1 Détail sur mon programme

1.1 Approximate_pi

Le programme est divisé en deux parties:

Utilisation en programme principal (i.e. ./approximate_pi.py n):
Géneration de n points aléatoires et l'on compte le nombre de points dans le cercle unité, cependant les points ne sont pas sauvegardés afin d'avoir une performance mémoire optimale.

Utilisation en appel:
Le programme stocke les n points sous forme de tuple de la forme (x, y, Bool) dans une liste, le booléen exprime si le point est dans le cercle unité ou non. La liste permet de conserver l'ordre de génération en comparaison à l'usage d'un dictionnaire. J'ai fais le choix de ne pas utiliser de namedtuple point par soucis de simplicité, de même je n'ai pas créé de fonction auxiliaire afin de ne pas faire d'appel de fonction pour optimiser la compléxité temporelle du programme. La fonction retourne uniquement la liste de tuple de taille n le nombre de points demandé.

1.2 Draw

L'objet principal de mon programme est un dictionnaire (que l'on nommera A par la suite) contenant lui-même n dictionnaires ayant pour clés des entiers, qui eux même contiennent des booléens et des None ayant aussi pour clés des entiers, un couple d'entier représente evidemment les coordonnées d'un point. Utiliser des dictionnaires paraissait être le meilleur choix en terme de compléxité temporelle et spatiale. Il permet d'accéder rapidement au résultat à la lecture, écrire rapidement dessus quelque soit la coordonnées et cela pour une taille mémoire raisonnable comparé à un tableau dynamique ou encore une liste chainée.
Pour creer les 10 images tout en respectant l'ordre de génération des points, le programme, entre chaque impression, ajoute dans le dictionnaire A les $\frac{n}{10}$ points suivants de la liste généré par approximate_pi. Pour imprimer pi, draw creer une copie B de A à l'aide de la fonction *.copy*(), c'est ensuite sur ce dictionnaire B que l'on implente pi à l'aide d'un afficheur sept segments. L'afficheur est fait entièrement à la main, la taille des caratères, du point ou encore l'espace entre les caractères s'adapte à la taille de l'image. Les traits ont une épaisseur de 3 pixels pour que les chiffres soient visibles.
Il vient maintenant de creer les images au format ppm, j'ai fais le choix de travailler au format $P3$ avec comme valeur maximale 1. J'ai choisi ce format par soucis de simplicité, je n'arrivais pas à convertir les images en gif au format P6. Cependant le choix de 1 comme valeur maximale permait de minimiser la taille mémoire de mon image. Afin d'écrire l'image dans le document, j'évalue B en chaque coordonnée possible. Pour maximiser l'espace mémoire du programme B ne contient que 3 états : $None$, $True$ et $False$ correspondant respectivement à noir, rouge et bleu, si l'on a une erreur, on l'attrape et on imprime du blanc.
Aprés la création des 10 images il vient la crétaion du gif à l'aide du module

subprocess, j'ai utilisé la commande *run* puis *convert* de bash.

2 Pistes d'amélioration

De manière générale ce que je reproche à mon programme c'est de ne pas être assez modulable, d'être trop compliqué à modifier. En prenant du recul sur le programme, il est brouillon, la fonction main reflete mon chemin de penser instinctif qui n'est pas optimal.

2.1 approximate_pi

Pour l'utilisation en programme principale, je ne vois pas comment l'améliorer en terme de performance temporelle et spatiale. Cependant cela doit rester possible de l'améliorer au vu des performances des autres candidats.
Pour l'utilisation en appel, il aurait je suppose était mieux de faire de cette fonction un générateur pour gagner de la mémoire et ne pas stocker les points sous forme de liste de tuple. Cependant comme précisé ci-dessus, cela m'aurait prit trop de temps de tout refaire aprés la découverte de l'outil *yield*.

2.2 draw

Afin de minimiser la taille de mon image, il aurait été intéressant de modifier le format de mon image en P6 et d'écrire en bit. Cependant je n'y suis pas arrivé. De même pour diminuer la mémoire utilisé par le programme, il faudrait trouver une autre méthode pour imprimer pi sans copier A. J'ai essayé, sans modifier ou encore faire de copie de A, de creer un autre dictionnaire de dictionnaire contenant $None$ aux points convenables pour imprimer pi. Cependant cela ne s'est pas avéré plus efficace, je suis donc revenu à la solution initiale.

3 Ce que je retiens du projet

Lors de l'élaboration de ce programme j'ai perdu beaucoup de temps à cause de la non structure initiale des programmes. Pour le prochain projet, je prendrai du temps au début pour poser le problème sur feuille et mieux stucturer mon programme. Cela m'aurait permis d'optimiser mon programme en amont plutot que de me lancer spontanément sur la programmation. Il était très intéressant de découvrir comment les autres ont développés et comparer avec ce que l'on a fait, j'y ai beaucoup appris.