1 Objectif

Il s'agit de réaliser une image montrant le contenu d'un fichier texte sur un fond flou de triangles aléatoires placés sur fond noir.

2 Modularité

Vous créerez trois fichiers :

1. `image_texte.py` contenant une classe de même nom.
   Le constructeur prend deux arguments :
   - un tuple `(largeur, hauteur)`
   - un chemin complet vers un fichier TrueTypeFont

   Cette classe possède quatre méthodes :
   - `traitement()`
   - `set_texte(chaine_fichier_texte)`
   - `get_image()`
   - `sauve(chaine_fichier_png)`
2. `image_triangles.py` contenant une classe de même nom.
   Le constructeur prend deux arguments :
   - un tuple `(largeur, hauteur)`
   - un nombre entier

   Cette classe possède quatre méthodes :
   - `un_triangle()`
   - `get_image()`
   - `sauve(chaine_fichier_png)`
3. `projet_code_image.py` qui utilise les classes précédentes pour réaliser l'objectif du projet à partir de la ligne de commande :
   `$ python3 projet_code_image.py 800 800 100 consignes.md`

Dans les classes (dans les modules), vous êtes libres du choix des attributs.

3 Éviter le découragement

Il va falloir être organisé et ne pas sauter les étapes. Il est possible que la conception d'un seul module mobilise le temps prévu pour ce projet. L'important est de faire bien ce qu'on fait, pas forcément de le finir. Si on ne fait que la partie triangles ou que la partie texte, c'est déjà très bien.

Il faudra lire la doc de `pillow` sur certains aspects qu'on n'a pas encore abordés. L'aide (voir pages suivantes) précise les fonctions importantes à utiliser, pour ne pas se perdre dans la documentation.

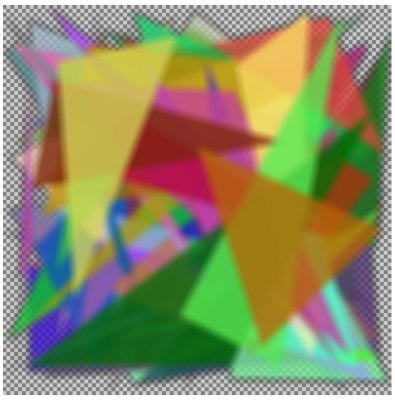
Sur simple demande par mail (académique) je vous envoie un code à compléter sur une partie ou toutes.

4 Fonctionnement attendu

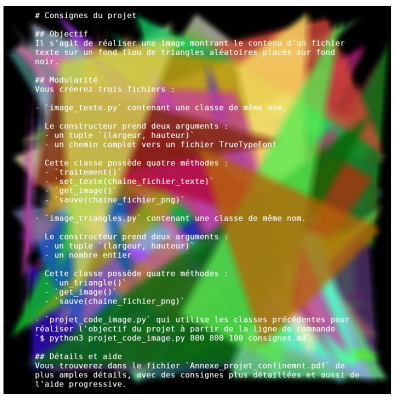
L'utilisateur choisit donc les dimensions d'une image (par exemple, un carré 800×800) et un texte (pas trop long ni trop large !! le script est fait pour les textes de taille raisonnable) et le programme crée une image aux dimensions demandées avec un fond de triangles et le texte demandé.

1. **Sur le fichier** `image_texte.py` :
   La classe crée un objet Image (PIL) à fond transparent. Dans cette image le texte (écrit en blanc) est centré et est complétement lisible.
   La méthode `sauve(chaine_fichier_png)` permet de débugger la classe, mais ne servira pas au fonctionnement global : c'est la méthode `get_image()` qu'il faut utiliser pour récupérer l'objet Image dans le script final `projet_code_image.py`.

2. **Sur le fichier** `image_triangles.py` :
   C'est presque la même interface que le fichier précédent avec les deux méthodes `get_image()` et `sauve(chaine_fichier_png)`. Le principe est donc le même, mais cette fois-ci on crée un nombre de triangles correspondant à l'argument du constructeur. Chaque triangle est créé avec la méthode `un_triangle()`. L'image est encore une fois sur fond transparent.



3. **Sur le fichier** `projet_code_image.py` :
   Il s'agit d'importer les modules précédemment créés, de gérer la ligne de commande et de sortir une seule image correspondant à l'objectif.



I Aide

1 Par quoi commencer ?

On développe d'abord les modules, on vérifie leur fonctionnement, on termine par le script « utilisateur ».
La conception de `image_triangles.py` est le plus simple. Ensuite on conçoit l'autre module `image_texte.py` On termine par le script `projet_code_image.py`.

2 Aide générale

Dans les deux modules, on insérera à la fin la ligne :
`if __name__ == "__main__":`
pour tester le module en utilisant une sortie fichier avec la méthode `sauve(chaine_fichier_png)`.
Pour le script final, on ne gèrera pas tout de suite les arguments en ligne de commande. Ce sera la dernière chose à faire dans le projet.

3 Aide sur les triangles

1. Dans la méthode `un_triangle()`, on crée une image transparente sur laquelle on place un triangle aléatoire et on fusionne cette image avec l'image courante qui est destinée à être renvoyée.
2. Pour gérer des placements aléatoires, on peut utiliser le fonction `randrange` de la bibliothèque `random`. Elle se prête parfaitement à cet usage.
3. L'image courante est un attribut. La méthode `get_image()` ne fait que renvoyer cet attribut.
4. Dans `un_triangle()`, on utilise les fonctions `Image.new`, `ImageDraw.Draw` et `Image.alpha_composite`.
5. Une fois les triangles positionnés, on peut flouter le tout avec la méthode `filter` du sous-module `ImageFilter`.
6. On travaille en `RGBA`.

4 Aide sur le texte

1. Dans le traitement, il faut trouver la bonne taille de police qui permettra d'insérer le texte complétement.

   Pour cela, on doit récupérer les dimensions d'un rectangle qui entoure le texte : voir la méthode `multiline_textbbox` sur un objet de type calque (context). Ces dimensions dépendent évidemment de la taille de police. On démarre avec une très grande taille de police.

   On boucle tant que l'ensemble ne rentre pas dans les dimensions de l'image, on décrémente progressivement la taille de police.
2. On peut utiliser `multiline_textbbox` en démarrant de $(0, 0)$, cela permet de récupérer directement la largeur et la hauteur du texte.
3. Une fois la bonne police trouvée, il suffit de centrer le texte dans l'image. Faire un petit schéma avec des dimensions permet de trouver le bon positionnement.
4. On doit lancer le traitement au dernier moment : dans `get_image()` et dans `sauve(chaine_fichier_png)` (pour la sortie débuggage).
5. La police d'écriture est importée avec `ImageFont.truetype`.
6. On travaille en `RGBA`.

5 Aide sur le script final

1. C'est dans ce script qu'on insére le fond noir, après avoir récupéré et fusionné avec `Image.alpha_composite` les deux objets Image créés avec les modules.

   Pour faire cela, on crée une image `RGB` de la même dimension et on va coller dessus notre image (qui a un fond transparent). On utilise la méthode `paste` avec l'option `mask` qui précise le canal à utiliser comme masque de transparence (pour nous c'est le 4e canal : le `A` de `RGBA`).
2. On utilise la bibliothèque `sys` pour gérer les arguments de la ligne de commande.
3. On peut nommer l'image de sortie en reprenant notamment le nom du fichier texte (sans son extension).