#### Exercices récursivité Correction

Exercices récursivité
Correction

Christophe Viroulaud

Terminale - NSI

Lang 06

exercice 1

xercice 2

xercice 3

er cice 3

tercice 1

#### Exercices récursivité Correction

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

- -i-- 2
- cice 3
- rcice 4
  - ice 6
  - ice 7
- .....
- ercice 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

kercice 3

. . . . . . . . .

vorcico 6

\_\_\_\_

xercice 8

exercice 9

```
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.

```
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

kercice 3

Exercice

Exercice 5

xercice 0

kercice /

xercice 9

#### Exercices récursivité Correction

- 1. Exercice 1
  - 2. Exercice 2
- 3 Exercice 3
- 4 Evereine
- 5 Exercice 5
- 6 Exercice (
- 7 Exercice
- 8. Exercice

Exercice 2

orcico 3

· ·

ice 5

ice 7

rcice 9

rcice 9

rcice 10

xercice d

......

xercice 8

Exercice 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

kercice 3

xercice 5

\_\_\_\_\_\_

....

. . .

ACI CICC 5

Exercices récursivité Correction

3. Exercice 3

Exercice

Exercice 3

xercice 5

varcica 8

Exercice 9

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

Exercice 2

#### Exercice 3

ercice 4

vercice 5

xercice 6

xercice 7

ercice 8

exercice 9

#### Exercices récursivité Correction

- 4. Exercice 4

```
def syracuse(u: int) -> None:
1
       print(u, end=" ")
2
       if u > 1:
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 :

kercice 3

Exercice 4

kercice 5

orcico O

kercice 9

```
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.

#### Exercices récursivité Correction

- 5. Exercice 5

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

xercice :

cercice 3

#### Exercice 5

kercice o

ercice 7

ercice o

Exercice 9

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

xercice :

ercice 3

cercice 4

Exercice 5

xercice o

cercice /

#### Exercices récursivité Correction

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

cice i

cice 3

rcice 4

Exercice 6

rcice 8

rcice 9

rcico 10

ice 10

1 2

3

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

Exercice

vercice 3

ercice 4

xercice 5

Exercice 6

cercice 7

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

ercice 3

CI CICC T

kercice 5

Exercice 6

......

```
def somme_rec(tab: list, deb: int, s: 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
        s (int): somme
    11 11 11
    if deb == len(tab):
        return s
   else:
        return somme_rec(tab, deb+1, s+tab[deb])
somme_rec(t, 0, 0)
```

1

2

3 4 5

7

q

10 11

12

13 14

15

Code 8 - Version récursive

```
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 9 – Autre version récursive avec slice

### Remarque

Le *slice* est hors programme.

rcice 1

. .

Exercice 3

xercice 4

xercice 5

\_XCICICC O

. . . . . .

.....

#### Exercices récursivité Correction

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

ercice 1

rcice 2

rcice 3

ercice 5

Exercice 6

Exercice 7

ercice 8

ercice 9

rcice 10

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

- Exercice 1
- xercice 2
- kercice 3
- xercice 4
- Kei Cice 3
- Xercice 0
- Exercice 7
  - Exercice 8
  - Exercice 9
- Exercice 10

kercice 2

ercice 3

kercice 4

xercice 5

kercice 6

Exercice 7

kercice 9

```
def mini rec(tab: list, deb: int, m: int) -> int:
    11 11 11
    cherche le plus petit élément du tableau
    Args:
        tab (list): le tableau
        deb (int): indice de l'élément en cours
        m (int): l'élément mini
    11 11 11
    if deb == len(tab):
        return m
    else:
        if tab[deb] < m:</pre>
             m = tab[deb]
        return mini_rec(tab, deb+1, m)
mini rec(t, 0, float("inf"))
```

1

2

4 5

6

8

9

10 11

12

13

14

15 16

17

Exercice 1
Exercice 2
Exercice 3
Exercice 4
Exercice 5
Exercice 6
Exercice 7
Exercice 8
Exercice 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 10 - Avec slice

- re
- 1. Exercice 1
- 0 Europias 0
- 2 Evereine 1
- \_\_\_\_
- E Evente E
- \_\_\_\_
- 7 :
- 8. Exercice 8

rcice 2

Exercices

récursivité

cice 3

ice 5

Exercice 7

rcice 9

rcice 9

cice 10

30 / 39

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

exercice 1

Exercice

ercice 3

-----

xercice 5

Code 11 – 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 12 - Version récursive

kercice 1

xercice 2

ercice 3

......

tercice 5

.........

Exercice 8

......

Exercices récursivité Correction

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

Code 13

Exercice 1

xercice 2

ercice 3

. . . . .

. .

.....

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 14 – Version terminale

#### kercice 1

xercice 2

cercice 3

kercice 4

xercice 5

ACICICC O

. .

Exercice 9

Exercices récursivité Correction

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

- \_ \_ .
- 8. Exercice

- reice 1
- cice 3
- rcice 4
- rcice 6
- rcice 2
- rcice 9
- rcice 9

```
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

xercice 5

ercice /

ercice o

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
```

Exercice 1

xercice 2

ercice 3

CI CICC T

tercice 5

xercice b

ercice /

xercice 9