Scénario pédagogique
codage RLE : Run Length Encoding

Version initiale le 8 décembre 2020. Dernière mise à jour le 27 janvier 2021

Auteur : Sébastien LOZANO

Sommaire

1.1 Les consignes pour le projet

1.1.1 Sujet

L'objectif de ce projet est la réalisation d'un **scénario pédagogique** sur l'activité de **codage RLE** vue en cours. Le scénario doit contenir une partie sur l'**activité débranchée** et une sur le **rebranchement**, en faisant **coder** aux élèves un **programme passant d'une image au format pbm au codage RLE de l'image**. Le programme des élèves pourra, en fonction de la durée de séquence pédagogique choisie, soit se contenter du codage RLE d'une image en une fois, soit décomposer l'image (comme dans l'activité vue en cours) avant de créer le codage de chaque morceau d'image.

1.1.2 Attendus

Le scénario pédagogique doit contenir a minima :

- ⇝ les prérequis nécessaires avant la séquence.
- ⇝ les compétences travaillées pendant la séquence la durée de la séquence et sa décomposition prévue en séances (durée de chaque séance et nombre de séances).
- ⇝ pour chaque séance, le déroulé de la séance, avec les objectifs de la séance, les compétences travaillées, l'énoncé, les questions posées, les difficultés éventuelles et les pistes pour aider les élèves, ainsi que le matériel nécessaire.
- ⇝ une partie de justification des choix principaux (structure de la séquence, choix des exercices, ...).

1.2 La séquence

1.2.1 Introduction

Pour stocker les pixels d'une image de façon optimale, il est possible d'opérer par concaténation. Une image monochrome ne faisant appel qu'à deux couleurs, un pixel de l'image peut donc être codé sur un seul bit pour gagner de l'espace mémoire.
La méthode de compression RLE, Run Length Encoding, est basée sur la répétition d'éléments consécutifs.
C'est l'étude de cette méthode de compression, d'abord de façon débranchée puis rebranchée, sur laquelle nous allons appuyer la construction de cette séquence pédagogique.

1.2.2 Prérequis

Partie débranchée

Cette partie est abordable dès l'école primaire. Seules sont nécessaires les aptitudes à :

- ⇝ Compter.
- ⇝ Dessiner.

Source : Computer Science Unplugged ⩾ csunplugged.org

Partie rebranchée

Les prérequis d'algorithmique de fin de cycle 4.

Algorithmique et programmation		
Écrire, mettre au point, exécuter un programme		
Les repères qui suivent indiquent une progressivité dans le **niveau de complexité des activités** relevant de ce thème. Certains élèves sont capables de réaliser des activités de troisième niveau dès le début du cycle.		
1er niveau	**2e niveau**	**3e niveau**
À un premier niveau, les élèves mettent en ordre et/ou complètent des blocs Scratch fournis par le professeur pour construire un programme simple. L'utilisation progressive des instructions conditionnelles et/ou de la boucle « répéter ... fois ») permet d'écrire des scripts de déplacement, de construction géométrique ou de programme de calcul.	À un deuxième niveau, les connaissances et les compétences en algorithmique et en programmation s'élargissent par : ⇝ l'écriture d'une séquence d'instructions (condition « si ... alors » et boucle « répéter ... fois ») ; ⇝ l'écriture de programmes déclenchés par des événements extérieurs ; ⇝ l'intégration d'une variable dans un programme de déplacement, de construction géométrique, de calcul ou de simulation d'une expérience aléatoire.	À un troisième niveau, l'utilisation simultanée de boucles « répéter ... fois », et « répéter jusqu'à ... » et d'instructions conditionnelles permet de réaliser des figures, des calculs et des déplacements plus complexes. L'écriture de plusieurs scripts fonctionnant en parallèle permet de gérer les interactions et de créer des jeux. La décomposition d'un problème en sous-problèmes et la traduction d'un sous-problème par la création d'un bloc-utilisateur contribuent au développement des compétences visées.

Quelques syntaxes python :

- ⇝ Structure conditionnelle.
- ⇝ Boucles itératives for, while, ...
- ⇝ La manipulation de fichier, ouverture, lecture, écriture.
- ⇝ Quelques méthodes de manipulation des chaînes de caractères.

En annexe du document, une partie est réservée à la description de méthodes utiles à la réalisation des scripts demandés aux élèves. Notamment sur la manipulation des chaînes.
La présentation des méthodes de manipulations des fichiers sera faite sous forme d'exercices pratiques. Notamment pour la gestion des fichiers : ouverture, lecture, écriture.

Documentation sur la manipulation des fichiers en annexe (Click Me !)

Documentation sur la manipulation des chaînes en annexe (Click Me !)

1.2.3 Descriptif global

Introduction :

La séquence vise à exposer :

- ⇝ Un petit historique sur l'encodage matriciel des images. On fera quelques décodages à la main.
- ⇝ L'intérêt de la compression d'un tel encodage et la réponse qu'y apporte le **codage RLE** [a]. On fera quelques encodages puis quelques décodages à la main.
- ⇝ Une partie sera réservée au développement de programmes python autour des images au format ***.pbm** [b] :
  - ⩾ à partir de la donnée d'un script modèle.
  - ⩾ sous forme de défis à partir d'une ou de plusieurs images à générer à l'aide de scripts.
- ⇝ Une autre partie sera réservée au développement de programmes autour du **codage RLE** :
  - ⩾ un programme python permettant de compresser une image ***.pbm** selon le **codage RLE**.
  - ⩾ un programme python permettant de décompresser une image préalablement compressée selon le **codage RLE** et de générer l'image au format ***.pbm**

a. RLE : **R**un **L**ength **E**ncoding
b. pbm : **p**ortable **bitm**ap

Durée :

4 séances d'1h30 à 2h, à éventuellement affiner.

Compétences travaillées

- ⇝ Décomposer un problème en sous-problèmes, reconnaître des situations déjà analysées et réutiliser des solutions.
- ⇝ Concevoir des solutions algorithmiques.
- ⇝ Traduire un algorithme dans un langage de programmation, en spécifier les interfaces et les interactions.
- ⇝ Comprendre et réutiliser des codes sources existants.
- ⇝ Développer des processus de mise au point et de validation de programmes.

Décomposition en séances

Globalement la séquence sera découpée en 3 ou 4 séances dont la durée pourra varier entre 1h30 et 2h00.

- ⇝ **Séance 1 et 2 - débranchées :** Analyse, prise en main, commentaires historiques, compression des images.
  - ⩾ Travail débranché sur les images matricielles.
  - ⩾ Travail débranché sur la compression RLE, principalement sur le décodage.
- ⇝ **Séance 3 - rebranchée :** Travail sur la manipulation des fichiers à partir de python via la mise au point de programmes pour générer des images au format ***.pbm**
- ⇝ **Séance 4 - rebranchée :** Travail spécifique sur la programmation autour de l'encodage RLE .

1.3 Les séances

1.3.1 Images matricielles et compression RLE - débranché

Déroulé

1/ Questions pour lancer la discussion, prétexte à une introduction historique, fax, monochrome ...

2/ Codage puis décodage d'une partie de l'alphabet pixelart en noir et blanc.

3/ Compression puis décompression de ces mêmes images.

Objectifs

L'objectif est de faire comprendre comment fonctionne le codage RLE sur des images en noir et blanc et de mettre en avant qu'il permet un gain de place de stockage par rapport à un codage matriciel des images.

Compétences travaillées

- ⇝ Analyse d'un problème ;
- ⇝ Décomposer un problème en sous-problèmes ;

Énoncé/Questions

1/ Questionnement

a) Dans quelles circonstances les ordinateurs ont-ils besoin de stocker des images ?
(Un programme de dessin, un jeu avec des graphiques, ou un système multimédia).

b) Comment les ordinateurs peuvent-ils stocker des images alors qu'ils n'utilisent que des nombres ?

Le format PBM a été défini par Jef Poskanzer dans les années 1980 comme un format d'images matricielles monochromes pouvant être transmises via un message électronique en texte ASCII, et permettant de supporter tout changement dans le formatage du texte.

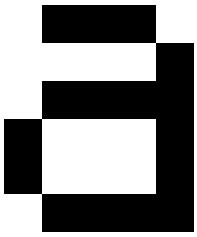
2/ Codage matriciel

Les écrans d'ordinateur sont divisés en une grille de petits points appelés pixels, compression de l'anglais **picture elements** qui signifie éléments d'image.

Dans une image en noir et blanc, chaque pixel est soit noir, soit blanc, on peut donc le représenter par un bit 0 pour le blanc et 1 pour le noir.

La lettre « a » et son codage matriciel ont été agrandis ci-dessous pour bien voir les pixels.

Rendu graphique

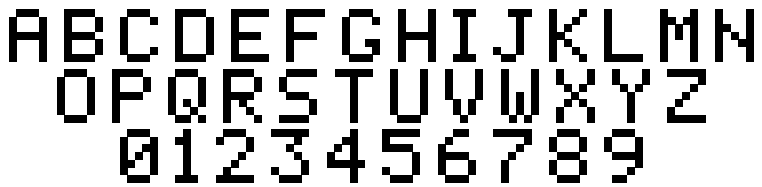


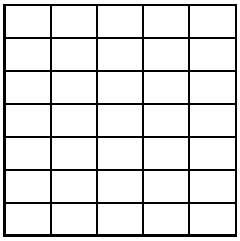
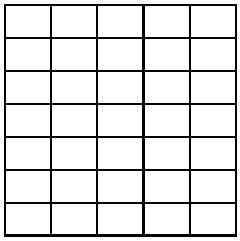
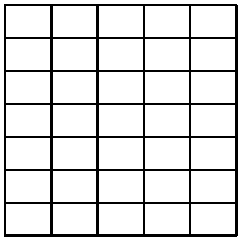
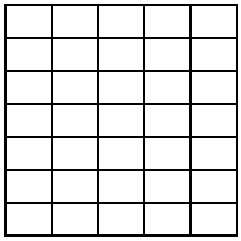
Codage matriciel

0	1	1	1	0
0	0	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	0	1
0	1	1	1	1

Pour le codage matriciel, on pourrait écrire la suite de 0 et de 1 en ligne et indiquer quelque part les dimensions de l'image, la largeur et sa hauteur. Pour l'instant afin d'avoir plus de lisibilité, nous le représenterons sous forme de tableau. On peut déjà se poser la question de la façon dont on va stocker l'image avec python.

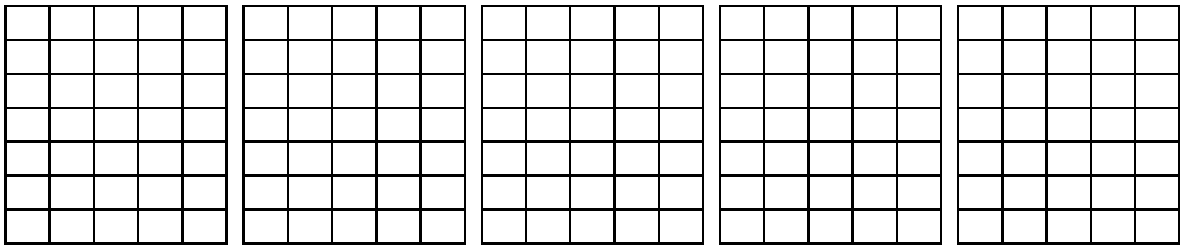
a) Coder deux des 36 caractères suivants au choix, à l'aide des grilles ci-dessous.



Rendu graphique	Codage matriciel	Rendu graphique	Codage matriciel
			

b) Sachant que les images font 5 pixels sur 7 pixels, trouver le rendu graphique correspondant à cette suite de codages :

- ⇝ 11110100011000111110100011000111110
- ⇝ 11110100011000111110101001001010001
- ⇝ 01110100011000111111100011000110001
- ⇝ 10001100011000110001010100101000100
- ⇝ 01110100011000110001100011000101110



3/ Ce type de codage n'est pas très efficace et peut prendre beaucoup de place en mémoire. En effet, il faudrait 10 000 zéros pour coder une image de 100 × 100 pixels blancs. Le **codage RLE**[a] permet de compresser de telles images en ne stockant plus chaque pixel un par un, mais en donnant la longueur des plages de pixels consécutifs de la même couleur. Nous allons maintenant prendre de nouvelles lettres et comparer la longueur du code compressé et du code non compressé. Plusieurs remarques :

- ⇝ Dans le **codage RLE**, on ne se préoccupe **pas des retours à la ligne**. Si l'image fait 10 pixels de large et que le code indique de colorier 12 pixels, on colorie toute la ligne plus les deux premiers pixels de la suivante.
- ⇝ Il faut une convention pour savoir par quelle couleur commencer, blanc ou noir.
  On choisira de **commencer par le blanc**.
  Ainsi , si une image commence par :
  - ➢ 3 pixels blancs, le codage RLE commence par "3".
  - ➢ 3 pixels noirs, le codage RLE va commencer par "0 3".
    On a en effet 0 pixels blancs suivis de 3 pixels noirs.

Voici les versions matricielles de quatre lettres au format (5 pixels) × (7pixels), donner leur **codage RLE**.

a) Codage matriciel aplati de la lettre : 10001110111010110101100011000110001

Codage RLE : .....

b) Codage matriciel aplati de la lettre : 01110100011000111111100011000110001

Codage RLE : .....

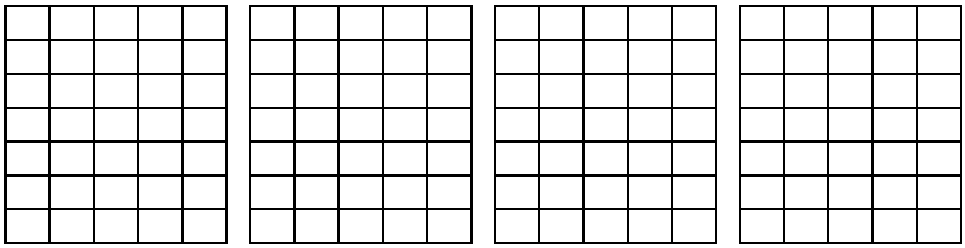
c) Codage matriciel aplati de la lettre : 11111001000010000100001000010000100

Codage RLE : .....

d) Codage matriciel aplati de la lettre : 10001100011000111111100011000110001

Codage RLE : .....

e) Vérifier les codages RLE à l'aide des grilles suivantes



a. RLE : **R**un **L**ength **E**ncoding
Source : Partiellement, Computer Science Unplugged ⩾ csunplugged.org
Source : Partiellement, le blog de Marie DUFLOT-KREMER ⩾ Dessine-moi un pixel !

Difficultés et pistes

⇝ La lecture sans se perdre dans les suites de chiffres.

Matériel

⇝ Un vidéoprojecteur.

⇝ Des photocopies de la "fiche élève" énoncé/réponse en annexe. (Click Me !)

Prolongement possible

Activité pikachu. Les élèves peuvent décoder seul, en binôme, à quatre. Fiche spécifique en annexe (Click Me !)

Énoncé/Questions

Reconstituer une grande image en accolant les 4 images ci-dessous, toutes de dimension 10 x 10 pixels.
Il faut mettre la 1 en haut à gauche, la 2 en haut à droite, la 3 en bas à gauche et la 4 en bas à droite.

⇝ code image 1 : 0 3 7 1 2 2 5 2 2 6 1 1 9 1 9 2 8 1 3 1 4 2 2 2 3 2 6 1 2 2 2 1 5

⇝ code image 2 : 11 7 2 2 3 3 7 2 6 2 7 2 6 1 2 1 6 1 2 2 9 1 5 1 2 2 5

⇝ code image 3 : 1 2 3 3 2 1 8 2 8 1 9 1 2 1 6 1 1 1 7 2 9 2 3 3 1 1 2 4 1 2 1 3 6

⇝ code image 4 : 2 2 9 1 4 2 3 2 3 2 4 1 1 3 2 1 2 1 1 3 3 1 1 3 6 2 7 2 6 2 2 1 6 3 6

image 1

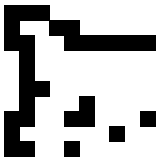
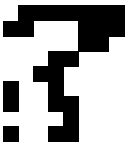


image 2



global

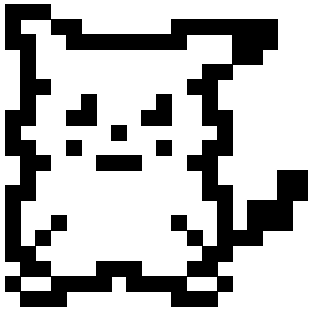


image 3

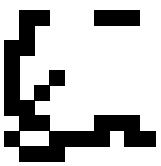
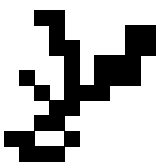


image 4



Source : Partiellement, le blog de Marie DUFLOT-KREMER ⩾ Dessine-moi un pixel !

1.3.2 Programmes de création d'une image *.pbm

Introduction

Avant de créer le script permettant de générer la compression RLE d'une image ***.pbm**, nous allons créer quelques scripts afin d'utiliser les méthodes de manipulation des fichiers en python.

- ⇝ Documentation sur la manipulation des fichiers en annexe (Click Me !)
- ⇝ Documentation sur la manipulation des chaînes en annexe (Click Me !)

Les fichiers PBM, PGM ou PPM sont composés sur la même base :

- ⇝ le nombre magique du format (deux octets) : il indique le type de format (PBM, PGM, ou PPM) et la variante (binaire ou ASCII) ;
- ⇝ un caractère d'espacement (espace, tabulation, nouvelle ligne) ;
- ⇝ la largeur de l'image (nombre de pixels, écrit explicitement sous forme d'un nombre en caractères ASCII) ;
- ⇝ un caractère d'espacement ;
- ⇝ la hauteur de l'image (idem) ;
- ⇝ un caractère d'espacement ;
- ⇝ les données de l'image : succession des valeurs associées à chaque pixel,
  - ⩾ l'image est codée ligne par ligne en partant du haut,
  - ⩾ chaque ligne est codée de gauche à droite.
- ⇝ Toutes les lignes commençant par croisillon # sont ignorées (lignes de commentaires).

Déroulé

A priori 1 séance de 1h30 à 2h. Seul ou en binôme.

- ⇝ On fournit un script exemple à imiter.
- ⇝ Les élèves doivent créer une ou plusieurs fonctions permettant de générer une image au format ***.pbm**

Objectifs

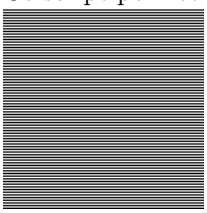
- ⇝ Imitation.
- ⇝ Coopération.
- ⇝ Faire des tests de validation.
- ⇝ Boucles.
- ⇝ Manipulation de fichiers.
- ⇝ ...

Compétences travaillées

- ⇝ Concevoir des solutions algorithmiques ;
- ⇝ Comprendre et réutiliser des codes sources existants.
- ⇝ Autonomie, les consignes sont moins guidées.
- ⇝ Debug, retour sur les erreurs de compilation.

Script

Le script suivant et ses commentaires donnent les manipulations nécessaires à la création d'images au format ***.pbm**, plus de détails sur les méthodes en annexe si nécessaire. (Click Me !)
Ce script permet d'obtenir l'image suivante :



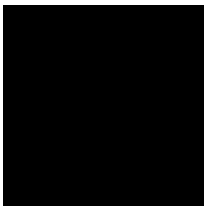
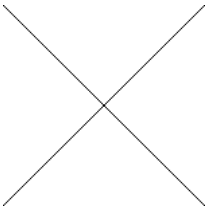
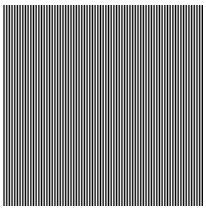
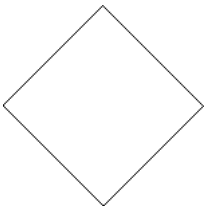
Source : IREM de Lyon, formation I.S.N 2018-2019 > Création d'images Portable Bitmap

1.1 – Création d'images - Lignes horizontales noires et blanches alternées

```
##----- On définit les dimensions de l'image -----##
largeur = 150
hauteur = 150

##----- Ouverture/création des Fichiers -----##
f = open("images_pbm/Lignes_Horizontales.pbm", "w")

##------ Écriture dans le fichier ------##
# P1 en ligne 1 pour déclarer le format .pbm puis passage à la ligne
f.write("P1 \n")
# Largeur et hauteur de l'image, séparées par un espace#
f.write('{} {} \n'.format(largeur, hauteur))

##------ Instructions principales ------##
for ligne in range(hauteur):
        for colonne in range(largeur):
                # Lorsque le numéro de ligne est pair
                if ligne%2 == 0:
                        f.write('0')
                else:
                        f.write('1')
        f.write('\n')

##----- Fermeture des Fichiers -----##
f.close()
```

Énoncé/Questions

Seul ou en binôme, écrire un script python qui permet d'obtenir le fichier ***.pbm** pour au moins deux de ces images au choix. Prendre une largeur de 150 et une hauteur de 150.

Source : IREM de Lyon, formation I.S.N 2018-2019 > Création d'images Portable Bitmap

Difficultés et pistes

- ⇝ Ajustement de la boucle pour générer la matrice de l'image.
- ⇝ Aide sur la manipulation des fichiers en annexe. (Click Me !)

Matériel

- ⇝ Machine avec IDE et environnement python.
- ⇝ Accès internet pour la documentation python.
- ⇝ Des photocopies de la "fiche élève" énoncé/réponse en annexe. (Click Me !)

Prolongement possible

les format ***.pgm** pour les niveaux de gris ou ***.ppm** pour la couleur

1.3.3 Programmes de compression/décompression d'un code RLE vers une image *.pbm

Introduction

Création de fonctions permettant :

- ⇝ Générer le **code RLE** d'une image *.pbm
- ⇝ Générer un **fichier *.pbm** à partir d'un codage RLE

Déroulé

A priori 1 séance de 1h30 à 2h. Seul ou en binôme.

1/ On rappelle rapidement les commandes de manipulation des fichiers introduites avec les scripts de la séance précédente.

2/ On donne les consignes.

3/ On donne un fichier avec les méthodes utiles qui peut être séparé de ce fichier global.

Objectifs

- ⇝ Coopération.
- ⇝ Faire des tests de validation.
- ⇝ Boucles.
- ⇝ Tableaux.
- ⇝ Manipulation de fichiers.
- ⇝ Manipulation de chaînes.

Compétences travaillées

- ⇝ Décomposition d'un problème en sous-problèmes, reconnaître des situations déjà analysées et réutiliser des solutions.
- ⇝ Concevoir des solutions algorithmiques.
- ⇝ Traduire un algorithme dans un langage de programmation, en spécifier les interfaces et les interactions, comprendre et réutiliser des codes sources existants, développer des processus de mise au point et de validation de programmes.
- ⇝ Autonomie, les consignes sont moins guidées.
- ⇝ Debug, retour sur les erreurs de compilation.
- ⇝ Développement de tests pour valider les scripts.

Énoncé/Questions

On pourra travailler à partir des fichiers de la séance précédente ou à partir de nouveaux.

1/ Compression PBM to RLE : Écrire deux fonctions pour

a) récupérer les dimensions de l'image ***.pbm** et aplatir la matrice de 0 et de 1.

b) générer le **code RLE** tout en conservant les dimensions de l'image d'origine.

2/ Décompression RLE to PBM : Écrire deux fonctions pour

a) générer la matrice de 0 et de 1 aplatie à partir du **code RLE** et des dimensions de l'image.

b) générer le fichier ***.pbm**

Difficultés et pistes

⇝ Méthodes de nettoyage des chaînes. Notamment pour la suppression des blancs et des retours à la ligne invisibles dans les fichiers *.pbm pour la création du code RLE.

⇝ Méthodes d'écriture dans un fichier pour le formatage pour le fichier *.pbm

Matériel

⇝ Machine avec IDE et environnement python.

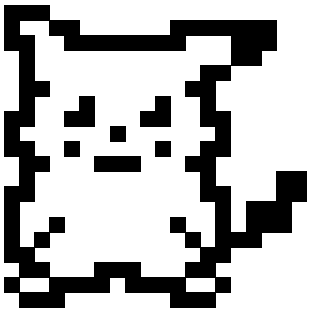
⇝ Accès internet pour la documentation python.

⇝ Des photocopies de la "fiche élève" énoncé/réponse en annexe. (Click Me !)

Prolongement possible

Travailler à partir d'une image globale et créer les découpages.
Permettant de générer des exercices du type de Pikachu décrit dans la section précédente.
Il pourra être utile de créer les images nécessaires au format PBM ainsi que leur codage RLE à partir des fonctions réalisées.

global



1.4 Justification des choix

Structure de la séquence, choix des exercices,...

Éléments du programme de 1ere - pour mémoire

⇝ Un enseignement d'informatique ne saurait se réduire à une présentation de concepts ou de méthodes sans permettre aux élèves de se les approprier en développant des projets applicatifs. Une part de l'horaire de l'enseignement d'au moins un quart du total en classe de première doit être réservée à la conception et à l'élaboration de projets conduits par des groupes de deux à quatre élèves.

⇝ **Représentation des données : types construits**

Contenus	Capacités attendues	Commentaires
Tableau indexé, tableau donné en compréhension	Lire et modifier les éléments d'un tableau grâce à leurs index. Construire un tableau par compréhension. Utiliser des tableaux de tableaux pour représenter des matrices : notation a [i] [j]. Itérer sur les éléments d'un tableau.	Seuls les tableaux dont les éléments sont du même type sont présentés. Aucune connaissance des tranches (slices) n'est exigible. L'aspect dynamique des tableaux de Python n'est pas évoqué. Python identifie listes et tableaux. Il n'est pas fait référence aux tableaux de la bibliothèque NumPy.

⇝ **Langages et programmation**

Contenus	Capacités attendues	Commentaires
Constructions élémentaires	Mettre en évidence un corpus de constructions élémentaires.	Séquences, affectation, conditionnelles, boucles bornées, boucles non bornées, appels de fonction.

⇝ **Algorithmique**

Contenus	Capacités attendues	Commentaires
Parcours séquentiel d'un tableau	Écrire un algorithme de recherche d'une occurrence sur des valeurs de type quelconque. Écrire un algorithme de recherche d'un extrémum, de calcul d'une moyenne.	On montre que le coût est linéaire.

Pourquoi ces choix ?

1/ On commence par des manipulations débranchées sur le format *.pbm et sur le codage RLE.
Cela permet d'intégrer les formats et de commencer à réfléchir à l'élaboration des scripts qui seront demandés ensuite. On prolonge sur l'activité pikachu selon le temps disponible, c'est une activité ludique qui permet de jauger la collaboration entre pairs et l'esprit de corps.

2/ Une fois ce temps débranché passé, on rebranche sur le codage des scripts pour générer des images au format ***.pbm**. L'objectif principal est la prise en main des méthodes de manipulation des fichiers, mais également le travail des boucles imbriquées et de la structure de contrôle "if".

3/ On termine par l'écriture de scripts de compression/décompression RLE, qui sont un peu plus complexes, l'objectif principal est le réinvestissement des notions précédentes, mais également la mise en place de tests pour valider les fonctions produites.

4/ En prolongement de cette dernière séance, on peut réinvestir le travail sur pikachu.

1.5 Annexes

En cas de défaut d'accès internet, on pourra toujours partager un fichier avec les méthodes utiles !

1.5.1 Python - Pour mémoire

Quelques liens en cas de besoin ...

Documentation officielle de Python en français à consulter en ligne

append()

str()

int()

Contrôle de flux, if, for, range()

1.5.2 Python - Manipulation des fichiers

1.2 – Ouverture d'un fichier

```
objet_fichier = open('chemin/vers/monFichier','modeLecture')
```

La méthode open() :

La méthode **open()** permet d'ouvrir un fichier.
Le chemin est à renseigner relativement au dossier contenant le script.
Les modes de lecture sont :

- ⇝ **r**, read, ouverture en lecture.
- ⇝ **w**, write, ouverture en écriture.
  Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est créé.
- ⇝ **a**, append, ouverture en mode ajout.
  On écrit à la fin du fichier sans écraser l'ancien contenu.

Le signe b peut être ajouté pour ouvrir le fichier en mode binaire.
La méthode **writelines()** permet d'écrire plusieurs lignes à la fois.

1.3 – Écrire dans un fichier - une ligne à la fois

```
## variables
ageDePier = 35
ageDePol = 47
ageDeJak  = 27
## ouverture d'un fichier en écriture, ajout d'une ligne à la fois
file = open('ages.txt','w')
file.write("Ages de Pier, Pol et Jak : \n")
file.write("Pier : " + str(ageDePier) + "\n")
file.write("Pol : " + str(ageDePol) + "\n")
file.write("Jak : " + str(ageDeJak) + "\n")
## fermeture du fichier
file.close()
```

1.4 – Écrire dans un fichier - plusieurs lignes en même temps

```
## ouverture d'un fichier en écriture, ajout de plusieurs lignes à la fois
file = open('agesMulti.txt','w')
file.write("Ages de Pier, Pol et Jak : \n")
file.writelines(["Pier : "+str(ageDePier)+"\n","Pol : "+str(ageDePol)+"\n"])
file.write("Jak : " + str(ageDeJak) + "\n")
## fermeture du fichier
file.close()
```

Résultat :

Les deux scripts ci-dessus créent un fichier avec le même contenu. Les fichiers **ages.txt** et **agesMulti.txt** contiennent donc :
Ages de Pier, Pol et Jak :
Pier : 35
Pol : 47
Jak : 27
Les méthodes **write()** et **writelines()** n'acceptent que des chaînes de caractères d'où l'utilisation de **str()**.

1.5 – Lire un fichier

```
## ouverture d'un fichier en lecture
file = open('ages.txt','r')
allContent = file.read()
print(allContent)
## fermeture du fichier
file.close()
```

Résultat :

L'ensemble du contenu du fichier est lu et affiché :
Ages de Pier, Pol et Jak :
Pier : 35
Pol : 47
Jak : 27

1.6 – Lire une partie d'un fichier

```
## ouverture d'un fichier en lecture
file = open('ages.txt','r')
partialContent = file.read(10)
print(partialContent)
partialContent = file.read(21)
print(partialContent)
## fermeture du fichier
file.close()
```

Résultat :

Ages de Pi
er, Pol et Jak :
Pie

1.7 – Lire une ligne d'un fichier

```
## ouverture d'un fichier en lecture
file = open('ages.txt','r')
## On saute la première ligne
file.readline()
## deuxième ligne
secondLine = file.readline()
print(secondLine)
## troisième ligne
thirdLine = file.readline()
print(thirdLine)
## fermeture du fichier
file.close()
```

Résultat :

Pier : 35

Pol : 47

1.5.3 Python - manipulation des chaînes

La méthode format() :

La méthode **format()** permet d'intégrer la chaîne directement et ainsi éviter les concaténations.
Les { } sont remplacées par les objets passés en paramètres à la méthode en respectant l'ordre.

1.8 – Utiliser la méthode format() de la classe Formatter

```
# Python3 program to demonstrate the str.format() method
# Source : https://www.geeksforgeeks.org/python-format-function/

# using format option in a simple string
print ("{}, A {} for geeks.".format("GeeksforGeeks","computer science portal"))

# using format option for a value stored in a variable
print ("This article is written in {}".format("Python"))

# formatting a string using a numeric constant
print ("Hello, I am {} years old !".format(18))
```

Résultat :

GeeksforGeeks, A computer science portal for geeks.
This article is written in Python
Hello, I am 18 years old !

La méthode strip([chars]) :

Retourne une copie de la chaîne sans les caractères passés en argument via **chars** en début ET en fin de chaîne. Si **chars** est omis ou None, la méthode supprime les espaces. ATTENTION, **chars** n'est pas un préfixe ou un suffixe, au contraire, toutes les combinaisons de ses valeurs sont supprimées.
Les cousines **lstrip()** et **rstrip()** font la même chose mais respectivement uniquement à gauche ou à droite.

1.9 – Méthodes strip() lstrip() et rstrip()

```
# Source : https://docs.python.org/3/library/stdtypes.html#str.strip

# On nettoie les espaces en début et en fin de chaîne
print("    spacious    ".strip())

# On supprime toutes les combinaisons possibles de 'cmowz.'
print("www.example.com".strip('cmowz.'))

# On nettoie uniquement en début ET en fin de chaine pas au milieu
print("#....... Section 3.2.1 Issue #32 .......".strip('.#! '))
```

Résultat :

spacious
example
Section 3.2.1 Issue #32

La méthode split(sep=None, maxsplit=-1) :

⇝ Renvoie une liste des mots de la chaîne, en utilisant sep comme séparateur de mots. Si maxsplit est donné, c'est le nombre maximum de divisions qui pourra être effectué.

⇝ Si sep est donné, les délimiteurs consécutifs ne sont pas regroupés et ainsi délimitent des chaînes vides.

⇝ Si sep n'est pas donné ou None, les espaces consécutifs sont considérés comme un seul séparateur. Le résultat ne contiendra pas les chaînes vides de début et de fin si la chaîne est préfixée ou suffixée d'espaces.

1.10 – Méthode split()

```
# Source : https://docs.python.org/fr/3/library/stdtypes.html?#str.split

# sep est donné
print('1,2,3'.split(','))
# sep donné, on ne traite que le premier séparateur
print('1,2,3'.split(',',maxsplit=1))

# sep donné, les délimiteurs sonsécutifs ne sont pas regroupés
print('1,2,,3,'.split(','))

# sep non donné, le séparateur utilisé par défaut et le regroupement d'espaces consécutifs
print('1 2 3'.split())
# Seul le premier espace est traiter
print('1 2 3'.split(maxsplit=1))

# plusieurs espaces sont considérés comme un seul separateur
print('   1   2   3   '.split())
```

Résultat :

'1,2,3'.split(',') donne : ['1', '2', '3'] **et** '1,2,3'.split(',',maxsplit=1) donne : ['1', '2,3']
'1,2,,3,'.split(',') donne : ['1', '2', '', '3', '']
'1 2 3'.split() donne : ['1','2','3'] **et** '1 2 3'.split(maxsplit=1) donne : ['1', '2 3']
' 1 2 3 '.split() donne : ['1', '2', '3']

La méthode replace(old, new[, count]) :

Renvoie une copie de la chaîne dont toutes les occurrences de la sous-chaîne **old** sont remplacées par **new**. Si l'argument optionnel **count** est donné, seules les **count** premières occurrences sont remplacées.

1.11 – Méthode replace()

```
# Source : https://docs.python.org/fr/3/library/stdtypes.html?#str.replace

str_replace = "esope reste et se repose"
print(str_replace.replace(" ",""))
```

Résultats :

esoperesteetserepose

FICHE ÉLÈVE

Fiche élève activité RLE - Débranchée

Nom : **Prénom :** **Classe :**

1/ Questionnement

a) Dans quelles circonstances les ordinateurs ont-ils besoin de stocker des images ?

.....

.....

b) Comment les ordinateurs peuvent-ils stocker des images alors qu'ils n'utilisent que des nombres ?

.....

.....

.....

.....

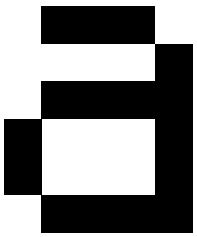
2/ Codage matriciel

Les écrans d'ordinateur sont divisés en une grille de petits points appelés pixels, compression de l'anglais **picture elements** qui signifie éléments d'image.

Dans une image en noir et blanc, chaque pixel est soit noir, soit blanc, on peut donc le représenter par un bit 0 pour le blanc et 1 pour le noir.

La lettre « a » et son codage matriciel ont été agrandis ci-dessous pour bien voir les pixels.

Rendu graphique

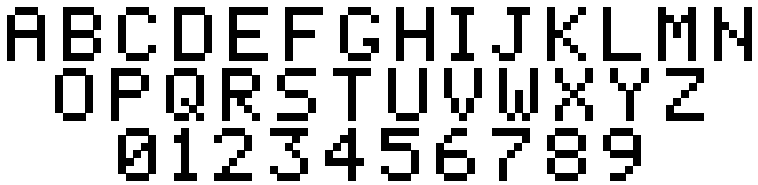


Codage matriciel

0	1	1	1	0
0	0	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	0	1
0	1	1	1	1

Pour le codage matriciel, on pourrait écrire la suite de 0 et de 1 en ligne et indiquer quelque part les dimensions de l'image, la largeur et sa hauteur. Pour l'instant afin d'avoir plus de lisibilité, nous le représenterons sous forme de tableau. On peut déjà se poser la question de la façon dont on va stocker l'image avec python.

a) Coder deux des 36 caractères suivants au choix, à l'aide des grilles ci-dessous.



Rendu graphique **Codage matriciel** **Rendu graphique** **Codage matriciel**

b) Sachant que les images font 5 pixels sur 7 pixels, trouver le rendu graphique correspondant à cette suite de codages :

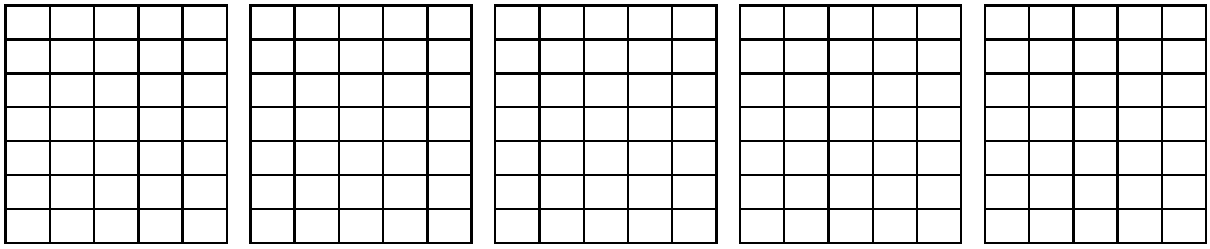
⇝ 11110100011000111110100011000111110

⇝ 11110100011000111110101001001010001

⇝ 01110100011000111111100011000110001

⇝ 10001100011000110001010100101000100

⇝ 01110100011000110001100011000101110



3/ Ce type de codage n'est pas très efficace et peut prendre beaucoup de place en mémoire. Il faudrait 10 000 zéros pour coder une image de 100 × 100 pixels blancs. Le **codage RLE** (i) permet de compresser de telles images en ne stockant plus chaque pixel un par un, mais en donnant la longueur des plages de pixels consécutifs de la même couleur. Nous allons maintenant prendre de nouvelles lettres et comparer la longueur du code compressé et du code non compressé. Plusieurs remarques :

⇝ Dans le **codage RLE**, on ne se préoccupe **pas des retours à la ligne**. Si l'image fait 10 pixels de large et que le code indique de colorier 12 pixels, on colorie toute la ligne plus les deux premiers pixels de la suivante.

⇝ Il faut une convention pour savoir par quelle couleur commencer, blanc ou noir.
On choisira de **commencer par le blanc**.
Ainsi , si une image commence par :

- 3 pixels blancs, le codage RLE commence par "3".
- 3 pixels noirs, le codage RLE va commencer par "0 3".
  On a en effet 0 pixels blancs suivis de 3 pixels noirs.

Voici les versions matricielles de quatre lettres, elles sont au format (5 pixels) × (7pixels), donner leur **codage RLE**.

a) Codage matriciel aplati de la lettre : 10001110111010110101100011000110001

Codage RLE : .....

b) Codage matriciel aplati de la lettre : 01110100011000111111100011000110001

Codage RLE : .....

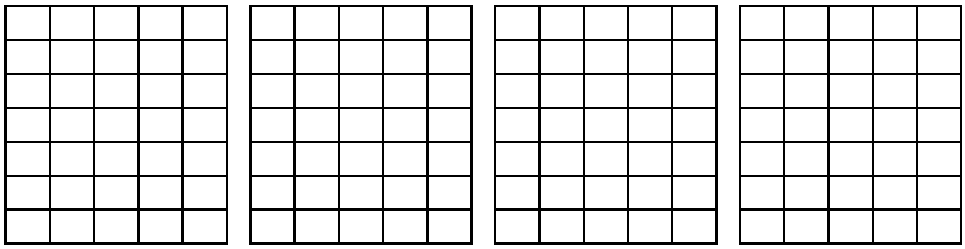
c) Codage matriciel aplati de la lettre : 11111001000010000100001000010000100

Codage RLE : .....

d) Codage matriciel aplati de la lettre : 10001100011000111111100011000110001

Codage RLE : .....

e) Vérifier les codages RLE à l'aide des grilles suivantes



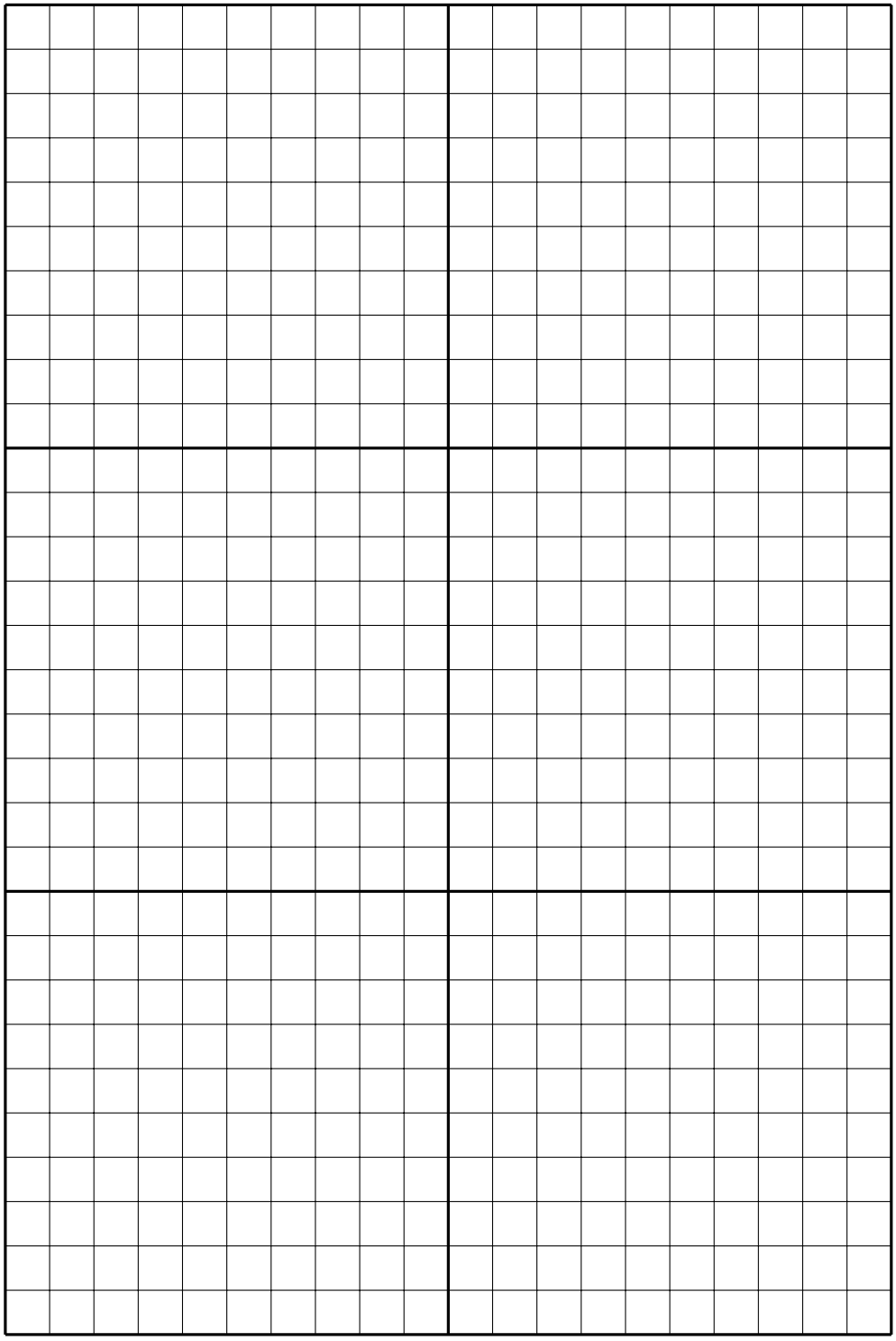
(i). RLE : **R**un **L**ength **E**ncoding

Activité de prolongement RLE - Débranchée

Nom : **Prénom :** **Classe :**

Reconstituer une grande image en accolant les 4 images ci-dessous, toutes de dimension 10 x 10 pixels.
Il faut mettre la 1 en haut à gauche, la 2 en haut à droite, la 3 en bas à gauche et la 4 en bas à droite.

- ⇝ code image 1 : 0 3 7 1 2 2 5 2 2 6 1 1 9 1 9 2 8 1 3 1 4 2 2 2 3 2 6 1 2 2 2 1 5
- ⇝ code image 2 : 11 7 2 2 3 3 7 2 6 2 7 2 6 1 2 1 6 1 2 2 9 1 5 1 2 2 5
- ⇝ code image 3 : 1 2 3 3 2 1 8 2 8 1 9 1 2 1 6 1 1 1 7 2 9 2 3 3 1 1 2 4 1 2 1 3 6
- ⇝ code image 4 : 2 2 9 1 4 2 3 2 3 2 4 1 1 3 2 1 2 1 1 3 3 1 1 3 6 2 7 2 6 2 2 1 6 3 6

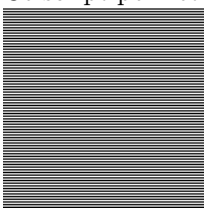


Fiche élève activité PBM - Rebranchée

Nom : **Prénom :** **Classe :**

Script

Le script suivant et ses commentaires donnent les manipulations nécessaires à la création d'images au format ***.pbm**, plus de détails sur les méthodes en annexe si nécessaire. (Click Me !)
Ce script permet d'obtenir l'image suivante :

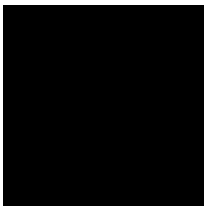
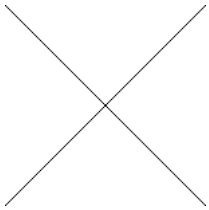
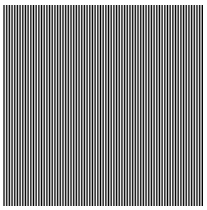
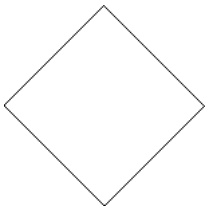


1.12 – Création d'images - Lignes horizontales noires et blanches alternées

```
##----- On définit les dimensions de l'image -----##
largeur = 150
hauteur = 150

##----- Ouverture/création des Fichiers -----##
f = open("images_pbm/Lignes_Horizontales.pbm", "w")

##------ Écriture dans le fichier ------##
# P1 en ligne 1 pour déclarer le format .pbm puis passage à la ligne
f.write("P1 \n")
# Largeur et hauteur de l'image, séparées par un espace#
f.write('{} {} \n'.format(largeur, hauteur))

##------ Instructions principales ------##
for ligne in range(hauteur):
        for colonne in range(largeur):
                # Lorsque le numéro de ligne est pair
                if ligne%2 == 0:
                        f.write('0')
                else:
                        f.write('1')
        f.write('\n')

##----- Fermeture des Fichiers -----##
f.close()
```

Énoncé/Questions

Seul ou en binôme, écrire un script python qui permet d'obtenir le fichier ***.pbm** pour au moins deux de ces images au choix. Prendre une largeur de 150 et une hauteur de 150.

Fiche élève activité RLE - Rebranchée

Nom : **Prénom :** **Classe :**

Énoncé/Questions

On pourra travailler à partir des fichiers de la séance précédente ou à partir de nouveaux.

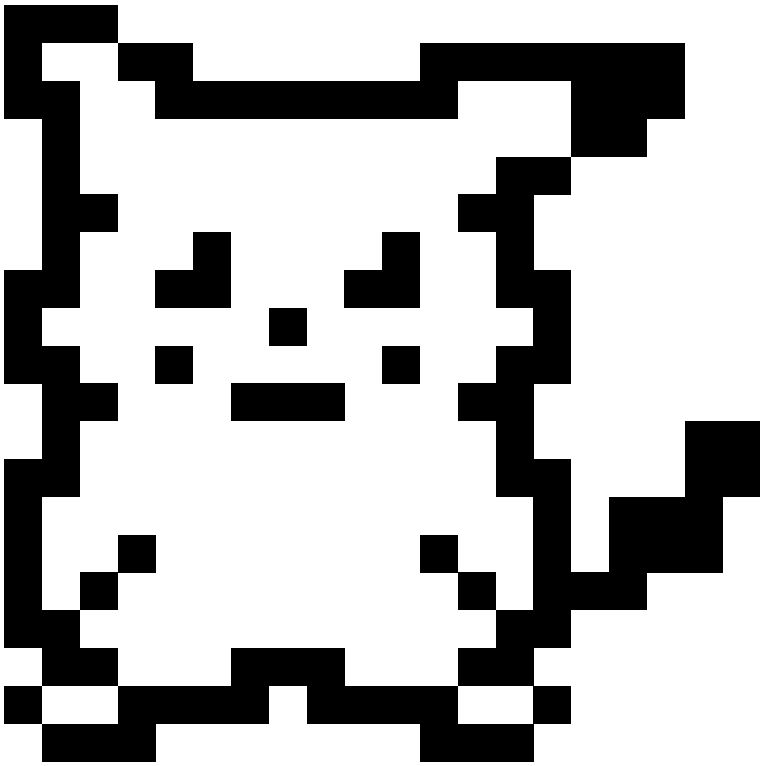
1/ **Compression PBM to RLE :** Écrire deux fonctions pour
   - **a)** récupérer les dimensions de l'image ***.pbm** et aplatir la matrice de 0 et de 1.
   - **b)** générer le **code RLE** tout en conservant les dimensions de l'image d'origine.

2/ **Décompression RLE to PBM :** Écrire deux fonctions pour
   - **a)** générer la matrice de 0 et de 1 aplatie à partir du **code RLE** et des dimensions de l'image.
   - **b)** générer le fichier ***.pbm**

Prolongement possible

Travailler à partir d'une image globale et créer les découpages.
Permettant de générer des exercices du type de Pikachu décrit dans la section précédente.
Il pourra être utile de créer les images nécessaires au format PBM ainsi que leur codage RLE à partir des fonctions réalisées.

global



1.7 Propositions de corrections

1.13 – Création d'images - Carré noir

```
##----- On définit les dimensions de l'image -----##
largeur = 150
hauteur = 150

##----- Ouverture/création des Fichiers -----##
f = open('images_pbm/Carre_Noir.pbm', 'w')

##------ Écriture dans le fichier ------##
# P1 en ligne 1 pour déclarer le format .pbm puis passage à la ligne
f.write("P1 \n")
# Largeur et hauteur de l'image, séparées par un espace#
f.write('{} {} \n'.format(largeur, hauteur))

##------ Instructions principales ------##
for ligne in range(hauteur):
        for colonne in range(largeur):
                f.write('1')
        f.write('\n')

##----- Fermeture des Fichiers -----##
f.close()
```

1.14 – Création d'images - Lignes verticales noires et blanches alternées

```
##----- On définit les dimensions de l'image -----##
largeur = 150
hauteur = 150

##----- Ouverture/création des Fichiers -----##
f = open('images_pbm/Lignes_Verticales.pbm', 'w')

##------ Écriture dans le fichier ------##
# P1 en ligne 1 pour déclarer le format .pbm puis passage à la ligne
f.write("P1 \n")
# Largeur et hauteur de l'image, séparées par un espace#
f.write('{} {} \n'.format(largeur, hauteur))

##------ Instructions principales ------##
for ligne in range(hauteur):
        for colonne in range(largeur):
                # On inscrit 0 lorsque colonne est pair, 1 sinon
                f.write(str(colonne%2))
        f.write('\n')

##----- Fermeture des Fichiers -----##
f.close()
```

1.15 – Création d'images - Croix

```
##----- On définit les dimensions de l'image -----##
largeur = 150
hauteur = 150

##----- Ouverture/création des Fichiers -----##
f = open('images_pbm/Croix.pbm', 'w')

##------ Écriture dans le fichier ------##
# P1 en ligne 1 pour déclarer le format .pbm puis passage à la ligne
f.write("P1 \n")
# Largeur et hauteur de l'image, séparées par un espace#
f.write('{} {} \n'.format(largeur, hauteur))

##------ Instructions principales ------##
for ligne in range(hauteur):
    for colonne in range(largeur):
        if (ligne == colonne) or (ligne+colonne == hauteur-1):
            f.write('1')
        else:
            f.write('0')
    f.write('\n')

##----- Fermeture des Fichiers -----##
f.close()
```

1.16 – Création d'images - Losange vide

```
##----- On définit les dimensions de l'image -----##
largeur = 150
hauteur = 150
demi = hauteur//2

##----- Ouverture/création des Fichiers -----##
f = open('images_pbm/Losange_Vide.pbm', 'w')

##------ Écriture dans le fichier ------##
# P1 en ligne 1 pour déclarer le format .pbm puis passage à la ligne
f.write("P1 \n")
# Largeur et hauteur de l'image, séparées par un espace#
f.write('{} {} \n'.format(largeur, hauteur))

##------ Instructions principales ------##
# Parcours de la 1ère moitié des lignes
for ligne in range(demi):
    for colonne in range(largeur):
        # 2ème diagonale du demi-carré et 1ère diagonale
        if (colonne+ligne == demi-1) or (colonne-ligne == demi-1):
            f.write('1 \n')
        else:
            f.write('0 \n')
# Parcours de la 2nde moitié des lignes
for ligne in range(demi):
    for colonne in range(largeur):
        # 1ère diagonale et 2nde diagonale
        if (colonne == ligne) or (colonne+ligne == largeur-1):
            f.write('1 \n')
        else:
            f.write('0 \n')

##----- Fermeture des Fichiers -----##
f.close()
```

1.17 – Aplatir la matrice pbm

```
##########################################################################
# Fonction pour aplatir la matrice de 0 et de 1 en une seule chaine
#
# Entrée : Le chemin vers le fichier *.pbm
# Sortie : Un tableau [largeur,hauteur,matrice de 0/1 aplatie]
#
# Le fichier *.pbm doit avoir la forme suivante
# P1 --> 1ere ligne nombre magique
# largeur hauteur --> deuxième ligne avec les dimensions
# les lignes suivantes avec les 0 et les 1 séparés par des espaces
##########################################################################
def pbm_to_clean_string(path_to_pbm):
    # On a besoin d'un tableau pour la sortie
    tab_out = []
    # On initialise la chaine de sortie
    str_out = ""
    # On ouvre le fichier en lecture
    file = open(path_to_pbm,'r')
    # On passe la première ligne qui contient le nombre magique
    file.readline()
    # On récupère les dimensions sur la seconde ligne
    second_line = file.readline().rstrip('\n').split(" ")
    tab_out.append(second_line[0])
    tab_out.append(second_line[1])
    for line in file:
        # On suprime le \n en fin de ligne
        line=line.rstrip('\n')
        # On suprime les blancs en début et fin de ligne
        line=line.strip()
        # On supprime tous les espaces de la chaine si necessaire
        line=line.replace(" ","")
        # On ajoute la ligne à la chaine de sortie
        str_out = str_out + line
    file.close()
    tab_out.append(str_out)
    return tab_out
```

1.18 – Générer le fichier pbm

```
##########################################################################
# Fonction pour générer un fichier *.pbm
#
# Entrées : largeur, hauteur, chaine de 0/1 , chemin vers le fichier pbm à écrire
# Sortie : un fichier *.pbm dont le format sera :
#
# P1 --> 1ere ligne nombre magique
# largeur hauteur --> deuxième ligne avec les dimensions
# les lignes suivantes avec les 0 et les 1 NON séparés par des espaces
##########################################################################
def clean_string_to_pbm(largeur,hauteur,chaine,path_to_pbm):
    # On ouvre un ficheir en écriture mode ajout pour la sortie
    file = open(path_to_pbm,'w')
    # On écrit le nombre magique
    file.write("P1\n")
    # On ecrit les dimensions de l'image
    file.write(str(largeur) + " " + str(hauteur)  + "\n")
    # On écrit toutes les autres lignes en coupant tous les "largeur" caractères
    k=0
    for i in range(len(chaine)//largeur):
        # On écrit les "largeur" caracteres de la chaine
        for j in range(k,k+largeur):
            file.write(chaine[j])
        # On ajoute le retour à la ligne
        file.write("\n")
        k+=largeur
    file.close()
```

```
##########################################################################
# Fonction pour donner le codage RLE de la chaine de 0 et de 1
#
# Entrée : largeur, hauteur et la chaine de 0 et de 1 issue de l'image pbm
# Sortie : Un tableau [largeur,hauteur,codage RLE de la chaine de 0 / 1]
#
# Par convention le premier nombre correspond au nombre de 0
# 0 représente le blanc
# Par exemple le code RLE de de 0001110011 vaut 3 3 2 2
# Celui de 1110001100 vaut 0 3 3 2 2
##########################################################################
def pbm_to_RLE(largeur,hauteur,chaine):
    # pour récupérer le code RLE de la chaine
    rle = ""
    # on récupère la taille de la chaine
    n = len(chaine)
    #On initialise le premier nombre du codage RLE à 0
    #si la chaine commence par 1
    if (chaine[0] == "1"):
        rle = rle + "0 "
    # compteur pour arrêter la boucle
    i = 0
    # tant que l'on n'est pas au nième caratctère de la chaine on répète
    while i < n- 1:
        # On compte le nombre d'occurence du caractère courant
        # on initialise donc le compteur à 1
        count = 1
        # Tant que le caractère courant n'est par le nième et que le caractère courant est égal au
        while (i < n - 1 and chaine[i] == chaine[i + 1]):
            # On incrémente le compteur
            count += 1
            # on incrémente l'indice du caractère
            i += 1
        # En sortie de boucle :
        # soit on est au (n-1)ième caractère et on a incrémenté n fois le compteur donc tous les c
        # soit le caractère suivant n'est pas égal au caractère courant et il reste à incrémenter
        # courant devienne le suivant.
        # dans le cas où tous les carcatères sont égaux, l'incrémentation ne gène pas.
        i += 1
        # On complète le code RLE
        rle=rle + str(count) + " "
    # On retourne le tableau avec largeur hauteur et le code RLE complet
    return [largeur,hauteur,rle]
```

```python
# Fonction pour decoder le codage RLE d'une chaine de 0 et de 1
# et écrire la matrice aplatie de l'image *.pbm
#
# Entrée : largeur, hauteur et le codage RLE de la chaine de 0 /1
# Sortie : un tableau avec largeur hauteur et la chaine de 0 et de 1
#
# Par convention le premier nombre correspond au nombre de 0
# 0 représente le blanc
# Par exemple le code RLE de de 0001110011 vaut 3 3 2 2
# Celui de 1110001100 vaut 0 3 3 2 2
#########################################################################
def RLE_decoding(largeur,hauteur,rle):
    # On initialise la chaine de sortie
    str_out=""
    # On récupère les valeurs du code RLE dans un tableau
    tab_RLE = rle.strip().split(" ")
    for i in range(len(tab_RLE)):
        for j in range(int(tab_RLE[i])):
            if i%2==0:
                str_out=str_out+"0"
            else:
                str_out=str_out+"1"
    return [largeur,hauteur,str_out]
```

MAY THE FORCE
BE WITH YOU

Source : IREM de Lyon, formation I.S.N 2018-2019 pour les scripts pbm » Création d'images Portable Bitmap
Source : Algo persos pour compression/decompression RLE