Exercices récursivité Correction

Exercices récursivité
Correction

Christophe Viroulaud

Terminale - NSI

Lang 06

exercice 1

xercice 2

xercice 3

kercice s

vercice i

exercice 10

Exercices récursivité Correction

- 1. Exercice 1
- 2. Exercice 2
- 3. Exercice 3
- / Evercice
- 5 Exercice 5
- 6 Exercice
- 7 Evercice
- 8. Exercice

Exercice 1

cice 2

ice 3

rcice 4

ice 6

ice 7

ercice 9

rcice 9

cice 10

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

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

```
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.

Exercice 1

xercice 3

.

Exercice 6

. . .

xercice 8

exercice 9

Exercice 1

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

Code 3 – Il n'y a que 15 itérations.

5 / 40

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

Code 4 – Version itérative

vercice S

varcica 3

Exercice

Exercice 5

exercice 9

Exercices récursivité Correction

- 1. Exercice 1
- 2. Exercice 2
- 3. Exercice 3
- 4 Evereice
- 5 Exercice 5
- 6 Exercice
- 7 Exercice
- 8. Exercice

Exercice 2

rcice 2

ercice 3

rcice 5

ice 6

rcice 8

ercice 9

rcice 10

xercice 3

tercice 8

xercice 9

```
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

xercice 1

Exercice 2

ercice 3

......

ercice 5

exercice 9

Hors programme

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

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

Code 6 - Version terminale

Exercice 1

Exercice 2

xercice 3

xercice 5

.

.

xercice 9

Exercices récursivité Correction

- 1. Exercice
 - ice 2
- 3. Exercice 3
- 4 Evereice
- 5 Evercice 5
- 6 Evereice
- 7 Europaine
- 8. Exercice

cice 2

Exercice 3

cice 3

cice 5

cice 7

rcice 8

cice 9

rcice 10

JCC 10

Exercice 3

Kercice =

xercice 5

. .

Exercice 9

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

xercice 2

Exercice 3

tercice 4

xercice 5

xercice 6

xercice 7

ercice 8

Exercice 9

Exercices récursivité Correction

- 4. Exercice 4

Exercice 4

14 / 40

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

xercice 1

Exercice :

Exercice 4

vorcico 6

ercice 7

.....

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

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

Exercice 4

Exercice 6

CI CICC I

xercice 9

Exercices récursivité Correction

- 1. Exercice
 - ice 2
- 3 Exercice 3
- 4 Exercice 4
- 5. Exercice 5
- 6 Exercice
- 7 Exercice
- 8. Exercice

--:-- 0

cice 3

cice 4

Exercice 5

ice 6

cice 7

rcice 9

--:-- 10

cice 10

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

xercice :

kercice 3

Exercice 5

Kercice U

ercice 7

Aercice 9

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

xercice :

ercice 3

kercice 4

Exercice 5

xercice o

ercice 7

Kercice o

Exercice 9

Exercices récursivité Correction

- 6. Exercice 6

1 2

3

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

xercice :

. .

ercice 4

ercice 5

Exercice 6

xercice 7

ercice 8

xercice 9

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

kercice 2

orcico 3

xercice 4

xercice 5

Exercice 6

. .

```
def somme_rec(tab: list, i: int) -> int:
    11 11 11
    calcule la somme des éléments du tableau
    Args:
        tab (list): le tableau
        deb (int): indice de l'élément en cours
    Returns:
        int: la somme
    11 11 11
    if i == len(tab):
        return 0
    else:
        return tab[i] + somme rec(tab, i+1)
somme_rec(t, 0)
```

2

5

6

8

9

10

11

12

13

14 15

16

Code 8 - Version récursive

```
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
                                                           Exercice 6
    Returns:
        int: la somme
    11 11 11
    if i == len(tab):
        return s
    else:
        return somme rec term(tab, i+1, s+tab[i])
```

Code 9 – Version récursive terminale

3

10

11 12

13

14

15 16

17

somme rec term(t, 0, 0)

```
def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
        return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
        return s

def somme_recurin somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab) == 0:
    return s

def somme_rec(tab: list, s: int) -> int:
    if len(tab: list, s: int) -> int:
    if
```

Code 10 – Autre version récursive avec slice

Remarque

Le *slice* est hors programme.

rcice 1

. .

Exercice 3

Xercice 4

xercice 5

vercice 10

Exercices récursivité Correction

1. Exercice

cice 2

B. Exercice 3

4. Exercice

5. Exercice 5

6 Exercice (

7. Exercice 7

8. Exercice

rcice 1

cice 2

rcice 4

ercice 5

Exercice 7

ercice 8

ercice 9

cice 10

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

vercice 2

varcica 3

Xercice 4

kercice b

Xercice 0

Exercice 7

refere 0

ercice 2

ercice 3

xercice 4

versice 5

xercice 6

Exercice 7

roreico O

```
def mini rec(tab: list, i: int, m: int) -> int:
    11 11 11
    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
    11 11 11
    if i == len(tab):
        return m
    else:
        if tab[i] < m:</pre>
             m = tab[i]
        return mini_rec(tab, i+1, m)
mini rec(t, 0, float("inf"))
```

1

2

4 5

6

8

9

10 11

12

13

14

15 16

17

ercice 1

ercice 2

kercice 3

Exercice 7

kercice 8

xercice 9

```
def mini rec2(tab: list, m: int) -> int:
1
        11 11 11
2
        avec slice
3
        11 11 11
4
        if len(tab) == 0:
5
             return m
6
7
        else:
             if tab[0] < m:
8
                 m = tab[0]
9
             return mini_rec2(tab[1:], m)
10
11
   mini_rec2(t, float("inf"))
12
```

Code 11 - Avec slice

Exercices récursivité Correction

- 1. Exercice
- 2. Exercice 2
- 3. Exercice 3
- 5 Evercice 5
- 6 F
- **-** .
- 8. Exercice 8

rcice 2

rcice 3

rcice 5

cice 7

Exercice 8

Exercice 9

cice 9

cice 10

31 / 40

$$a = 20, b = 35$$
 $35 = 20 \times 1 + 15$
 $20 = 15 \times 1 + 5$
 $15 = 5 \times 3 + 0$
 $20 = 5$

exercice

ercice 3

(ercice =

Exercice 5

Exercice 8

. . .

Code 12 – Version itérative

xercice 1

xercice :

ercice 3

......

....

_

Exercice 8

Exercice 9

Exercices récursivité Correction

```
def pgcd_rec(a: int, b: int) -> int:
    if a == 0:
        return b
else:
    return pgcd_rec(b % a, a)
```

Code 13 – Version récursive

xercice 1

kercice 2

ercice 3

CI CICC T

cercice 5

..........

kercice 7

Exercice 8

Exercices récursivité Correction

- 1. Exercice
 - ice 2
- 3. Exercice 3
- 4 Evereice
- 5. Exercice 5
- 6 Eversies
- 7 Eversice
- 8. Exercice

rcice 2

rcice 3

rcice 5

ercice 7

Exercice 9

ercice 9

ercice 10

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

Code 14

Exercice 1

xercice 2

ercice 3

CI CICC T

......

Exercice 9

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

Code 15 - Version terminale

xercice 1

xercice 2

kercice 3

xercice 4

kercice 5

xercice 6

.....

Exercice 9

Exercices récursivité Correction

- 1. Exercice
- 2. Exercice 2
- 3. Exercice 3
- 4 Evereice
- .
- 8. Exercice

rcice 2

rcice 4

cice 5

cice 7

rcice 8

Exercice 9

ice 10

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

xercice :

ercice 3

CI CICC T

....

kercice /

xercice 9

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

xercice 2

kercice 3

CI CICC +

tercice 5

xercice 0

exercice 9