**Projet n°1 Python**

Date de remise : 07 novembre 2019

**LE NOMBRE DE SCHUR S(n)**



Projet réalisé par Raoul Lisa, Soussan Chloé (Année L1 classique)

**SOMMAIRE**

**Introduction**

1. **Détail des solutions**
2. Programmes choisis transcris en langage algorithmique et choix de structures de données
3. Pourquoi avoir choisi ces programmes ?
4. Programmes en langage algorithmique
5. Réponses aux questions notées QT dans l’énoncé
6. **Nos remarques personnelles**
7. Points positifs du projet
8. Bénéfices des programmes pour l’utilisateur final
9. Bénéfices pour nous
10. Difficultés rencontrées

**Conclusion**

**INTRODUCTION**

Pour notre premier semestre de programmation nous avons dû réaliser ce projet qui porte sur le Nombre de Schur.

\* Qu’est-ce que le nombre de Schur ?

Le nombre de Schur est une suite de nombres entiers qui permet de réaliser différentes actions comme :

* colorier N entiers en utilisant une palette de n couleurs tout en respectant certaines contraintes (qu’on verra par la suite)
* déterminer si les triplets sont monochromatiques ou polychromatiques
* calculer N entiers pour tout n>=2 couleurs

\* Quelles sont les contraintes à respecter ?

Ces contraintes sont les suivantes :

* déterminer jusqu’à quel entier naturel N peut-on colorier avec n couleurs choisies. Celles-ci doivent être présentes au moins une fois. Cette condition doit être respectée en évitant qu’il y ait des triplets de la forme (a,b,a+b) monochromatiques.
* trouver une coloration utilisant les n couleurs souhaitées pour laquelle aucun triplet n’est monochromatique (c’est-à-dire que les chiffres qui sont dans le triplet ne sont pas de la même couleur)

Pour appréhender le résultat de notre projet, nous présentons en première partie les programmes qui nous semblent les plus importants. Puis, en deuxième partie nous formulons nos remarques personnelles relatives aux points positifs de notre projet et aux difficultés rencontrées.

1. **Détail des solutions** 
   1. Programmes choisis transcris en langage algorithmique et choix des structures de données

Nous avons décidé de présenter les programmes en langage algorithmique qui nous semblent les plus importants.

1. Pourquoi avoir choisi ces programmes ?

Il nous a paru plus judicieux de présenter le développement des programmes suivants : **[exo 3.3.1], [exo 3.3.4], [exo 3.3.7], [exo 3.3.9]** car ils font partis du cœur du programme final.

Pour présenter chacun de ces programmes, nous avons choisi, d’expliquer dans un premier temps les méthodes employées puis dans un second temps de transcrire ces programmes en langage algorithmique.

1. Programmes en langage algorithmique

* Le premier programme permet de savoir si les nombres saisis par l’utilisateur formant les triplets sous forme (a ,b,a+b) sont monochromatiques ou pas. **[exo 3.3.1]**

Pour construire ce programme nous avons d’une part utilisé les **méthodes** suivantes :

\* **« Random.shuffle() »**: permet de mélanger les couleurs

En effet, « random.shuffle(my\_color) » permet de mélanger les couleurs dans ma liste my\_color

C’est pour cela que nous avons importé la bibliothèque « random »

\* **« alea() »** : permet de choisir quelque chose

En effet « alea(L) » permet de choisir une couleur du tableau L

\* la condition « **si L[i] = B alors :** » vérifie que la couleur choisie est la même que la première couleur de la liste my\_color

D’autre part, nous avons employé la **variable** suivante **« verification »**. L’incrémentation de 1 permet de déterminer si les 3 nombres saisis sont de la même couleur : c’est pour cette raison que nous avons mis comme condition « si verification = 3 ».

Ce premier programme a le **langage algorithmique** suivant :

Programme : Monochromatique ou pas **[exo 3.3.1]**

Variables locales : my\_color, tableau 1D à 6 valeurs, contient toutes les couleurs de départ

x,y,z, entiers, représentent les valeurs du triplet

L, tableau 1D, contiendra nouvelles couleurs du tableau my\_color mélangé

A, entier, représente nbCouleurs saisit par l’utilisateur

verification, entier, permettra de déterminer si les nombres saisis sont de mêmes couleurs

B, chaine de caractères, représente le premier élément de la liste L

i, entier, compteur de boucle

DEBUT:

W <- "\033[0m"

R<- "\033[31m"

G<- "\s033[32m"

O<-"\033[33m"

B<-"\033[34m"

P<-"\033[35m"

my\_color <- [R,B,W,G,R,P]

afficher "saisir une valeur: "

x<- saisie

afficher "saisir une deuxième valeur: "

y<- saisie

afficher "saisir une troisième valeur: "

z<- saisie

L<-[]

A<- 3

random.shuffle(my\_color) \\ nous n’avons pas trouvé d’équivalent pour la méthode random.shuffle en algorithme

pour i allant de 0 à 3 faire:

L <- L + my\_color[i] \\ représente en python L.append(my\_color[i])

fin pour

pour j allant de 1 à A faire :

alea(L) \\ alea() est l’équivalent de random.choice(L)

fin pour

B<-L[0]

i<-1

verification<-1

tant que i < taille(L) faire:

si L[i] = B alors :

verification <- verification + 1

i<- i+1

sinon :

i <- i+1

fin tant que

si verification = 3:

afficher " Ce triplet ( "

afficher L[0] + x

afficher ","

afficher L[1] + y

afficher ","

afficher L[2] + z

afficher R

afficher ") est monochromatique"

sinon:

afficher L[0] + x

afficher ","

afficher L[1] + y

afficher ","

afficher L[2] + z

afficher R

afficher " ) n’est pas monochromatique "

fin si

FIN

* Le deuxième programme permet de déterminer à partir d’un certain nombre de couleurs donné et d’entiers N, une coloration aléatoire. Si celle-ci inclut les n couleurs souhaitées alors affichage d’un message de validité sinon affichage d’un message de non validité. **[exo 3.3.4]**

Pour construire ce programme nous avons utilisé les **méthodes** suivantes :

\* **« Random.shuffle() »**: permet de mélanger les couleurs

C’est pour cela que nous avons importé la bibliothèque « random »

\* **« alea() »** : permet de choisir quelque chose

En effet « alea(colors) » permet de choisir une couleur du tableau colors

\* la condition « **si i = len(liste\_couleur2)-1 alors:** » permet de vérifier si la liste a été parcouru entièrement

\* **« ‘\n’ »**: permet d'aller à la ligne

Ce deuxième programme a le **langage algorithmique** suivant :

Programme : Coloration nombres **[exo 3.3.4]**

Variables locales : my\_color, tableau 1D à 6 valeurs, représente les couleurs disponibles

N, entier, représente le nombre d’entier souhaité

nbCouleurs, représente le nombre de couleurs souhaité

i, entier, compteur de boucle

colors, tableau 1D, contiendra couleurs en fonction du nombre de couleurs voulu

liste\_couleur2, tableau 1D, contiendra les couleurs apparues

DEBUT:

W <- "\033[0m"

R<- "\033[31m"

G<- "\033[32m"

O<-"\033[33m"

B<-"\033[34m"

P<-"\033[35m"

my\_color <- [W, R, P,O, G, B]

afficher "Veuillez saisir le nombre d'entier que vous souhaitez: "

N<- saisie

afficher "Veuillez saisir le nombre de couleurs que vous souhaitez: "

nbCouleurs<- saisie

colors <- []

random.shuffle(my\_color) \\ nous n’avons pas trouvé d’équivalent pour la méthode random.shuffle en algorithme

pour i allant de 0 à nbCouleurs faire :

colors <- colors + my\_color[i]

fin pour

liste\_couleur2 <- [""]

pour k allant de 1 à N+1 faire:

c<- alea(colors)

i <- 0

tant que i < taille(liste\_couleur2) faire:

si c != liste\_couleur2[i] alors:

si i = taille(liste\_couleur2)-1 alors:

si liste\_couleur2[0] = "" alors:

liste\_couleur2[0] <- c

i <- i+1

sinon:

liste\_couleurs2 <- liste\_couleur2[i] + c

i <- i+1

sinon:

i <- i+1

sinon:

i <- taille(liste\_couleur2)

afficher c + k

afficher '\033[0m' \\ représente la couleur choisie, blanc

fin pour

si nbCouleurs = taille(liste\_couleur2) alors:

afficher " ’\n’ Le coloriage des nombres est valide."

sinon:

afficher " ’\n’ Le coloriage des nombres n'est pas valide. "

fin pour

FIN

* Le troisième programme explique comment convertir un nombre décimal en un nombre binaire. **[exo 3.3.7]**

Pour construire ce programme nous avons utilisé d’une part, la **méthode** suivante :

\***« str( ) »**: renvoi une chaine de caractère

D’autre part, nous avons employé la **variable** suivante car :

**\* « resultat »** : cette variable contiendra le nombre binaire du nombre décimal

Ce troisième programme a le **langage algorithmique** suivant :

Programme : Nombre décimal **[exo 3.3.7]**

Variables locales : nombre, entier, représente le nombre décimal entré

liste, tableau 1D, contiendra les restes d’une opération euclidienne

resultat, chaine de caractère, contiendra le nombre binaire

i, entier, compteur de boucle

nb\_voulu, entier, contiendra le nombre entré

nb\_choisi, entier, représente la valeur entière du quotient de la division

quotient, entier, représente le quotient de la division

DEBUT :

afficher "Veuillez entrer un nombre décimal: "

nombre <- saisie

liste<- []

nb\_voulu <- nombre

tant que nombre <= 0 faire:

afficher "Votre nombre doit être positif ! "

afficher "Veuillez saisir un nouveau nombre décimal: "

nombre <- saisie

fin tant que

tant que nombre != 0 faire:

nb\_choisi <- nombre

quotient <- nombre / 2

nombre <- floor(quotient)

fin tant que

si (nb\_choisi modulo 2) != 0 alors : \\ en python modulo représente %

reste <- " " + str(nb\_choisi modulo 2)

liste <- liste + reste

sinon :

liste <- liste + "0"

fin si

afficher(liste)

resultat <- ""

i <- taille(liste) - 1

tant que i >= 0 faire:

resultat <- resultat + liste[i]

i <- i-1

fin tant que

afficher "Le nombre souhaité " + nb\_voulu + " a pour valeur binaire " + resultat

FIN

* Le quatrième programme permet de déterminer quels sont les triplets monochromatiques à partir de N nombres entiers entrés par l’utilisateur avec n couleurs choisies aléatoirement. **[exo 3.3.9]**

Pour construire ce programme nous avons utilisé d’une part, les **méthodes** suivantes :

\* **« Random.randrange »** : permet de choisir un nombre aléatoirement

En effet, randrange(0,n) permet de choisir un nombre entier aléatoire de 1 à n-1

\* **« Random.shuffle() »**: permet de mélanger les couleurs

\* **« floor() »**: permet de récupérer la valeur entière de notre nombre

Ainsi comme ces méthodes font appel à la bibliothèque random, alors nous devons l’importer dès le début du programme python.

Par ailleurs, on a utilisé ces méthodes pour que les couleurs ne soient pas identiques (méthode shuffle()) et qu’elles soient mélangées à plusieurs reprises (méthode random.randrange().

D’autre part, nous avons employé **la variable** suivante **« Verification »**. Cette variable détermine si les couleurs de chaque élément sont égales

Cette variable permet de déterminer s’il y a, ou non, au moins un triplet monochromatique. La condition « verification = 1» détecte un triplet monochromatique si tel est le cas alors affichage d’un message "Faux" sinon affichage message "Vrai" si aucun triplet monochromatique est détecté.

Ce quatrième programme a le **langage algorithmique** suivant :

Programme: Triplets **[exo 3.3.9]**

Variables locales : my\_color, tableau 1D à 6 valeurs, représente les couleurs disponibles

N, entier, représente nombre d’entier souhaité

nbCouleurs, entier, représente le nbCouleurs souhaité

colors, tableau 1D ,stockera les nbCouleurs voulu

i, entier, compteur de boucle

l, tableau 1D, stockera les colorations à partir de n couleurs pour N premiers entiers

verification, entier, détermine les triplets monochromatiques

DEBUT:

W <- "\033[0m"

R<- "\033[31m"

G<- "\033[32m"

O<-"\033[33m"

B<-"\033[34m"

P<-"\033[35m"

my\_color <- [W, R, P, O, B, G]

afficher "Veuillez saisir le nombre d'entier que vous souhaitez: "

N <- saisie

afficher "Veuillez saisir le nombre de couleurs que vous souhaitez: "

nbCouleurs <- saisie

colors <- []

l<- []

random.shuffle(my\_color)

verification <- 0

pour i allant de 0 à nbCouleurs faire:

colors <- colors + my\_color[i]

fin pour

pour a allant de 1 à floor(N/2) + 1 faire:

random.shuffle(colors) \\ nous n’avons pas trouvé d’équivalent pour la méthode random.shuffle en algorithme

couleur\_a <- colors[random.randrange(0,nbCouleurs)]

couleur\_b <- colors[random.randrange(0,nbCouleurs)]

couleur\_somme <- colors[random.randrange(0,nbCouleurs)]

pour b allant de 0 à N faire:

si (a+b)<= N et a>= floor(N/2) alors:

si b >= a alors:

afficher "("

afficher couleur\_a + '' ''+ a

afficher ","

afficher couleur\_b + '' ''+ b

afficher ","

afficher couleur\_somme+ (a+b)

afficher ")"

si couleur\_a = couleur\_b = couleur\_somme alors :

verification <- 1

fin si

sinon si ((a+b <= N) et (a!=0) et (b!=0)) alors:

afficher "("

afficher couleur\_a + '' ''+ a

afficher ","

afficher couleur\_b + '' ''+ b

afficher ","

afficher couleur\_somme+ (a+b)

afficher ")"

si couleur\_a = couleur\_b = couleur\_somme alors :

verification <- 1

fin si

si verification = 1:

afficher "False"

sinon:

afficher "True"

l <- l +(couleur\_a+couleur\_b+couleur\_somme)

afficher l

fin si

FIN

* 1. **Réponses aux questions notées QT dans l’énoncé**

En se basant sur les programmes réalisés, nous avons pu répondre aux questions demandées.

* **Question 3.3.5)**

D’après la règle énoncée par Schur qui exclut tout triplet monochromatique de la forme (a, b, a+b), on observe qu’avec seulement 2 couleurs il ne peut pas y avoir plus de 4 coloriages.

Par exemple :

Prenons n=2, on a : 1 2 , 1 2

Prenons n=4, on a : 1 2 3 4 , 1 2 3 4 , 1 2 3 4

Prenons n=5 : aucun coloriage n’est possible en respectant la consigne émise précédemment.

En effet, avec l’exemple de (1,1,2), pour éviter que le triplet ne soit monochromatique il faut que 2 soit bleu. On obtient ainsi, ce triplet (1,1,2).

Puis, prenons ce triplet (2,2,4) même idée que précédemment pour que ce triplet ne soit pas monochromatique, on doit colorier 4 en rouge : ainsi on obtient (2,2,4).

Cela amène à la conclusion suivante : au vu de ces deux exemples, on constate que le 5 peut être ni rouge ni bleu sinon on obtiendrait un triplet monochromatique.

* **Question 3.3.6)**

Avec un certain nombre de couleurs n, le nombre total de coloriages pour N>2 nombres entiers est nN.

Soit l’exemple suivant pour N=3 on trouve qu’il y a 2 couleurs par chiffre, ainsi on a 23 = 8 colorations possibles.

* **Question 3.4.2)**

« Selon le théorème de Schur, pour n donné, il existe un nombre S(n) défini ainsi : c’est le plus grand entier N pour lequel il est possible de colorier en n couleurs les entiers de 1 à N de façon à exclure tout triplet monochromatique de la forme (a,b,a+b) avec a,b,a+b compris entre 1 et N. »

Ainsi, S(2) vaut 4 car les triplets polychromatiques sous la forme ( a, b ,a+b) et respectant ce prédicat sont: (1,1,2 ) ; (1,2,3) ;(2,2,4) ;(1,3,4).

* **Question 3.4.4)**

D’après le théorème de Schur et le programme effectué dans la partie 3.4) Appliquer, on obtient les 4 premiers nombre de Schur suivants : S(2) = 4, **S(3)= 13**, S(4)=44.

Prenons l’exemple de S(3)=13. Grâce au programme 3.4.3&5, nous nous arrêtons à 13 pour 3 couleurs car à partir de 14, l’ensemble des colorations admettent au moins un triplet monochromatique.

Cependant, nous nous arrêtons à N= 4 car plus N est grand plus il y a un nombre de colorations importantes.

1. **Nos remarques personnelles**

Nous avons décidé de dédier cette partie à nos remarques personnelles liées aux difficultés rencontrées et aux points positifs de notre projet.

1. **Points positifs du projet**

* + 1. Bénéfices des programmes pour l’utilisateur final

Les bénéfices sont les suivants :

\* la vitesse d’exécution : en effet, les algorithmes programmés permettent de déterminer toutes les colorations possibles d’un nombre N d entiers souhaité en temps rapide (programme 4.1)) (soit l’expression S(n)= N plus le n est élevé plus le programme est long et a une conséquence sur la durée)

\* la facilité d’utilisation : un seul clic est nécessaire pour lancer chaque programme

\* un jeu ludique : cela se traduit par l’emploi aléatoire de couleurs pour suivre l’exécution des programmes

* + 1. Bénéfices pour nous

**Pour Lisa :**

Tout d’abord, j’ai pris plaisir à construire plusieurs programmes qui permettent :

* d’une part, de déterminer si le triplet obtenu est monochromatique ou non
* d’autre part, de trouver au moins une coloration avec les n couleurs souhaitées tout en respectant la règle suivante : le triplet obtenu ne doit pas être monochromatique.

De plus, ce projet m’a permis personnellement de revoir des notions appréhendées en terminale.

Cependant, au cours du projet j’ai rencontré deux difficultés dont une que j’ai pu résoudre comme nous le verrons dans la prochaine partie.

**Pour Chloé :**

Je ne savais pas coder avant de rentrer à l’Efrei, ce qui à travers ce projet m’a permis d’approfondir mes connaissances en python.

Par exemple :

- Mettre en place des triplets

- Colorier des triplets

- Vérifier s’ils sont monochromatiques ou pas dans un laps de temps à calculer

De plus, j’ai pu apprendre de mes erreurs en avançant sur le projet. Tout ceci grâce au travail que j’ai réalisé avec mon binôme qui était plus expérimentée que moi et qui m’a beaucoup aidée en étant à l’écoute de tous mes problèmes rencontrés.

1. **Difficultés rencontrées**

**Pour Lisa :**

D’une part, j’ai rencontré un problème d’affichage des virgules entre les triplets monochromatiques. Ce problème n’a pas été résolu.

D’autre part, concernant l’exo 3.4.1) je ne savais pas par où commencer après avoir traduit l’exercice en algorithme. Toutefois, ce problème a été résolu en utilisant deux boucles. La première est « tant que » qui permet de calculer la valeur binaire d’un nombre décimal pour la raison suivante : le nombre convertit en base de nbCouleurs doit être positif et inférieur au nb\_coloration possible pour nbCouleurs donné.

La seconde est « pour » qui associe une couleur à chaque élément du triplet sous forme (a ,b,a+b).

Et pour finir j’ai utilisé une condition « si » qui détermine si la coloration affichée contient ou pas un triplet monochromatique.

**Pour Chloé :**

Pour la rédaction du code j’ai eu des difficultés avec l’indentation et l’affichage des parenthèses pour les triplets et leurs couleurs (préparer le terrain : ex3.3.1)).

Ensuite, je n’avais pas également compris la fonctionnalité de la bibliothèque « random » qui m’a beaucoup freiné au début de chaque programme.

De plus j’ai eu du mal avec le choix de mes conditions «si » lorsque que je l’utilisais avec la condition « alors » car je ne réalisais pas que je mettais plusieurs conditions «si ». C’est pourquoi j’avais plusieurs erreurs de codes au début.

Cependant j’ai résolu ce problème avec l’emploi de la condition « sinon si ».

Pour la compréhension du code à réaliser, j’ai eu du mal à appliquer les codes que j’avais réalisé à travers l’exercice 4) car je ne comprenais pas les bornes de S(n).

**Conclusion**

Nous avons apprécié de travailler ensemble car nous nous sommes entraidées lorsque nous rencontrions un problème. En effet, nous avons pu échanger, partager nos doutes et interrogations, confronter nos points de vue pour mieux comprendre ou déceler de nouvelles pistes de travail. Le travail en équipe a donc une vraie valeur ajoutée car on est plus fort à plusieurs.

En outre, nous avons développé notre capacité d’adaptation et de persévérance pour trouver la réponse adaptée au problème et pouvoir avancer.

Par ailleurs, nous avons employé un raisonnement analytique pour écrire des algorithmes plus complexes comme l’exercice ex3\_2. En effet, nous avons réfléchi aux différentes étapes pour arriver au programme final : tout d’abord, nous avons analyser le problème posé, puis nous avons listé des idées et pour finir nous avons développé celles-ci.

Donc, ces atouts nous ont permis de réaliser ce projet dans les temps.

Sources :

<http://images.math.cnrs.fr/Les-nombres-de-Schur-des-centenaires-pleins-d-avenir.html>

<https://www.ccmm.ca/documents/retour_ecole/2006/cles06_emploi_fr.pdf>