PROJET 1 - PYTHON

LE NOMBRE DE SCHUR S(n)

Detaille des solutions (en langage algorithmiques) et explication des choix de structures de données

Théorème (Schur, 1916). 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.

Malo ARHIMBAULD

Pierrick DELRIEU

Table des matières

La résolution en langage algorithmique et détailler pour chaque programme dans un fichier annexe

QUELQUES EXERCICES D'ECHAUFFEMENT

Programme 1

Objectif : Écrire un programme qui permet de colorer les ***N*** premiers nombres entiers en deux couleurs, telle que les nombres pairs sont affichés en rouge, et les nombres impairs sont affichés en bleu.

CF ANNEXE (pour algorithme)

- *Utilisation de i%2=0 qui signifie que i divisé par 2 n'a pas de reste, soit que i est un nombre pair*

- *Utilisation de (end='') pour afficher le résultat sur la même ligne.*

- *Utilisation d'une boucle for car on connait la fin de la récurrence qui est N (la boucle for commence à 1 et fini à N (soit N-1 en langage python)*

Programme 2

Objectif : Écrire un programme qui permet de colorer les ***N*** premiers nombres entiers en ***n*** couleurs choisies aléatoirement (les ***n*** couleurs doivent être présentes).

CF ANNEXE (pour algorithme)

- *Nous avons rencontré une difficulté par rapport au choix des couleurs de manières aléatoires. Le programme pouvez choisir deux fois la même couleur. Pour cela, nous avons créé une liste (couleur) où la taille de la liste correspond au nombre de couleur. Nous avons ensuite associé à chaque nombre N un élément de la liste (soit une couleur).*

CF commentaire du programme

```
#Création d'une liste aléatoire c avec n couleurs différentes
c=[]
for i in range (1,n+1):
    a=random.choice(color) # choix d'une couleur aléatoire dans color
    while (a in c) : #tant que la couleur est une couleur deja chosie (cad appartenant a c), redemander la couleur
        a = random.choice(color)
    c.insert(i-1,a) #inserer la nouvelle couleur chosie aleatoirement dans color dans c


#Insertion de N valeur (couleur) dans la liste w parmi c
w=c #la liste w a deja n valeur
for cpt in range (n+1,N+1): #de n+1 à N pour que la liste w est N valeur (car elle est initialisé a n valeurs)
    b=random.choice(w)
    w.insert(cpt,b)
random.shuffle(w) #Melanger la liste w pour melanger les trois premiers couleurs de la liste


#Attribution de chaque valeur de la liste w à chaque Nombre
for j in range(1,N+1):#du nombre 1 à N
    print(w[j-1],j,end='') #j-1 car la liste w commence à l'indice 0
```

PREPARER LE TERRAIN

Programme 1

Objectif : Écrire un programme qui, à partir de trois entiers et leurs trois couleurs respectives, affiche VRAI si le triplet est monochromatique, FAUX sinon.

CF ANNEXE (pour algorithme)

- *Pour réaliser le programme, il fallait d'abord initialiser une liste comprenant les couleurs disponibles (noir, rouge, vert, orange, bleu, violet) ainsi qu'une liste comprenant les chiffres de 1 à 9.*
- *Une valeur aléatoire est prise dans la liste des couleurs pour définir la couleur du premier nombre et une autre dans la liste des nombres pour définir la valeur du nombre.*
- *Ensuite, on vérifie si les trois valeurs de couleur de chaque nombre est égal.*
- *Lors de l'affichage, il a fallu recolorer la fin du message et les virgules car celles-ci étaient coloriées en la couleur du nombre précédent.*
- *Lors de l'exécution du programme on observe une fréquence plus élevée de triples qui ne sont pas monochromatiques. Cependant les triplets monochromatiques apparaissent plus fréquemment.*

```
>>> %Run 'Programme 1.py'
  ce triplet (  4  ,  2  ,  6  ) est monochromatique
>>> %Run 'Programme 1.py'
  ce triplet (  8  ,  6  ,  14  ) n'est pas monochromatique
```

Programme 2

Objectif : Écrire un programme qui à partir d'un entier p ≤ 100, génère et affiche la liste de tous les triplets (a, b, a + b) avec a, b, a + b ≤ p

CF ANNEXE (pour algorithme)

- *D'abord, il faut demander à l'utilisateur une valeur de p qui doit être comprise entre 2 et 100 car a et b doivent être différents de 0 et que par conséquent, la valeur minimale de a + b et de 1. On sécurise donc la saisie*
- *On initialise a et b à 1, plus petite valeur possible pour les deux nombres. Tant que a + b est inférieure ou égale à p, on rajoute 1 à b si b est inférieur à p - a ou sinon, on remet b à 1 et on rajoute 1 à a.*
- *Pour n'avoir qu'une seule fois les triplets, on enlève ceux dans lesquels a est supérieur à b car on a d'abord augmenté la valeur de b.*

```
a = 1
b = 1
while (a+b<=p):
    if (a<=b ):         # comme b augmente avant a, on enlève les valeurs de a qui sont plus grandes que b
        print("(",a,",",b,",",a+b,")")
    if (b<p-a):         # permet d'augmenter la valeur de b en vérifiant que b ne seras pas trop grand
        b = b+1
    else:               # permet d'augmenter a de 1 et de remettre b à 1 quand b est trop grand
        b = 1
        a = a+1
```

Programme 3

Objectif : Écrire in programme qui utilise deux couleurs pour colorier N nombres entiers aléatoirement. Ce programme doit afficher VRAI si tous les triplets ne sont pas monochromatiques, FAUX sinon.

CF ANNEXE (pour algorithme)

- *Pour faire ce programme, il fallait d'abord demander à l'utilisateur de saisir un nombre, dernière valeur possible de la somme. Cette valeur est inférieure ou égale à 100 pour éviter que le programme dure trop longtemps. On doit associer les nombres à leurs couleurs, déterminés aléatoirement. Ensuite, il faut déterminer les triplets et les afficher. On enlève les triplets déjà apparus. Puis, on vérifie si au moins un des triplets est monochromatique.*

- *On a rencontré un problème au niveau de l'affichage de toutes les combinaisons possibles en couleurs. Pour cela, on a modifié les conditions pour que toutes les combinaisons s'affichent et qu'un compteur change quand on croise un triplet monochromatique.*
- *Nous avons réutilisé le programme précédent pour faire les triplets.*

Programme 4

Objectif : Écrire un programme qui à partir d'un nombre de couleurs ***n*** et d'un nombre d'entiers ***N***, génère une coloration aléatoire. Il affiche VRAI si la coloration générée inclut les ***n*** couleurs utilisées. FAUX sinon.

CF ANNEXE (pour algorithme)

- *Inversement a l'exercice 2 de la première partie, ici la liste peu choisir aléatoirement deux fois la même couleur (si c'est le cas le programme affichera faux).*
- *On fait donc un test pour voir si dans la liste que l'on a définie contenant normalement n couleurs il y a bien n couleurs différentes.*

```
#Test si il y a deux couleurs identiques dans c
test=0 #test=0 si les n couleurs sont presentes
for j in (W,R,G,O,B,P):
    x=c.count(j) #renvoi combien de fois il y a j dans c
    if(x>1):
        test=1 #test=1 si les n couleurs ne sont pas presentes
```

Utilisation de la fonction coun pour vérifier si un élément est contenu plusieurs fois dans la liste.

- *Ensuite comme dans l'exercice deux on modifie la liste pour quel contienne N éléments et puis on associe à chaque N une couleur.*

Question 5

Énoncé : Avec seulement 2 couleurs, quel est le nombre total de coloriages possibles pour colorier N nombres entiers ?

A chaque fois qu'on rajoute un nombre N, avec 2 couleurs, on double le nombre de combinaisons qu'on avait avant

Soit le nombre de combinaisons est 2×2×2x... N fois. Donc le nombre de combinaisons pour n=2 est **2^N.**

Question 6

Énoncé : Et si le nombre de couleurs est *n*, quel est le nombre total de coloriages possibles pour colorier N nombres entiers ?

On en déduit ainsi d'après le résultat obtenu précédemment si on généralise alors on a le nombre de combinaisons de colorations avec n couleurs pour N nombres est **n^N.**

Programme 7

Objectif : Écrire un programme qui convertit un nombre décimal en un nombre binaire (11 → 1011).

CF ANNEXE (pour algorithme)

Le projet ne prend pas en compte les nombres négatifs et les nombres décimales. Ainsi, le programme doit convertir seulement les entiers positifs en binaire.

Méthode : On divise par 2 autant de fois qu'il est nécessaire pour obtenir un quotient nul. Et on écrit ensuite les reste dans l'ordre inverse où ils ont été obtenus.

On divise ainsi une première fois puis procède par récurrence avec une boucle while.

Le seul problème rencontré est d'inverser l'ordre des restes obtenue. (Voir ci-dessous pour comprendre la solution).

Nous avons initialisé le rang à 0.

```
#Toutes les autres divisions
while (quotient!=0):
    reste=quotient%2
    quotient = quotient/2
    quotient = int(quotient)#On garde la partie entiere du quotient
    rang=rang-1 #On ajoute a chaque fois le reste obtenue dans la liste binaire vers la gauche gauche (soit au rang,rang-1)
    binaire.insert(rang,reste)
```

Programme 8

Objectif : Généraliser le programme précèdent afin de convertir un nombre décimal en base ***b*** (>1).

CF ANNEXE (pour algorithme)

- La base de conversion est strictement supérieure à 1 et peut aller jusqu'à 10 + 26 lettres de l'alphabet soit 36 (inclus)
- Idem programme précèdent, cependant il a fallu à chaque fois que le reste obtenu était supérieur ou égale à 10 le convertir en la lettre de l'alphabet qui correspond. Nous avons pour cela créé une liste alphabet (voir ci-dessous)

```
alphabet=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
```

```
#1ere division
reste = n%base
if (reste>=10):
    reste = alphabet[reste-10]
```

Ici, on pour reste=10 alors reste sera égal à la valeur de la liste d'indice 0 soit A ; Ainsi, de suite.

Programme 9

Objectif : Écrire un programme qui stocke la liste de toutes les colorations possibles à partir de ***n*** couleurs pour ***N*** nombres entiers et où les ***n*** couleurs y figurent. Le programme doit afficher la liste obtenue à la fin.

CF ANNEXE (pour algorithme)

- *Pour la création d'une liste aléatoire contenant les **n** couleurs choisies par l'utilisateur nous avons utilisé la même méthode que dans l'exercice 2 de la première partie.*
- *L'objectifs était ensuite de créer à partir de cette liste toutes les combinaisons possibles avec **N** valeurs dans la liste. Toutes les combinaisons possibles doivent être stocker dans une liste.*
- *Pour obtenir toutes les combinaisons de c, avec N valeurs, nous avons utilisés une boucle pour Pour et le programme précèdent (8) pour convertir les toutes les combinaisons soient tous les nombres allant de n**N*
- *On n'a pas besoin de la partie de l'alphabet dans ce programme car le reste ne sera jamais supérieur ou égal a 10 car on convertit seulement en base 6 max.*
- *Pour cela, nous avons créé une nouvelle liste qui a chaque fois stockera la combinaison de c.*
- *Cependant, nous avons rencontré un problème car seules les combinaisons contenant les n couleurs doivent être ajouter à la liste.*
- *Pour cela nous avons utilisé count qui nous renvoi le nombre de fois qu'un élément est dans une liste. Avec cela nous avons établie des conditions, soit si tous les éléments (soit les n couleurs) ne sont pas dans la liste alors ils ne sont pas ajoutés à la liste finale.*

Insertion de la liste seulement si la combinaison contient les n couleurs :

```
#Insertion dans une liste seulement les combinaisons qui contiennent les n couleurs
valide=0 #Initialisation de la variable valide (0 si cb valide et 1 sinon)
for i in range (0,n): #Pour n couleurs qui doivent apparaitre
    a=conv.count(i) #on regarde combien de fois la couleur est dans la liste conv
    if (a==0): #Si la couleur est 0 fois dans la liste conv
        valide=1 #alors la combinaisons n'est pas valide

if (valide==0): # si la combinaison est valide
    #Insertion de la combinaison dans une liste
    liste.append(conv) #utilisation de append pour inserer une liste dans une liste
    nb_cb=nb_cb+1
```

Pour la suite des programmes nous ne pourrons pas utilisées ce programme 9 identique mais nous utiliserons se programme 9 avec l'apparition de toutes les combinaisons.

Remarque : Ce programme nous a permis de comprendre comment faire toutes les combinaisons possibles.
Nous avons remplacé les couleurs pas des chiffres. Et ainsi, la conversion se fais en base n (=le nombre de couleur pour obtenir) n chiffres differents et pas de lettre.

Exemple : pour N=3 et n=2
Chaque liste doit donc contenir N=3 valeurs

[0,0,0] → conversion de 0 en base n=2
[0,0,1] → conversion de 1 en base n=2
[0,1,0] → conversion de 2 en base n=2
[0,1,1] → conversion de 3 en base n=2
…
[1,1,1] → conversion de ((n**N)-1)=7 en base n=2

Par exemple pour la deuxième combinaison : [0,0,1]
Posons 0 = la couleur bleue et 1 = la couleur rouge

On a 1 2 3.

APPLIQUER

Programme 1

Objectif : En se basant sur les programmes écrits précédemment, ainsi que le premier exemple illustratif, écrire un programme qui calcule ***S(2)***

CF ANNEXE (pour algorithme)

- *Nous avons d'abord posé la couleur bleu = 1 et rouge = 0.*
- *Nous avons ensuite combiné les programmes 9 et 2 de préparer le terrain.*
- *Avec une boucle pour et le programme 9 nous avons pu réaliser toutes les combinaisons de couleurs possibles contenant N nombres.*
- *Dans la boucle que nous pour, nous avons imbriqué le programme 2 pour faire tous les triplets possibles de cette combinaison. A chaque triplet on vérifie si le triplet est monochromatique ou pas. Puis on en déduit si N est valide.*
- *On réalise se programme tant que N est valide.*

Question 2

Objectif : Quel est le résultat final pour ***S (2)*** ? Justifiez.

- *Le résultat obtenu pour S (2) est 4.*
- *Cependant, lors de l'affichage nous devons afficher N-1 (soit 5-1=4) car le la boucle tant que n'est pas réitérer pour N=5 ce qui signifie que N=5 n'est pas valide. Il faut donc afficher N-1*
- *Le résultat obtenu est en accord avec l'exemple du sujet.*

Programme 3

Objectif : Généraliser le programme et calculer ***S(n) avec n > 2***.

CF ANNEXE (pour algorithme)

- *Pour généraliser le programme 1 Nous avons simplement remplacer la conversion d'un nombre décimal en binaire par la conversion d'un nombre décimal en base n (avec n qui correspond au nombre de couleur) afin d'obtenir des combinaisons avec n nombres différents soit n couleurs différentes*

Question 4

Objectif : Combien de nombres de Schur avez-vous réussi à calculer ? Expliquez le problème rencontré.

- *Nous avons réussi à calculer 3 nombres de Schur :*

S(2) = 4
S(3) = 13
S(4) = 44

Pour le calcul de S(4), le programme a tourné pendant longtemps.

Nous rencontrons un problème pour calculer un nombre de Schur avec un n (= nombre de couleur) supérieur ou égal à 5.

Cependant, nous avons pu observer sur internet que le nombre de Schur maximum calculable est S(5) = 160.

Ainsi, il n'est pas possible de calculer un nombre de Schur supérieur ou égal à 6.

```
Saisir n > 2 : 3

DONC S( 3 ) = 13
Temps d execution : 390.71398401260376 secondes ---
```

FONCTIONS AVANCÉES

Programme 1

Objectif : Écrire un programme qui permet d'afficher pour n'importe quelle valeur de ***n*** (***n >= 6)***, l'intervalle dans lequel se trouverait son résultat ***N***.

CF ANNEXE (pour algorithme)

- *Tout d'abord, nous avons défini la factorielle.*
- *Ensuite, nous avons simplement afficher le résultat comme demander dans l'énoncé.*

Programme 2

Objectif : Pour les plus motivés : Réécrire le programme ***S(n)*** en utilisant les fonctions.

CF ANNEXE (pour algorithme)

Tout d'abord, nous avons établie différentes fonctions :
- Saisie du nombre de couleur n
- Conversion d'un nombre n en une base B
- Détermination si la combinaison obtenue avec la conversion est valide ou non
- Détermination si le nombre N de nombres est valide en fonction des combinaisons valides
- Affichage du résultat
- Définition de la factorielle
- Si n ≥ 6 : affichage de l'intervalle de S(n)

Nous avons ensuite imbriqué ceci en remplaçant par le nom des fonctions dans le programme 3 d'Appliquer. Et nous avons rajouter le cas ou n ≥ 6.

Nous avons défini un fichier python contenant toutes les fonctions et un autre fichier situé dans le même dossier avec le programme où l'on a appelé le fichier contenant les fonctions avec from my_fct import * (* pour importer tout le fichier my_fct).

CONCLUSION :

Ce projet nous a permis de comprendre comment colorier des couleurs en python et également de savoir réaliser des combinaisons.

La mise en relation des projets de préparation pour obtenir le programme final a été quelques choses de difficile. Ceci nous a permis de travailler notre réflexion sur un sujet d'informatique.

Il nous a également permis d'approfondir notre entrainement en python. Avant ce projet, nous ne connaissions pas les fonctions. Grace à internet, nous avons pu obtenir différentes informations sur comment établir des fonctions et d'ailleurs réussi à réaliser le programme final avec des fonctions

Le nombre de Schur reste un théorème de nos jours compliquer à résoudre pour des entiers supérieurs ou égal à 6.

Le dernier nombre de Schur obtenue est S(5) il y a deux ans. Ceci montre la complexité et le temps nécessaire pour résoudre ce théorème pour tout entiers naturels n.