



## NF04 – P22

Rapport de Mini Projet : Le Compte Est Bon

Teddy Roncin

## Table des matières

<b>Quelques termes de vocabulaire .....</b>	<b>2</b>
<i>Hardcoder .....</i>	<i>2</i>
<i>Softcoder .....</i>	<i>3</i>
<i>camelCase.....</i>	<i>3</i>
<i>snake_case .....</i>	<i>3</i>
<b>Modifications par rapport au dernier rapport.....</b>	<b>3</b>
<b>Projet Python .....</b>	<b>3</b>
<i>Description.....</i>	<i>3</i>
<i>Exemples.....</i>	<i>3</i>
<b>Projet C .....</b>	<b>4</b>
<i>Description.....</i>	<i>4</i>
<i>Exemples.....</i>	<i>4</i>

## Quelques termes de vocabulaire

**Hardcoder** : Anglicisme. Signifie que le programme n'est pas modifiable facilement. On utilise souvent ce mot pour décrire une constante : on dit qu'une valeur est *hardcodée* quand elle n'est pas stockée dans une variable. Ce type de valeur est donc difficile à changer dans un gros programme. Ici, je parlerai des combinaisons, mais de la même manière, il n'est pas simple de rajouter un quatrième nombre dans le programme avec l'algorithme que j'ai utilisé en C.

**Softcoder** : Anglicisme. C'est le contraire de *hardcoder*, c'est-à-dire que le programme est simple à modifier. Ici, un programme *softcodé* sera donc un programme qui calcule les combinaisons pendant l'exécution.

**camelCase** : C'est une façon d'écrire le nom d'une variable. Il en existe beaucoup. Quand on utilise le *camelCase*, on fait commencer le premier mot par une minuscule, puis les autres mots commencent par une majuscule. Les mots ne sont pas séparés les uns des autres.

**snake\_case** : C'est une autre façon d'écrire le nom d'une variable. Quand on utilise le *snake\_case*, on fait commencer tous les mots par une minuscule. Les mots sont séparés par un underscore (`_`).

## Modifications par rapport au dernier rapport

Je n'ai rien modifié au niveau du fonctionnement des algorithmes depuis le dernier rapport. J'ai seulement modifié la forme de mes programmes Python et C, mais pas le fonctionnement général.

## Projet Python

### Description

Pour le projet Python, j'ai décidé d'utiliser un algorithme qui me permettrait de ne pas *hardcoder* toutes les solutions, mais qui en même temps serait assez efficace : il ne teste pas beaucoup de solutions plusieurs fois. L'algorithme n'a pas vraiment changé depuis mon dernier rapport, j'ai juste modifié le nom de quelques variables. J'avais écrit toutes mes variables en *camelCase*, mais en général, en Python, par convention, on utilise plutôt du *snake\_case*. De plus, je me suis rendu compte que le nom d'une de mes variables n'avait pas de sens : `combin`. J'avais dû la créer pour tester, en n'étant pas sûr que j'étais en train de

sauvegarder la combinaison de la bonne façon. J'avais donc mis un nom rapide à écrire, et que je pouvais comprendre assez facilement. Je l'ai donc renommée en `combination_string`. J'ai aussi enlevé un `print` qui affichait toutes les combinaisons essayées. Ce `print` était là pour le debug. J'ai aussi corrigé une faute d'orthographe dans un commentaire, et enfin ajouté des parenthèses autour des calculs pour l'affichage.

### Exemples

Voici des exemples de fonctionnement du programme :

<pre>Entrez la première valeur : 1 Entrez la deuxième valeur : 2 Entrez la troisième valeur : 3 0 [(1 + 2) - 3 = 0] 1 [1 = 1] 2 [2 = 2] 3 [3 = 3] 4 [1 + 3 = 4] 5 [3 + 2 = 5] 6 [3 * 2 = 6] 7 [(3 * 2) + 1 = 7] 8 [(1 + 3) * 2 = 8] 9 [(1 + 2) * 3 = 9]  Process finished with exit code 0</pre>	<pre>Entrez la première valeur : 2 Entrez la deuxième valeur : 1 Entrez la troisième valeur : 3 0 [(2 + 1) - 3 = 0] 1 [1 = 1] 2 [2 = 2] 3 [3 = 3] 4 [3 + 1 = 4] 5 [2 + 3 = 5] 6 [2 * 3 = 6] 7 [(2 * 3) + 1 = 7] 8 [(3 + 1) * 2 = 8] 9 [(2 + 1) * 3 = 9]  Process finished with exit code 0</pre>
--	--

Les résultats obtenus sont les mêmes, mais les opérations pour y arriver sont différentes. On peut voir que les résultats sont triés dans l'ordre croissant, et que donc la liste obtenue est exactement la même. Dans le programme en C, les résultats seront aussi les mêmes, mais l'ordre sera différent, puisque je ne fais pas de tri dans l'algorithme que j'utilise en C.

## Projet C

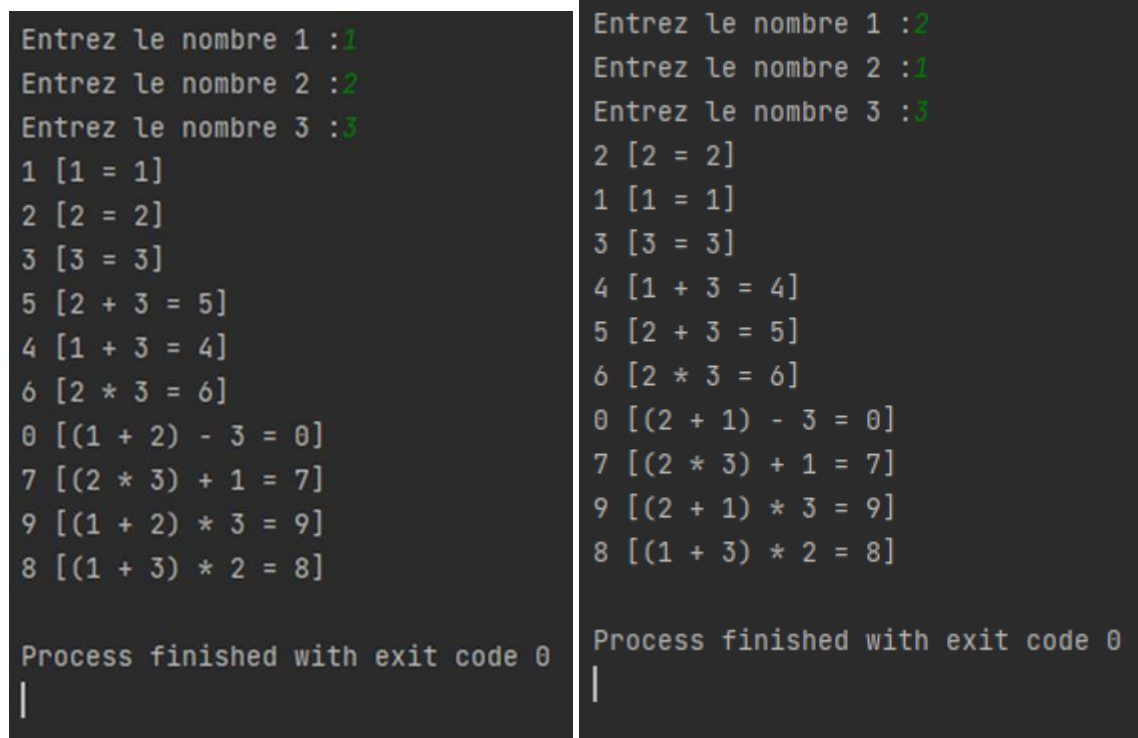
### Description

Pour le projet C, j'ai utilisé un fichier qui contient toutes les combinaisons possibles. C'est donc la manière la plus efficace de résoudre ce problème, puisqu'un algorithme ne peut pas vraiment deviner seul et rapidement si deux calculs sont équivalents, par exemple si  $a + b = b + a$ . Et même s'il pouvait, on pourrait par exemple avoir 2 calculs qui semblent similaire :  $a \div b \times c$  et  $a \times c \div b$ . Mais avec les contraintes que nous nous sommes donnés, le premier calcul peut être impossible, alors que le deuxième est réalisable. Par exemple, prenons les valeurs  $a = 1, b = 2, c = 4$ . Le premier calcul n'est pas possible,

car 1 n'est pas un multiple de 2. Cependant, le deuxième calcul est possible, car  $1 \times 4 = 4$  est un multiple de 2. C'est pour ce type de cas que je pense qu'il n'existe probablement pas d'algorithme *softcodé* pouvant être aussi efficace qu'un algorithme *hardcodé*.

### Exemples

Voici quelques captures d'écran du fonctionnement du programme :



```
Entrez le nombre 1 :1
Entrez le nombre 2 :2
Entrez le nombre 3 :3
1 [1 = 1]
2 [2 = 2]
3 [3 = 3]
5 [2 + 3 = 5]
4 [1 + 3 = 4]
6 [2 * 3 = 6]
0 [(1 + 2) - 3 = 0]
7 [(2 * 3) + 1 = 7]
9 [(1 + 2) * 3 = 9]
8 [(1 + 3) * 2 = 8]

Process finished with exit code 0
|

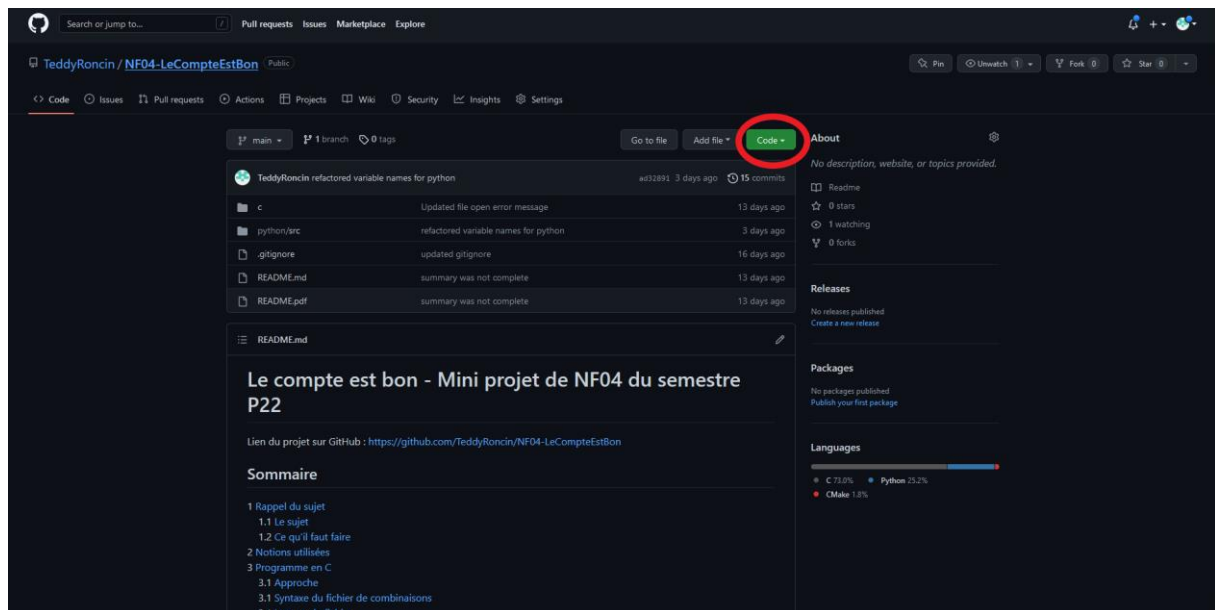
Entrez le nombre 1 :2
Entrez le nombre 2 :1
Entrez le nombre 3 :3
2 [2 = 2]
1 [1 = 1]
3 [3 = 3]
4 [1 + 3 = 4]
5 [2 + 3 = 5]
6 [2 * 3 = 6]
0 [(2 + 1) - 3 = 0]
7 [(2 * 3) + 1 = 7]
9 [(2 + 1) * 3 = 9]
8 [(1 + 3) * 2 = 8]

Process finished with exit code 0
|
```

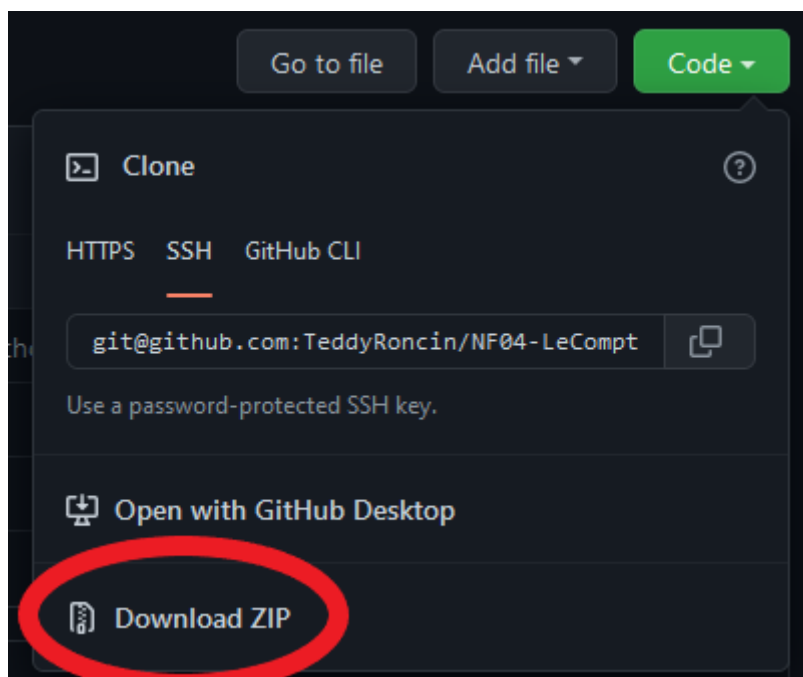
On peut remarquer que les résultats obtenus sont les mêmes, mais dans un ordre différent, ce qui est bien un résultat attendu : calculer les combinaisons de calculs possibles pour 1, 2 et 3 revient à calculer les celles possibles pour 2, 1 et 3. L'ordre et les calculs faits pour arriver à chaque valeur changent car on est obligé de définir une priorité pour les calculs. Ici on suit l'ordre du fichier.

## Programmes

Les codes sources des programmes sont disponibles sur la page GitHub de mon Mini-Projet. Vous pourrez aussi y retrouver le fichier expliquant en profondeur le principe des algorithmes que j'utilise. Voici le lien vers le projet : <https://github.com/TeddyRoncin/NF04-LeCompteEstBon>. Pour télécharger le projet, vous pouvez cliquer sur le bouton « Code »



Puis cliquez sur « Download ZIP » :



Vous trouverez ensuite le fichier `.zip` dans votre dossier de téléchargements.