

Étape III – Programmation dynamique

Cours 10 - [Récurtivité](#)

Cours 11 – Examen 2 PL (en classe)

Cours 12 - [Algorithme Glouton](#)

Cours 13 - [Programmation Dynamique](#)

Cours 14 - [Programmation Dynamique](#)

Cours 15 – Examen final

Étape III – Notes de cours – NSI



Auteur : David Roche

Récurtivité -

https://dav74.github.io/site_nsi_term/c4c/

NSI terminale

Partie 1 >

Partie 2 >

C4 : récursivité >

Cours >

Activités >

Exercices >

Révision >

C5 : listes-piles-files >

C6 : les dictionnaires >

C7 : les arbres >

C8 : algo arbres binaires >

*C9 : les graphes >

*C10 : algo graphes >

Partie 3 >

Partie 4 >

Partie 5 >

Partie 6 >

Projets >

Cours

Considérons le programme suivant :

```
def fctA():
    print ("Début fonction fctA")
    i=0
    while i<5:
        print(f"fctA {i}")
        i = i + 1
    print ("Fin fonction fctA")

def fctB():
    print ("Début fonction fctB")
    i=0
    while i<5:
        if i==3:
            fctA()
        print("Retour à la fonction fctB")
        print(f"fctB {i}")
        i = i + 1
    print ("Fin fonction fctB")

fctB()
```

1) Pile d'exécution

```
1  def fcta():
2      i = 0
3      while i < 5:
4          print(f"fctA {i}")
5          i = i + 1
```

```
8  def fctb():
9      i = 0
10     while i < 5:
11         if i == 3:
12             fcta()
13         print(f"fctB {i}")
14         i = i + 1
```

- Lecture des notes de cours de la section 1 Pile d'exécution P 2-4

Pile d'exécution dans le débogueur

```
1  def fcta():  
2      i = 0  
3      while i < 5:  
4          print(f"fctA {i}")  
5          i = i + 1
```


```
8  def fctb():  
9      i = 0  
10     while i < 5:  
11         if i == 3:  
12             fcta()  
13         print(f"fctB {i}")  
14         i = i + 1
```


- Que se passe-t-il à la ligne 12 ?
- Combien de variable `i` existe en mémoire lorsque l'on exécute la ligne 5, et 13 ?

L12; appel de fcta() et empilement de fctb()

```
1 def fcta():  
2     i = 0  
3     while i < 5:  
4         print(f"fcta {i}")  
5         i = i + 1
```


```
8 def fctb():  
9     i = 0  
10    while i < 5:  
11        if i == 3:  
12            fcta()  
13        print(f"fctB {i}")  
14        i = i + 1
```

Debug:  pile_execution_v2 x

Debugger 

Frames

- MainThread
- fcta, pile_execution_v2.py:2
- fctb, pile_execution_v2.py:12
- <module>, pile_execution_v2.py:17

Variables  Console

Connected to pydev debugger (build 211.7142.13)

fctB 0
fctB 1
fctB 2
>>>

L5; 2 variables i distinctes sont en mémoire

```
1 def fcta():
2     i = 0    i: 0
3     while i < 5:
4         print(f"fcta {i}")
5         i = i + 1
```

```
8 def fctb():
9     i = 0
10    while i < 5:
11        if i == 3:
12            fcta()
13        print(f"fctB {i}")
14        i = i + 1
```

Frames	Variables	Console
MainThread		
fcta, pile_execution_v2.py:5	fctB 0 fctB 1 fctB 2 fcta 0	>>>
fctb, pile_execution_v2.py:12		
<module>, pile_execution_v2.py:17		

Frames	Variables	Console
MainThread		
fcta, pile_execution_v2.py:5	01 i = {int} 0	

Frames	Variables	Console
MainThread		
fcta, pile_execution_v2.py:5		
fctb, pile_execution_v2.py:12	01 i = {int} 3	
<module>, pile_execution_v2.py:17		

L13; 1 seule variable i est en mémoire

```
1 def fcta():  
2     i = 0  
3     while i < 5:  
4         print(f"fctA {i}")  
5         i = i + 1
```

```
8 def fctb():  
9     i = 0    i: 3  
10    while i < 5:  
11        if i == 3:  
12            fcta()  
13        print(f"fctB {i}")  
14        i = i + 1
```

Frames	Variables	Console
MainThread		
fctb, pile_execution_v2.py:14	fctB 1 fctB 2 fcta 0 fcta 1 fcta 2 fcta 3 fcta 4 fctB 3	
<module>, pile_execution_v2.py:17		

Frames	Variables	Console
MainThread		
fctb, pile_execution_v2.py:14	01 i = {int} 3	
<module>, pile_execution_v2.py:17		

Pile d'exécution

Une pile d'exécution permet d'enregistrer des informations sur les fonctions en cours d'exécution dans un programme.

2) Fonction récursive

- Lecture des notes de cours de la section 2 « Fonction récursive »

```
1  def fcta():  
2      print("Hello")  
3      fcta()  
4  
5  
6  fcta()  
7  |
```

Fonction récursive

Une fonction peut s'appeler elle-même, on parle alors de fonction récursive.

Fonction récursive

Une fonction récursive doit disposer d'une condition d'arrêt sinon Python interrompt le programme en générant une erreur quand la pile d'exécution dépasse une certaine taille.

```
RecursionError: maximum recursion depth exceeded while calling a Python object
```

Fonction réursive

```
1 def fonct(n):  
2     if n>0:  
3         fonct(n-1)  
4     print(n)  
5  
6 fonct(3)  
7
```

- Combien y-a-t-il d'appel(s) de fonction?
- Avec quelle valeur de n la pile contient toutes ces fonctions?
- Quelle est la condition d'arrêt qui permet le dépilement des fonctions?

4 fonctions sont empilées lorsque $n = 0$

The screenshot displays the state of a Python IDE during a recursive call. The **Frames** pane on the left shows the call stack, with the top frame being `fonct, fonction_n.py:2`. Below it are four more frames, all labeled `fonct, fonction_n.py:3`, indicating that the function has been called multiple times. The bottom frame is `<module>, fonction_n.py:6`. The **Variables** pane on the right shows the current frame's variables, with `n = {int} 0`. A smaller, overlapping window shows the same IDE interface but with the `n` variable set to `3`, representing a previous state of the function call.

Frames

- MainThread
- fonct, fonction_n.py:2
- fonct, fonction_n.py:3
- fonct, fonction_n.py:3
- fonct, fonction_n.py:3
- fonct, fonction_n.py:3
- <module>, fonction_n.py:6

Variables

01 `n = {int} 0`

Variables

01 `n = {int} 3`

Lorsque $n = 0$, on dépile après le print()

```
1  def fonct(n):  n: 1
2      if n>0:
3          fonct(n-1)
4      print(n)
```

Frames	Variables	Console
MainThread		Files\JetBrains\PyCharm
fonct, fonction_n.py:3		--multiproc --qt-supp
fonct, fonction_n.py:3		C:/Users/lv027851/Deskt
fonct, fonction_n.py:3		Connected to pydev debug
<module>, fonction_n.py:6		0
		>>>

Exercice – Corriger la fonction fcta()

```
1 def fcta():  
2     print("Hello")  
3     fcta()  
4  
5  
6 fcta()  
7 |
```

```
1 + def fcta(n): ...  
5  
6  
7 fcta(3)
```

Fcta(n) doit afficher récursivement n

fois « hello »

Reproduisez ce résultat

1) Hello

2) Hello

3) Hello

Fonction récursive – fact(n)

```
1 def fact(n):  
2     if n > 0:  
3         return n * fact(n - 1)  
4     else:  
5         return 1  
6  
7  
8 print(fact(3))
```


3) Récurrence mathématique

- En mathématiques une suite définie par récurrence est une suite définie par son premier terme et par une relation de récurrence, qui définit chaque terme à partir du précédent (ou des précédents lorsqu'ils existent)
- Lecture des notes de cours de la section 3

Relation de récurrence fact(n)

- Condition d'arrêt;

$$u_0 = 1$$

- Récurrence;

$$n \text{ entier et } n > 0, \quad u_n = n * u_{n-1}$$

- $u_0 = 1$

$$u_1 = 1 * u_0 = 1 * 1 = 1$$

$$u_2 = 2 * u_1 = 2 * 1 = 2$$

$$u_3 = 3 * u_2 = 3 * 2 = 6$$

Exercice – Coder la suite de Fibonacci

- Condition d'arrêt;

$$u_0 = 0 \text{ et } u_1 = 1$$

- Récurrence;

$$n \text{ entier et } n > 1, \quad u_n = u_{n-1} + u_{n-2}$$

- $u_0 = 0$

$$\text{fib}(0) = 0$$

$$u_1 = 1$$

$$\text{fib}(1) = 1$$

$$u_2 = u_1 + u_0 = 0 + 1 = 1$$

$$\text{fib}(2) = 1$$

$$u_3 = u_2 + u_1 = 1 + 1 = 2$$

$$\text{fib}(3) = 2$$

$$u_4 = u_3 + u_2 = 2 + 1 = 3$$

$$\text{fib}(4) = 3$$