

Projet algorithmique et programmation - Anonymat photographique

Paul Gaborit

IFIA 1^{re} année – Septembre 2023 – IMT Mines Albi

1 Consignes

Ce travail s'effectue en binôme. L'archive ([.zip](#), [.rar](#), [.7z](#), [.tgz](#)...) à déposer sur Campus intégrera :

1. un document (en PDF) contenant :
 - a. le nom et le prénom de chacun des membres du binôme,
 - b. les schémas bulle, les algorithmes et la description du fonctionnement de chacune des fonctions que vous aurez conçues,
 - c. la description des images traitées pour faire vos tests,
 - d. toute autre information que vous jugerez utile pour valoriser votre travail,
2. les scripts Python (les fichiers [.py](#)) que vous aurez réalisés,
3. des exemples de résultats obtenus (avec les images d'origine, les fichiers d'ordres [json](#) et les images produites).

Les éléments qui **ne doivent pas être** dans cette archive sont : votre environnement virtuel (le répertoire [venv-projet](#)), le ou les répertoires [__pycache__](#).

2 Objectif

L'objectif de ce projet est de concevoir et d'écrire en Python un script permettant de cacher plusieurs parties d'une image (pour préserver l'anonymat de certains sujets par exemple).

Il y a trois phases permettant de produire des versions de plus en plus complètes de ce script : [anonymat-p1.py](#), [anonymat-p2.py](#), [anonymat-p3.py](#).

3 Le déroulement d'un script [anonymat.py](#)

L'exécution d'un script [anonymat.py](#) se déroule en plusieurs étapes :

1. Il analyse le contenu d'un fichier d'ordres (dont le nom est fourni sur la ligne de commande ou demandé à l'utilisateur) contenant les paramètres des opérations à réaliser : le

nom du fichier image à traiter, le nom du fichier image à produire, la liste des formes à cacher.

2. Il lit l'image à traiter et la duplique pour créer l'image résultat.
3. Pour chaque forme à cacher, en fonction de sa forme et de ses paramètres, il calcule la moyenne des couleurs des pixels couverts par cette forme dans l'image d'origine puis remplace par cette couleur moyenne la couleur de ces mêmes pixels dans l'image résultat.
4. Une fois toutes les formes traitées, le script enregistre l'image résultat dans le fichier image à produire.

4 L'ébauche du script

Nous vous fournissions une première ébauche du script [anonymat-ebauche.py](#) (dont le code est visible dans les figures [7 page 8](#) et [8 page 9](#)).

Lors de la première phase, vous recopierez cette ébauche dans le fichier [anonymat-p1.py](#) pour pouvoir ensuite le modifier.

4.1 Environnement virtuel

Ce script utilise le module [simple_image.py](#) et (comme lors du TP sur les images) nécessite donc la création et l'activation d'un environnement virtuel Python.

Pour le créer, l'activer et le peupler, dans VS Code, après avoir ouvert le répertoire du projet, ouvrez un terminal et utilisez les trois commandes suivantes :

```
C:> python -m venv venv-projet  
C:> venv-projet\Scripts\activate.bat  
(venv-projet) C:> pip install packaging Pillow
```

4.2 Utilisation du script [anonymat-ebauche.py](#)

Vous pouvez déjà essayer ce script en utilisant la commande de la figure [1 page suivante](#).

Le script se déroule et indique (entre autres) qu'il lit l'image [images/chats.jpg](#) puis qu'il écrit l'image [images/chats-flou.png](#).

```
(venv-projet) C:> python anonymat-ebauche.py images\chats.json
Image à lire: images/chats.jpg
lecture d'une image (960x720) depuis le fichier 'images/chats.jpg'.
nouvelle image (960x720).
Liste des formes à flouter:
    Cercle de centre (285.0, 277.0) de rayon 100.0
    ** TODO: calculer la couleur moyenne du cercle dans 'im_in'
    ** TODO: appliquer cette couleur au cercle dans 'im_out'
    Cercle de centre (700.0, 289.0) de rayon 90.0
    ** TODO: calculer la couleur moyenne du cercle dans 'im_in'
    ** TODO: appliquer cette couleur au cercle dans 'im_out'
Image à produire: images/chats-flou.png
écriture d'une image (960x720) dans le fichier 'images/chats-flou.png'.
```

Figure 1 – Exécution du script `anonymat-ebauche.py`

Mais les deux images sont totalement identiques ! C'est normal puisque ce script est une simple *ébauche* !

Les fonctions principales sont déjà codées :

- La fonction `main` qui enchaîne les opérations principales : lecture des ordres puis exécutions des ordres.
- La fonction `read_orders_from_json` qui lit et vérifie les ordres lus depuis un fichier `json`.
- La fonction `clone_image` qui permet de cloner l'image reçue en paramètre.
- La fonction `exec_orders` qui affiche et exécute effectivement les ordres reçus en paramètre.

Cette dernière fonction (`exec_orders`) affiche des messages `TODO` (via des appels à `print`) indiquant une action à réaliser (mais non codée puisque c'est une ébauche).

5 Travail demandé

Le travail demandé se déroule en trois phases : la première phase est *obligatoire*, la deuxième phase est *fortement conseillée* et la troisième phase est *optionnelle*.

5.1 Première phase : les cercles

Le script produit lors de cette phase s'appellera `anonymat-p1.py`.

Vous devez remplacer les deux messages `TODO` (calculer la couleur moyenne du cercle dans `im_in` et appliquer cette couleur au cercle dans `im_out`) par des appels à deux fonctions que vous aurez préalablement conçues et codées :

- La fonction `average_color_circle` recevra comme paramètres une image à traiter `im`, le centre d'un cercle à traiter `cxy` (sous la forme d'un tuple : `(x, y)`) et son rayon du cercle `cr`. Elle calculera la couleur moyenne du cercle

dans l'image et retournera cette couleur (sous la forme d'un triplet `(r, g, b)`).

- Dans la fonction `exec_orders`, elle sera appelée avec comme paramètres l'image d'origine (`im_in`), le centre du cercle (le tuple `(shape["x"], shape["y"])`) et son rayon (`shape["cr"]`). La couleur résultat sera stockée dans la variable `average_color`.
- La fonction `fill_circle` recevra comme paramètres une image à modifier `im`, le centre d'un cercle à traiter `cxy` (sous la forme d'un tuple : `(x, y)`), le rayon de ce cercle `cr` et la couleur de remplissage (`color`). Dans l'image, elle remplira le cercle par la couleur de remplissage.

Dans la fonction `exec_orders`, elle sera appelée avec comme paramètres l'image à produire (`im_out`), le centre du cercle (le tuple `(shape["x"], shape["y"])`), son rayon (`shape["cr"]`) et la couleur moyenne `average_color` préalablement calculée.

Après cette première phase, votre script devrait être capable de produire les images de la figure 4 page 5.

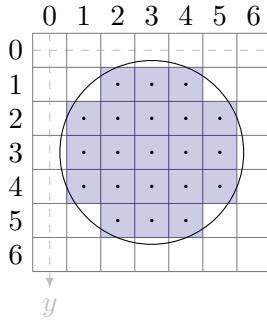
5.2 Deuxième phase : les rectangles

Le script produit lors de cette phase s'appellera `anonymat-p2.py`.

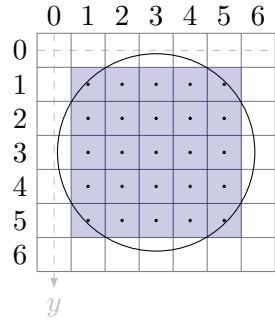
Si vous essayez le script de la phase précédente avec le fichier `images/test-rectangle.json`, vous constaterez qu'il produit encore des messages d'avertissement tel `** Forme 'rectangle' inconnue !`.

Dans le fichier `images/test-rectangle.json`, outre le type de forme `circle`, on trouve aussi le type `rectangle`.

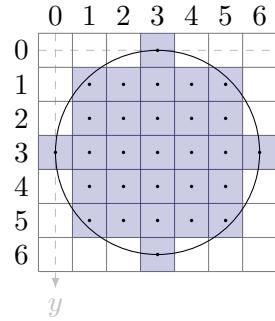
Cette seconde phase aura pour but d'ajouter le traitement des rectangles (à bords horizontaux et verticaux).



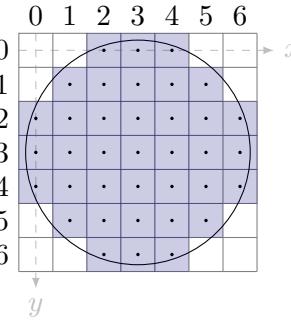
(a) Rayon 2.7



(b) Rayon 2.9



(c) Rayon 3.0



(d) Rayon 3.3

Figure 2 – Les pixels colorés font partie du cercle. Dans les quatre graphiques, le centre est en (3,3).

Un **rectangle** est défini par deux coins : (**c1x**, **c1y**) et (**c2x**, **c2y**).

En vous inspirant du code déjà écrit, il vous faudra :

- modifier la fonction **read_orders_from_json** pour qu'elle lise et vérifie les formes de type **rectangle**;
- modifier la fonction **exec_orders** pour qu'elle traite les forme de type **rectangle**;
- ajouter les fonctions nécessaires au calcul de la moyenne des couleurs d'un **rectangle** et à son remplissage avec cette couleur.

La figure 5 page 6 montre les résultats que vous devrez obtenir après cette deuxième phase (regardez bien les bords du rectangle orange dans l'image **test-flou-rectangle.png**).

5.3 Troisième phase : autres formes

Le script produit lors de cette phase s'appellera **anonymat-p3.py**.

Dans cette troisième phase, vous ajouterez un nouveau type de formes à traiter. Par exemple des ellipses (avec des axes horizontaux et verticaux), des losanges ou d'autres formes comme des polygones réguliers ou non.

Vous déterminerez les paramètres nécessaires (à fournir dans le fichier d'ordres), vous ajouterez le code nécessaire à son traitement et vous donnerez des exemples de résultats obtenus.

Si vous le voulez, vous pouvez aussi simplifier le code existant (par exemple si vous ajoutez le type **ellipse**, **circle** ne devient plus qu'un cas particulier de **ellipse**).

6 Éléments pratiques

Voici quelques éléments pratiques pour vous aider à réaliser votre script.

6.1 Quels pixels sont dans un cercle ?

On considère que les coordonnées d'un pixel sont le centre de ce pixel. Ainsi, le centre du pixel en haut à gauche d'une image (de coordonnées (0,0)) est l'origine de notre repère.

Pour qu'un pixel soit considéré comme faisant partie d'un cercle, son centre doit être dans le cercle. La figure 2 illustre ces explications.

Il pourra s'avérer pratique de concevoir et définir une fonction permettant de tester si un point est dans un cercle.

Note : les deux inéquations suivantes sont mathématiquement équivalentes (dans la mesure où r est positif) :

$$\sqrt{x^2 + y^2} \leq r$$

$$x^2 + y^2 \leq r^2$$

Mais informatiquement la seconde est plus rapide et plus précise !

6.2 Une couleur moyenne

Pour calculer la couleur moyenne d'un ensemble de pixels, il faut tout d'abord calculer trois sommes : la somme des composantes rouge, les somme des composantes verte et la somme des composantes bleu pour tous les pixels. Les trois composantes de la couleur moyenne sont ces trois sommes divisées par le nombre total de pixels. Note : une composante d'une couleur étant un entier, il faut arrondir le résultat de ces divisions.

6.3 Les calculs avec des entiers

En Python, pour maîtriser la manière dont une valeur réelle est tronquée pour devenir un nombre entier, vous pouvez utiliser les trois fonctions suivantes :

1. **ceil** qui renvoie le plus petit nombre entier n'étant pas strictement inférieur à son paramètre réel.

2. `floor` qui renvoie la plus grande valeur entière qui n'est pas strictement supérieure à son paramètre réel.
3. `round` qui renvoie la valeur entière la plus proche de son paramètre réel.

La fonction `round` fait partie intégrante du langage Python. Par contre les fonctions `ceil` et `floor` doivent être importées depuis le module `math` :

```
from math import ceil, floor
```

Le langage Python intègre aussi les fonctions `min` et `max` qui retournent le minimum et le maximum des valeurs qu'on leur passe comme paramètres.

6.4 Le contenu d'un fichier d'ordres

La figure 3 montre un exemple de fichier d'ordres demandant à cacher deux cercles.

Il contient un dictionnaire (`{}`) avec trois clés : `in`, `out` et `shapes` :

- Les clés `in` et `out` ont pour valeur les noms des fichiers image d'entrée et de sortie.
- La clé `shapes` est associée à une liste (`[]`) de formes à traiter.
- Chaque forme est décrite par un dictionnaire (`{}`) avec comme clé son `type` puis les paramètres décrivant cette forme (exemple pour `circle` : son centre (`x`, `y`) et son rayon `r`).

Note : dans un fichier `json`, l'ordre des clés dans un dictionnaire n'a pas d'importance. En revanche, dans une liste l'ordre des items, l'ordre est important (dans notre cas, cela indique comment les formes traitées seront éventuellement superposées).

```
{
  "in": "example.jpg",
  "out": "resultat.png",
  "shapes": [
    {
      "type": "circle",
      "x": 10.75,
      "y": 10,
      "r": 3
    },
    {
      "type": "rectangle",
      "c1x": 15,
      "c1y": 14.73,
      "c2x": 20.3,
      "c2y": 20
    }
  ]
}
```

Figure 3 – Exemple de fichier d'ordres (au format `json`).

6.5 Images à traiter

Dans le dossier `images` de l'archive du projet, nous vous fournissons des fichiers `.json` et des images `.jpg` ou `.png` à traiter. Chacun des fichiers `.json` contient des instructions à fournir au script `anonymat` pour traiter l'image correspondante.

Les figures 4 page suivante et 5 page 6 affichent côté à côté l'image d'origine et l'image résultat (après application du fichier d'ordres associé).

Outre les images et ordres fournis, vous devrez ajouter **au moins une image** (et le fichier d'ordres associé) que vous aurez vous-même choisie.

Note : les images d'origine sont au format `jpg` ou `png` mais les images produites sont *toujours* enregistrées dans des fichiers au format `.png` pour éviter les dégradations liées au format `.jpg`.

7 Quels algorithmes fournir ?

Pour chacune des fonctions Python que vous aurez ajoutées dans vos script `anonymat.py`, vous devrez fournir dans votre rapport :

- la description de son principe de fonctionnement,
- son schéma bulle,
- son algorithme.

La figure 6 page 7 donne l'exemple de la fonction `clone_image`.

8 Remarques

N'oubliez pas que les formes décrites dans une fichier d'ordres `json` peuvent être en partie (ou même *en totalité*) en dehors de l'image.

Comme leur paramètre sont des réels, elles peuvent aussi être tellement petites qu'elles n'englobent aucun pixel.

9 Comment vérifier votre code ?

Le répertoire `images-reference` contient le résultat des images après application des ordres fournis. Ces images peuvent vous servir de référence : vous pouvez comparer vos propres résultats avec ces images.

Il pourra être pratique de concevoir de d'écrire un script `comparaison.py` qui comparera deux fichiers image afin de vérifier que les images qu'ils contiennent sont bien identiques.



chats.jpg



chats-flou.jpg

(a) Application du fichier d'ordres images/chats.json.



une-personne.jpg



une-personne-flou.jpg

(b) Application du fichier d'ordres images/une-personne.json.



garcon.jpg



garcon-flou.jpg

(c) Application du fichier d'ordres images/garcon.json.



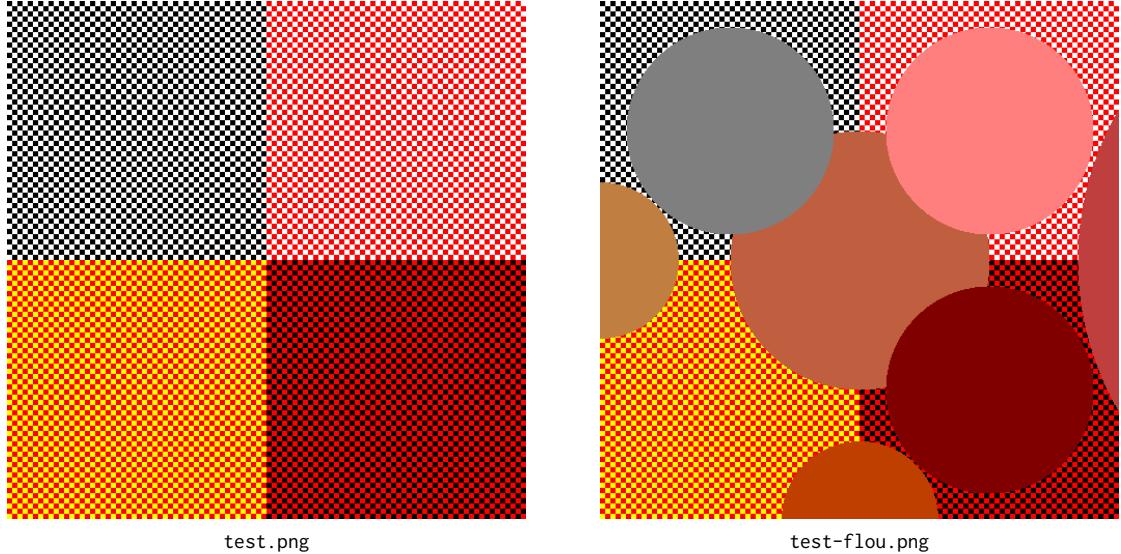
enfants.jpg



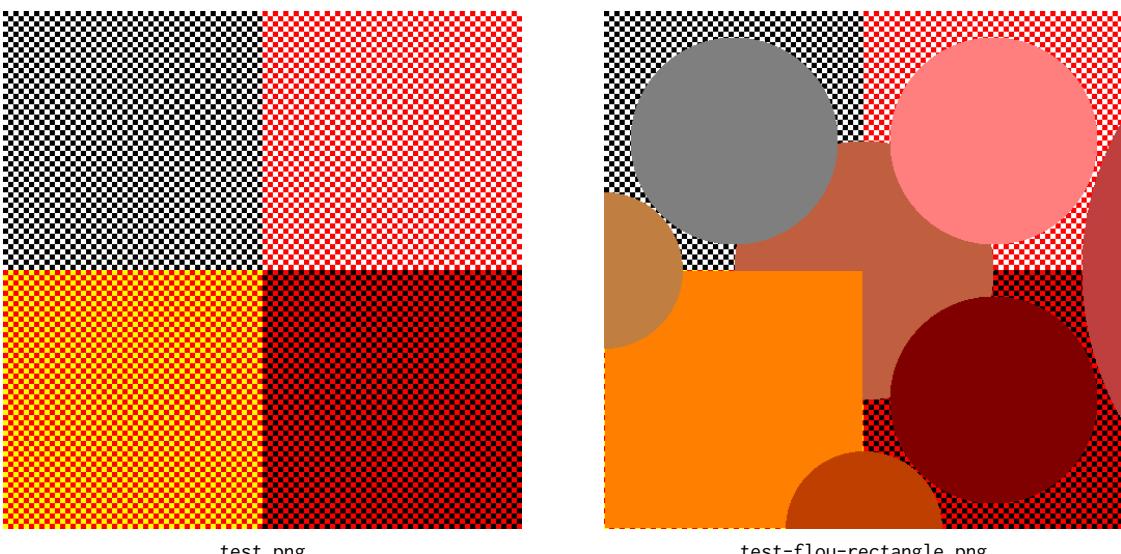
enfants-flou.jpg

(d) Application du fichier d'ordres images/enfants.json.

Figure 4 – Exemples de résultats d'utilisation du script `anonymat.py` sur différentes images (avec leur fichier d'ordres associé).



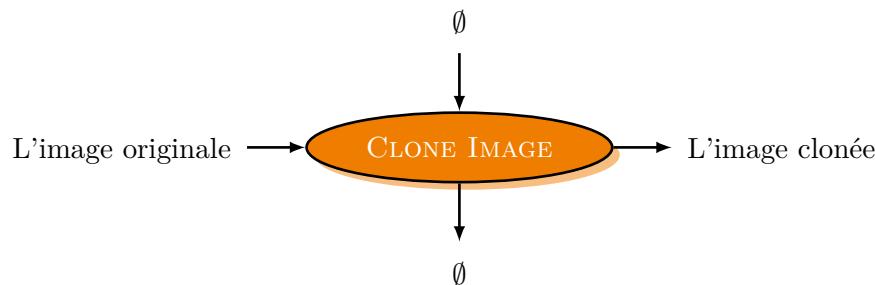
(a) Application du fichier d'ordres images/test.json.



(b) Application du fichier d'ordres images/test-rectangle.json.

Figure 5 – Application des ordres test.json puis test-rectangle.json au fichier image test.png.

La fonction CLONE IMAGE a pour but de cloner une image. On crée tout d'abord une nouvelle image de la même taille que l'image reçue en paramètre. Puis, avec une double boucle, on parcourt tous les pixels. À chaque fois, on récupère la couleur de ce pixel dans l'image originale et on l'utilise comme couleur de ce même pixel dans l'image résultat.



CLONE IMAGE(*im*)

Clone l'image *im* et retourne ce clone.

Paramètre : *im* (image) l'image à cloner.

Variables : *r*, *v*, *b* (entiers) les 3 composantes de la couleur d'un pixel.

Résultat : *res* (image) l'image clonée.

Début

```

- .-. - création d'une image vide de même taille que l'originale
res ← Image.new(im.width, im.height)
- .-. - parcours de tous les pixels de l'image
Pour x variant de 0 à im.width − 1 Faire
    Pour y variant de 0 à im.height − 1 Faire
        - .-. - récupération de la couleur du pixel dans l'image originale
        (r, v, b) ← im.get_color((x, y))
        - .-. - application de cette couleur au pixel dans l'image clonée
        res.set_color((x, y), (r, v, b))
    Fin Pour
Fin Pour
Retour (res)
Fin
  
```

Figure 6 – Exemple d'analyse, de schéma bulle et d'algorithme pour la fonction `clone_image`

```

1 #!/usr/bin/env python3
2 import os
3 import sys
4 import json
5 from simple_image import Image
6 from math import floor, ceil
7
8
9 def read_orders_from_json(json_filename):
10     """Lit et retourne les ordres depuis le fichier JSON nommé
11     `json_filename`."""
12     print(f'Lecture du fichier d'ordres "{json_filename}"')
13     with open(json_filename, 'r') as f:
14         orders = json.load(f)
15     # on vérifie la présence des 3 clés de base (et uniquement elles) et
16     # le type de valeurs associées
17     for key in orders.keys():
18         if key not in ["out", "in", "shapes"]:
19             print(f'** Clé "{key}" inconnue')
20     if "in" not in orders.keys() or not isinstance(orders["in"], str):
21         print("La clé 'in' doit être le nom du fichier image à lire")
22         sys.exit(1)
23     if "out" not in orders.keys() or not isinstance(orders["out"], str):
24         print("La clé 'out' doit être le nom du fichier image à produire")
25         sys.exit(1)
26     if "shapes" not in orders.keys() or not isinstance(orders["shapes"], list):
27         print("La clé 'shapes' doit être une liste de formes à flouter")
28         sys.exit(1)
29     # pour chaque type de formes on vérifie la présence des clés
30     # attendues
31     for shape in orders["shapes"]:
32         if "type" not in shape.keys():
33             print("Une forme doit définir la clé 'type'")
34             sys.exit(1)
35     if shape["type"] == "circle":
36         # les clés fournies ont-elles les noms attendus ?
37         for key in shape.keys():
38             if key not in ["type", "x", "y", "r"]:
39                 print(f'Clé "{key}" inconnue pour une forme "circle"')
40                 sys.exit(1)
41         # les clés attendues sont-elles présentes avec des valeurs
42         # du bon type (des entiers ou des réels) ?
43         for key in ["x", "y", "r"]:
44             if (key not in shape.keys() or not isinstance(shape[key], (int, float))):
45                 print(f"La clé '{key}' d'un 'circle' doit être un nombre")
46                 sys.exit(1)
47     else:
48         print(f'** Forme "{shape["type"]}" inconnue !')
49
50     # on retrouve le chemin d'accès des images `in` et `out`
51     # relativement au chemin d'accès du fichier JSON
52     dirname = os.path.dirname(json_filename)
53     orders["in"] = os.path.join(dirname, orders["in"])
54     orders["out"] = os.path.join(dirname, orders["out"])
55     return orders

```

Figure 7 – Source de l'ébauche du script anonymat-ebauche.py (partie 1)

```

58 def clone_image(im):
59     """Permet de cloner une image."""
60     res = Image.new(width=im.width, height=im.height)
61     for x in range(im.width):
62         for y in range(im.height):
63             (r, g, b) = im.get_color((x, y))
64             res.set_color((x, y), (r, g, b))
65     return res
66
67
68 def exec_orders(orders):
69     """Exécute les ordres `orders`..."""
70     print("Image à lire:", orders["in"])
71     im_in = Image.read(orders["in"])
72     im_out = clone_image(im_in)
73
74     if len(orders["shapes"]) == 0:
75         print("Pas de formes à flouter")
76     else:
77         print("Liste des formes à flouter:")
78         for shape in orders["shapes"]:
79             if shape["type"] == "circle":
80                 print(f" Cercle de centre ({shape['x']}, {shape['y']})",
81                      f"de rayon {shape['r']}")  

82                 print(" ** TODO: calculer la couleur moyenne du cercle dans 'im_in'")
83                 print(" ** TODO: appliquer cette couleur au cercle dans 'im_out'")
84             else:
85                 print(f" ** Forme '{shape['type']}' inconnue !")
86                 print(f" ** TODO: ajouter le traitement des formes de type '{shape['type']}'")
87
88     print("Image à produire:", orders["out"])
89     im_out.save(orders["out"])
90
91
92 def main():
93     """Programme principale qui lit ou demande un fichier d'ordres puis
94     les exécute."""
95     if len(sys.argv) == 2:
96         orders_filename = sys.argv[1]
97     else:
98         orders_filename = input("Nom du fichier d'ordres: ")
99
100    orders = read_orders_from_json(orders_filename)
101    exec_orders(orders)
102
103
104 if __name__ == "__main__":
105     main()

```

Figure 8 – Source de l'ébauche du script `anonymat-ebauche.py` (partie 2)