercice 14

```
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)
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

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 # programme principal
10 syracuse(5)
```

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

```
1 def syracuse2(u: int, t: list) -> None:  
2     t.append(u)  
3     if u > 1:  
4         if u % 2 == 0:  
5             syracuse2(u // 2, t)  
6         else:  
7             syracuse2(3 * u + 1, t)  
8  
9  
10 tab = []  
11 syracuse2(5, tab)  
12 print(tab)
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

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

Remarque

Quelle que soit la valeur de u_0 il existe un n tel que $u_n = 1$. Cette conjecture n'est toujours pas prouvée à ce jour.

Exercice 5

```
1 def suite(n: int) -> int:  
2     if n == 0 or n == 1:  
3         return n  
4     else:  
5         return 3*suite(n-1) + 2*suite(n-2) + 5  
6  
7  
8 print(suite(5))
```

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

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 11

Exercice 12

Exercice 13

Exercice 14

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Code 8 – Version récursive

```
1 def somme_rec_term(tab: list, i: int, s: int) -> int:  
2     """  
3         version terminale  
4     Args:  
5         tab (list): le tableau  
6         i (int): indice de l'élément en cours  
7         s (int): somme  
8  
9     Returns:  
10        int: la somme  
11    """  
12    if i == len(tab):  
13        return s  
14    else:  
15        return somme_rec_term(tab, i+1, s+tab[i])  
16  
17 # programme principal  
18 somme_rec_term(t, 0, 0)
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Code 9 – Version récursive terminale

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 11

Exercice 12

Exercice 13

Exercice 14

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

```
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 # programme principal
9 mini(t)
```

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

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

Exercice 8

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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 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)
```

Exercice 1

Exercice 2

ercice 3

ercice 4

ercice 5

ercice 6

ercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Code 10 – **Hors programme** : Version *terminale*

Exercice 9

```
1 def somme(n: int) -> int:
2     if n // 10 == 0:
3         return n
4     else:
5         return n % 10 + somme(n // 10)
6
7 # programme principal
8 print(somme(5425))
```

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

Exercice 10

```
1 def compter_chiffres(nb: int, chiffres: list) -> None:
2     if nb < 10:
3         chiffres[nb] = chiffres[nb]+1
4     else:
5         unite = nb % 10
6         chiffres[unite] = chiffres[unite]+1
7         compter_chiffres(nb//10, chiffres)
8
9 # Programme principal
10 chiffres = [0 for _ in range(10)]
11 compter_chiffres(2020, chiffres)
12 print(chiffres)
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Exercice 11

```
1 import doctest
2
3 def zenon(d: float, n: int, n_max: int) -> float:
4     """
5         >>> zenon(4, 1, 2)
6         6.0
7         >>> zenon(4, 1, 20)
8         7.999992370605469
9     """
10    if n == n_max:
11        return d
12    else:
13        d_next = d+4/2**n
14        return zenon(d_next, n+1, n_max)
15
16 # programme principal
17 doctest.testmod(verbose=True)
```

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

```
1 def zenon2(d: float, n: int) -> float:  
2     """  
3         >>> zenon2(4, 2)  
4         6.0  
5         >>> zenon2(4, 20)  
6         7.999992370605469  
7     """  
8     if n == 1:  
9         return d  
10    else:  
11        d_next = 0.5*d+4  
12        return zenon2(d_next, n-1)
```

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

Exercice 12

```
1 import doctest
2
3
4 def heron(x: float, a: int, n: int) -> float:
5     """
6         >>> heron(1, 2, 5)
7             1.414213562373095
8         >>> heron(2, 5, 5)
9             2.23606797749979
10    """
11    if n == 0:
12        return x
13    else:
14        x_next = (x+a/x)/2
15        return heron(x_next, a, n-1)
16
17
18 doctest.testmod(verbose=True)
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Exercice 13

$$a = 20, b = 35$$

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

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

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

$$\text{pgcd} = 5$$

[Exercice 1](#)[Exercice 2](#)[Exercice 3](#)[Exercice 4](#)[Exercice 5](#)[Exercice 6](#)[Exercice 7](#)[Exercice 8](#)[Exercice 9](#)[Exercice 10](#)[Exercice 11](#)[Exercice 12](#)[Exercice 13](#)[Exercice 14](#)

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

Code 12 – Version itérative

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

```
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

Exercice 1

Exercice 2

Exercice 3

Exercice 4

Exercice 5

Exercice 6

Exercice 7

Exercice 8

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Exercice 14

```
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 11

Exercice 12

Exercice 13

Exercice 14

```
1 for n in range(10): # chaque ligne
2     for p in range(n+1): # chaque élément de
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

Exercice 11

Exercice 12

Exercice 13

Exercice 14