Message au travers d’une image

Ce programme a été conçu dans l’optique d’un projet de BAC en option ISN par Simon LÉONARD et Kylian BACHELET

1. Idée et explication du projet
2. Organisation et répartition des tâches
   1. Le dépôt GIT
   2. Trello
   3. Organisation personnelle
3. La gestion du programme
   1. La classe « Image »
   2. La classe « Message »
   3. Les fonctions de « utils.py »
4. Comment marche le programme
5. Les problèmes survenus
6. Améliorations possibles
7. Idée et explication du projet

Ce projet est né d’une longue phase de discussion avec mon coéquipier, mais on a réussi à se mettre d’accord sur ce projet.

Le but de ce projet est de construire une image, de sorte à ce que quand on rentre un texte, il en sort une image de type aléatoire ; et si on rentre une image, il en sort un texte.  
Pour ce faire, l’image a des spécificités particulières, comme par exemple, des valeurs permettant au programme de savoir quelle est la longueur du texte, ou comment il est `` caché ´´ dans l’image.

Le programme est organisé de sorte à ce que l’image et le message soient interprétés comme des objets. Cela propose une simplicité dans le cadre d’une modification du programme, on en parlera lors de la présentation des classes.

1. Organisation et répartition des tâches
   1. Le dépôt GIT

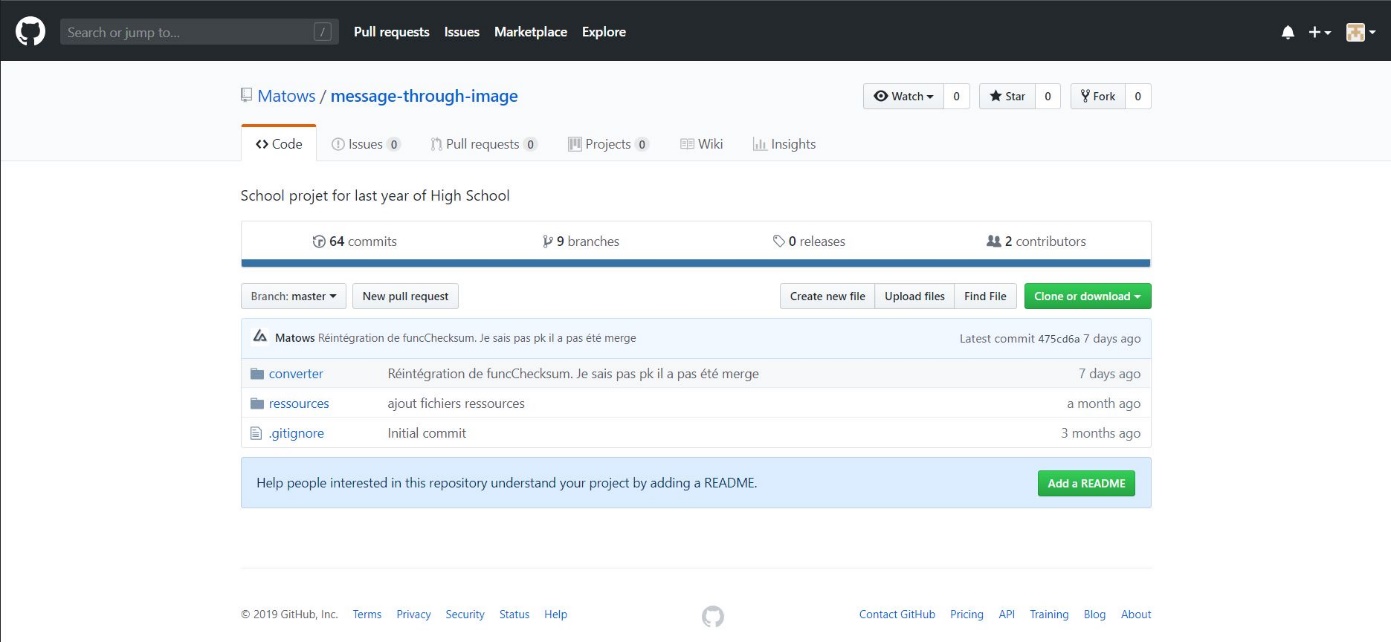
Dès lors qu’on a eu le projet en tête et qu’on a posé les bases, on a convenu de partager et modifier le programme via un dépôt GIT. 

Image du dépôt GIT

Ce dépôt nous a permis de pouvoir modifier le programme, et voir les modifications de l’autre, pour mettre en commun à la fin, et de temps à autre.

GIT est un site permettant de pouvoir collaborer à plusieurs sur un même programme, tout en créant des branches permettant de faire plusieurs fonction et les modifiées séparément, en tout cas, c’est comme ça qu’on a procédé.

Ici, on à toutes les branches qu’on a utilisées pour ce projet, tout à gauche, c’est la branche Master, celle où tout est réuni. La toute première, la noire, c’est pour la fonction de nettoyer le message. La deuxième, en vert clair, c’est pour vérifier les entrées. Celle en vert foncé et en bleu clair c’est pour le passage des pixels en nombre et inversement. Celle en rouge c’est pour crée et lire l’image. Celle en jaune c’était une fonction qu’on a remplacé plus tard, car elle ne fonctionnait pas.

* 1. Trello

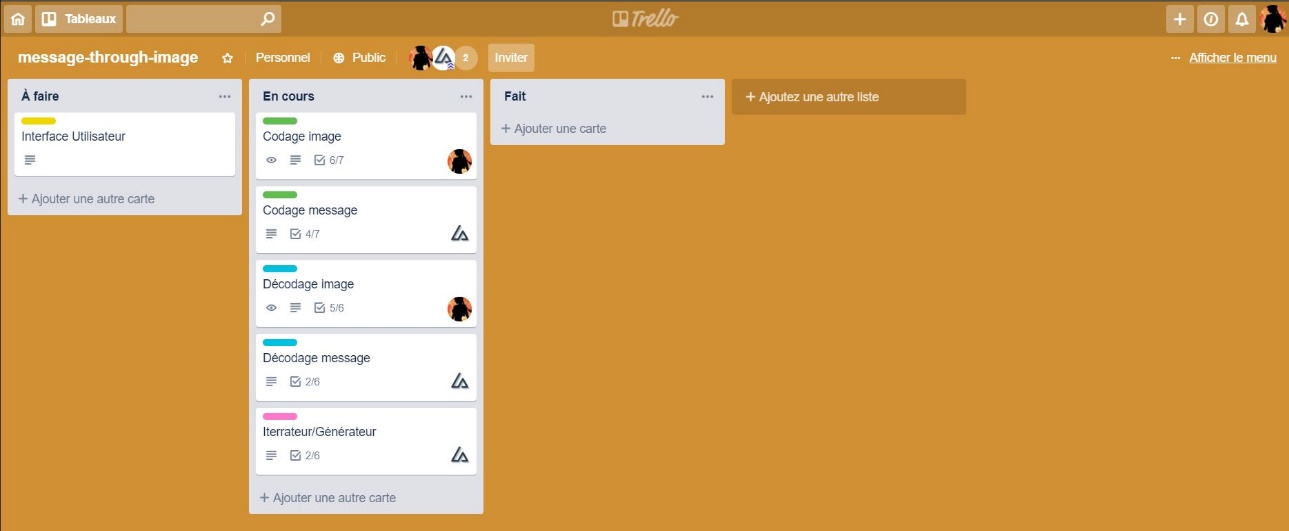
Après le dépôt GIT, on s’est aussi rapidement organisé autour de « Trello » un site permettant de faire des tableaux d’organisation.

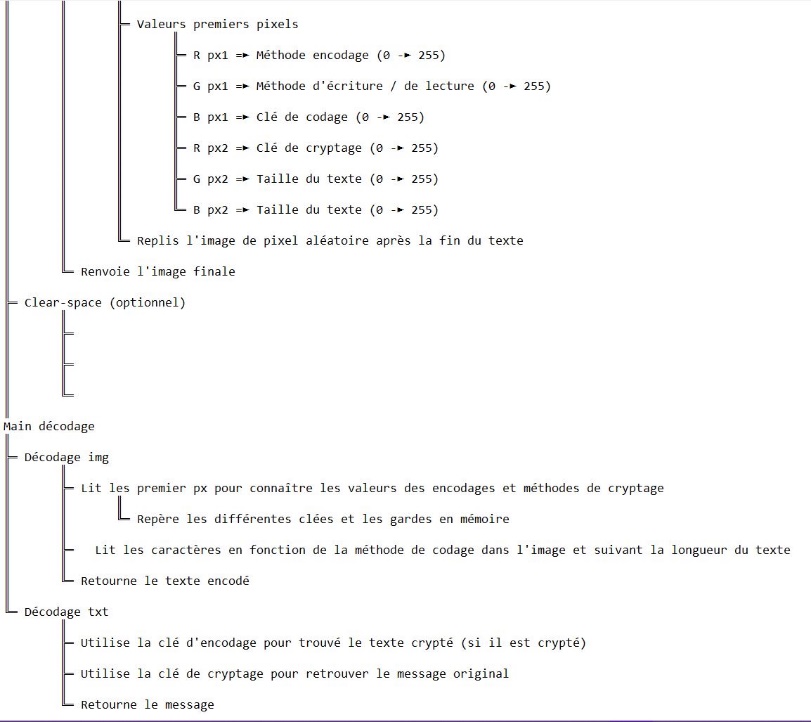
Image du Trello

On s’en est servi afin de définir qui faisait quoi et où on en était dans l’avancement général et particulier (c’est-à-dire dans les fonctions qu’on créait). Simon s’occupait de la partie message, et moi, de la partie image.

Les dernières choses à voir, c’est quelques fonction `` secondaires ´´ qu’on avait laissés de côté, plus le test final afin de voir si le programme fonctionne parfaitement.

* 1. Organisation personnelle

Afin de ne pas me perdre dans ce qu’il fallait faire, j’ai décidé de me crée trois documents, le premier basique avec que du texte m’as permis de voir ce qu’on allait mettre dans chaque fonction et mieux comprendre comment fonctionne le programme.

C:\Users\Asus\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Txt_int f() _ 3.jpg



Ce document a été légèrement modifié par la suite avec le deuxième document, qui est l’analyse fonctionnelle réalisé avec ClickChart, qui est un logiciel gratuit. Le troisième document est un organigramme qui a été réalisé avec LARP. LARP qui est originellement un logiciel de programmation pour Robot et appareils électroniques. [Tout est donné en annexe]

Lors de la conception du programme, je me suis penché sur la partie Image, mais Simon m’as bien aidé pour la fonction `` assemblage ´´ et `` des-assemble ´´ où il a fait la majeure partie du programme.

1. La gestion du programme

On a décidé de faire ce projet en `` orienté objet ´´, c’est-à-dire qu’on définit le message et l’image en tant qu’objet. Comme dit précédemment, la création des classe donne un avantage sur le point de vue de la modification du programme, on peut facilement modifier une partie, sans se soucier du reste du programme. On utilise trois classes dans ce projet.

* 1. La classe Message

La classe Message est celle qui va permettre de transformer le texte en donnée utilisable pour être transformer en image. Mais il peut aussi faite l’opération inverse, c’est-à-dire qu’il peut recueillir les données de l’image pour en ressortir un message.

Elle contient les fonctions servant à vérifier les caractères ASCII, à crypter le message, à faire passer les caractères en nombre, et les fonctions checksum.

* 1. La classe Image

La classe image, sert à crée et codée l’image selon les données que la classe Message lui donne. Il sert également à donner les données d’une image à cette même classe.

Elle contient les fonctions permettant de passer des nombre aux valeurs RGB, des valeurs RGB aux nombres, d’assembler l’image et de pouvoir la lire et extraire les valeurs RGB.

* 1. Le fichier Utils

Le fichier Utils est un fichier permettant de placer quelques foncions de vérification qui ne rentrent pas en compte ni dans Message, ni dans Image.

Il comprend les fonctions permettant de vérifier qu’on donne que le texte ou que l’image, la fonction permettant de renvoyer un « hash md5 », et celle du générateur qui dit comment lire une image. Il contient également un algorithme de cryptage simple qui sert d’exemple.

1. Comment marche le programme

Au départ, l’utilisateur rentre un message, il passe dans une fonction qui vérifie qu’il n’y a que des caractères de la table ASCII `` simple ´´ (du 0 au 84 ème [fournit en annexe]). Quand le message est correct, on demande si l’utilisateur veut le crypter, et après, il repasse dans une fonction où chaque caractère est transformé en nombre. C’est la fin de l’utilisation de la classe Message. Ensuite, c’est au tour de la classe Image. Il prend la suite de nombre que lui as donné la classe Message et la transforme en valeurs RGB. Une fois ces valeurs acquises, il les passes à une autre fonction qui va s’occuper de crée l’image. Et c’est terminé, l’utilisateur peut récupérer l’image.

Voici un exemple d’image crée :

Le pixel rouge correspond au pixel donnant la longueur du message. Les pixels cyan, sont les pixels correspondant au message. Les pixels violets sont ceux qui correspondent au checksum. Et ceux en vert sont aléatoires.



Bien sûr, ça n’aurait aucun intérêt si le programme ne pouvait pas faire l’inverse. Et c’est pourquoi, on a créé les fonctions réciproques de celle pour passer du message à l’image. Dans ce cas, l’utilisateur rentre l’image, le programme en ressort les valeurs RGB qu’il va ensuite convertir en nombre. Ensuite la classe Message prend ces nombre pour les transformer en caractères, puis les décrypter si besoin et enfin, donner le message contenu dans l’image.

1. Les problèmes survenus

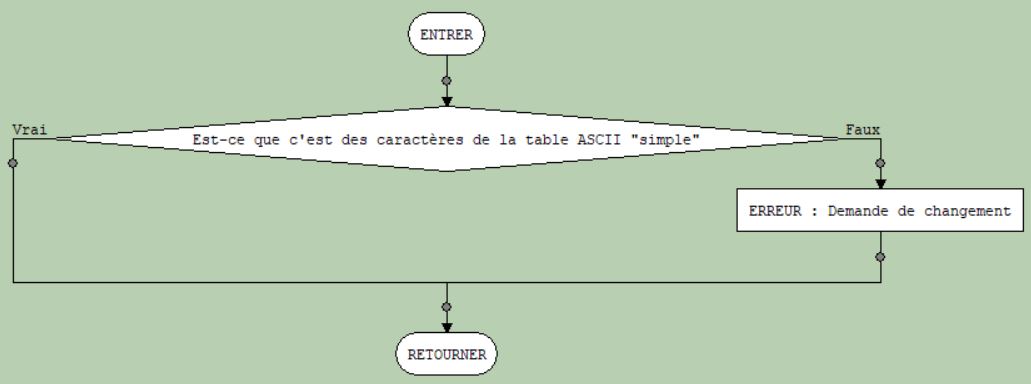
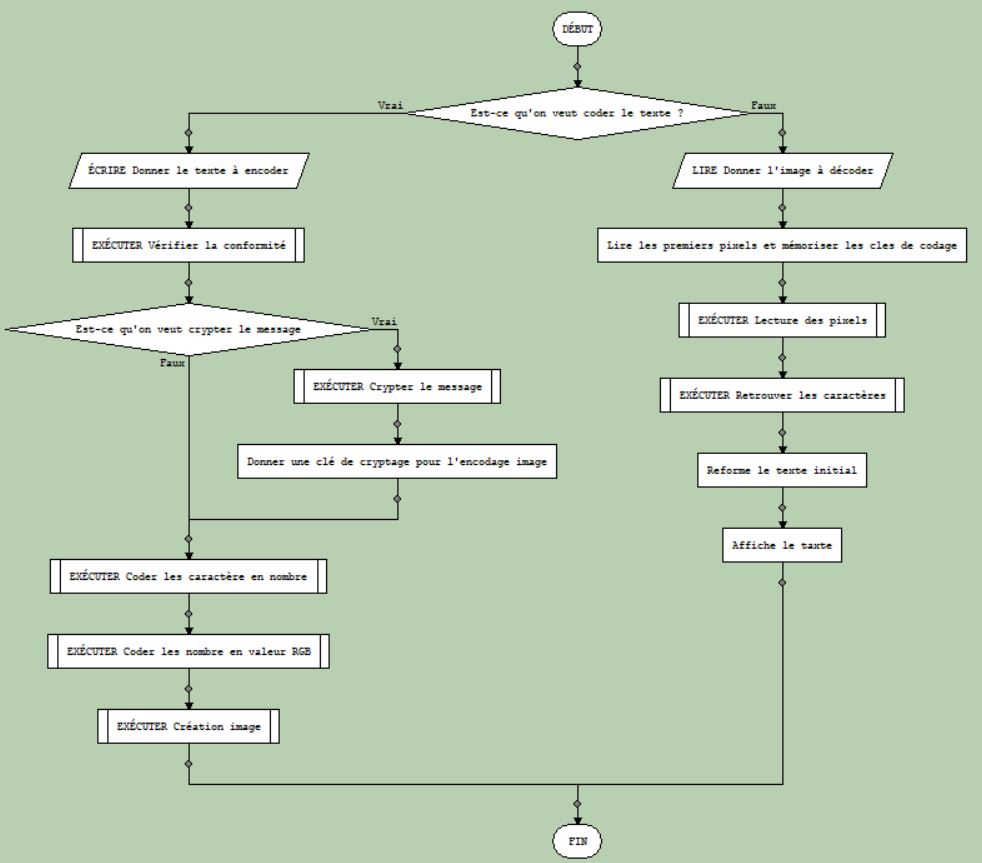
Comme tous projet, nous avons aussi eu notre lot de problème, et le plus important que j’ai eu, c’est que j’avais du mal à comprendre ce que devait faire le programme ou comment Simon voulait faire quelque chose, comme par exemple, pour l’encodage de l’image, on arrivait pas à se mettre d’accord sur la manière dont il fallait donner les informations.

J’ai également rencontré des problèmes avec GIT, quelques soucis de synchronisation avec les fichiers originaux, comme par exemple, le logiciel où je programme, `` Atom ´´, ne reconnaissait plus le dossier GIT, il a fallu supprimer le dossier, le recréé et recopier toutes les branches. Ce n’est pas grand-chose, mais ça fait perdre un peu de temps.

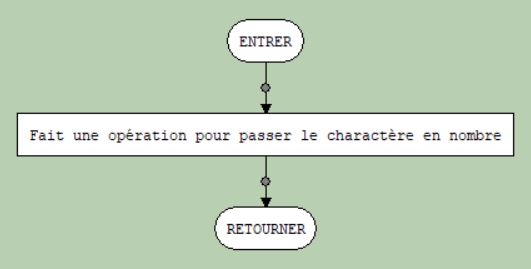
On a eu d’autres problème avec GIT notamment avec les branches où on a eu quelques soucis quand on voulait les incorporés dans la branche master.

ANNEXES

Fonctionnement du programme au complet.

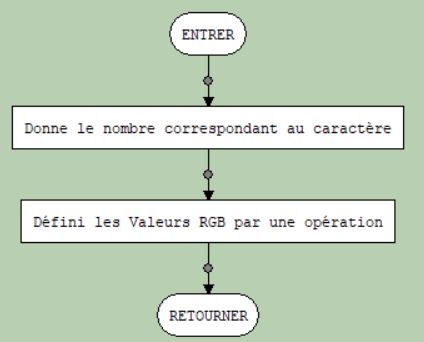


Celui-là sert à vérifier la conformité.

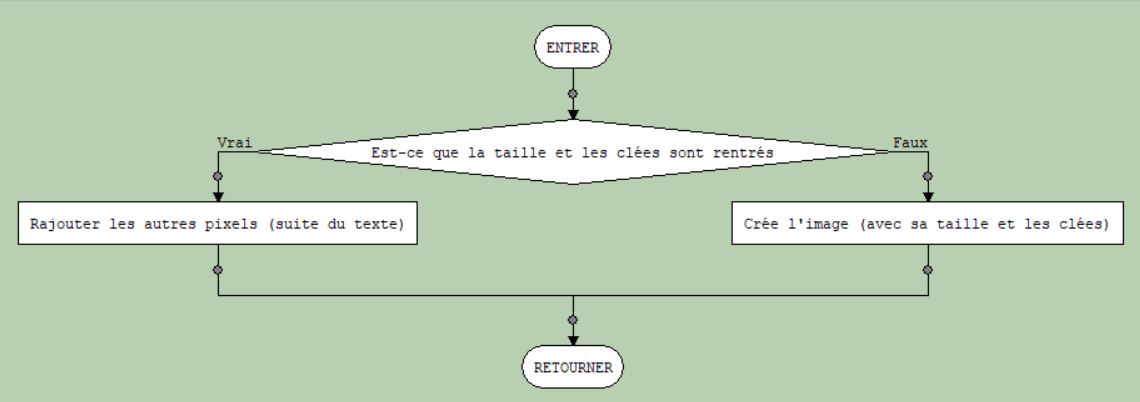


Celui-ci c’est pour passer des caractères en nombre.

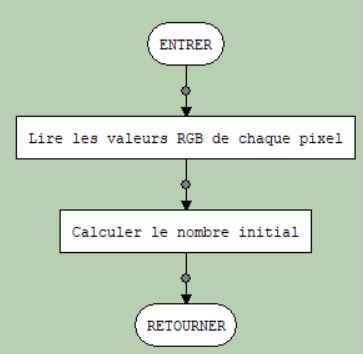
Là on passe des nombre en valeurs RGB.



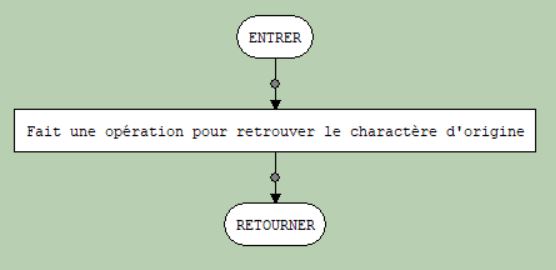
Ici c’est pour créer l’image.



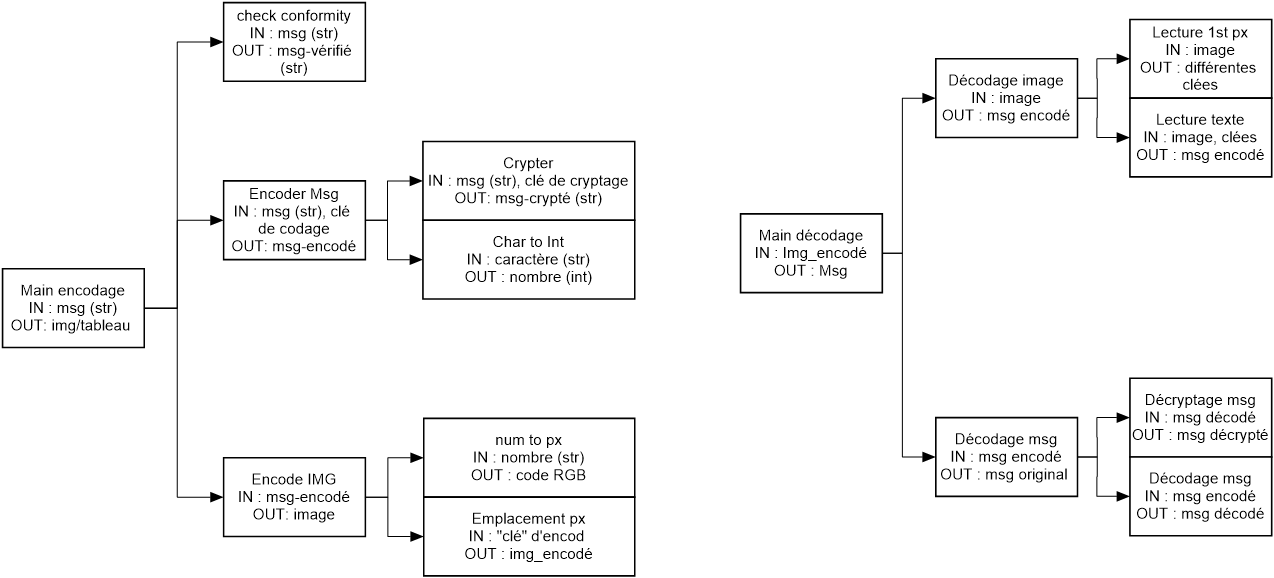
On lit les valeurs RGB.

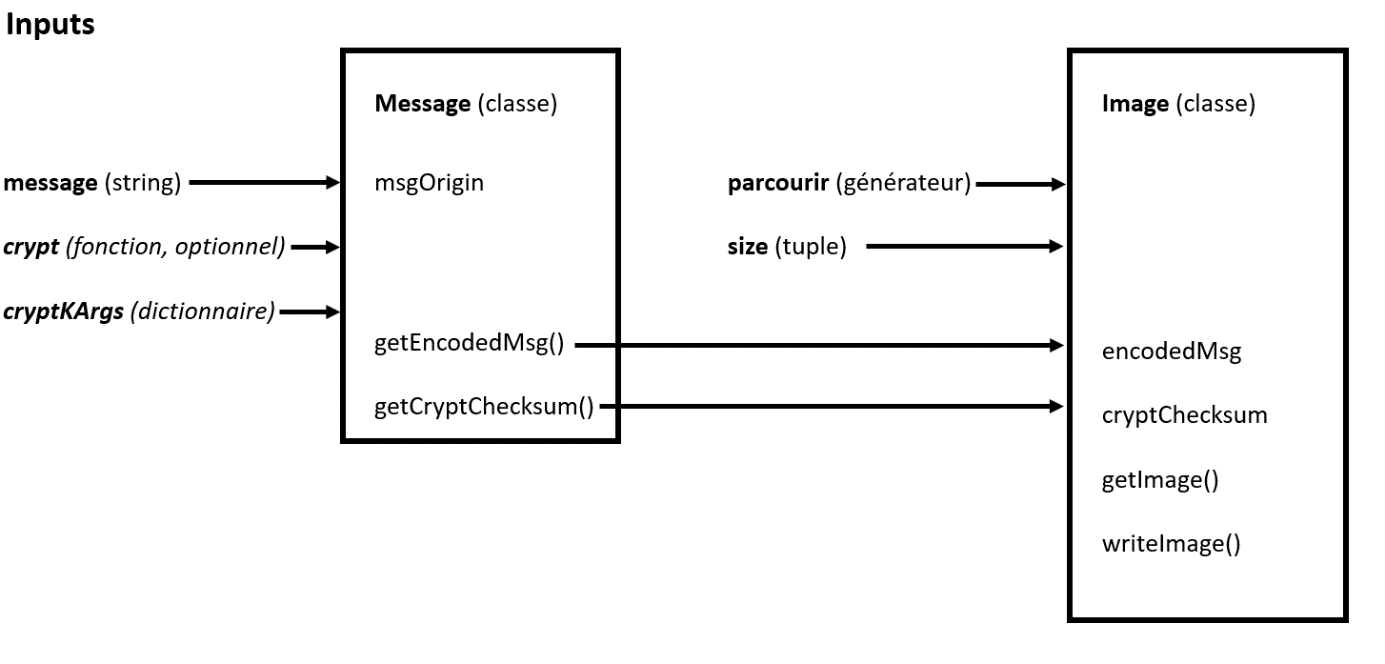


On retrouve les nombres.



L’analyse fonctionnelle :





**Inputs**

**Message** (classe)

imgOrigin

getListPx()

pxToNum()

**Image**

**Image** (classe)

numToChar()