Cet article est rédigé par des élèves. Il peut comporter des oublis et imperfections,
autant que possible signalés par nos relecteurs dans les notes d'édition.

Motifs Labyrinthiques

Année 2023– 2024

Par : Anaïs LOISELEUR, Clara RINI, Clarisse DESEAU, Clément CARAYON, Idana FERNANDEZ, Ilhan SALHIOUI-LOPEZ, Léonard MEMETEAU, Marion LECOQ, Philippe AUBAILLY (3e et 2nde)

Établissement(s) : Lycée français Vincent van Gogh – La Haye

Enseignant·e(s) : Stéphane BERINGUE, Florence DECOOL

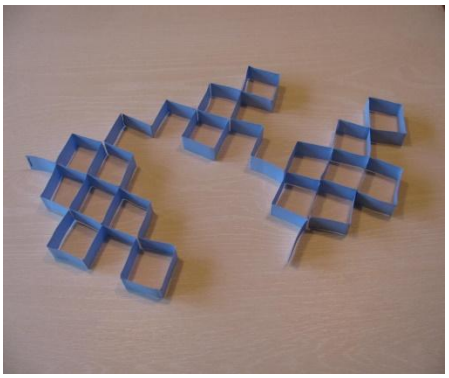
Chercheur : Pierre ALBERT, université d'Utrecht aux Pays-Bas.

Sommaire

I. Problématique

L'objet de cette expérience est de plier à plusieurs reprises des bandes de papier sans les déplier entre chaque pli.

Une fois tous les plis réalisés, il vous suffit de déplier la bande pour observer le motif qui en résulte.



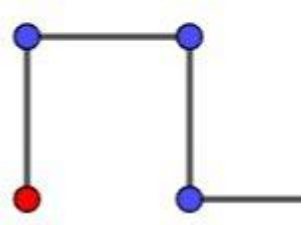
II. Présentation du sujet

Notre but est de pouvoir prévoir la forme d'une bande de papier après l'avoir pliée plusieurs fois par la moitié. Lors du dépliage de nos bandes de papier, il faut que les angles des plis mesurent 90°. Le sujet se nomme " motif labyrinthiques " car la forme finale ressemble à celle d'un labyrinthe.

Il faut respecter plusieurs consignes afin de plier proprement les bandes :

- Premièrement il faut respecter l'ordre de pliage grâce à notre code binaire :
  - Il faut garder le début de notre pliage dans la main gauche pour ne pas tourner la bande et perdre le fil du pliage.
  - Pour le (0) nous devons plier notre bande vers le bas et pour le (1) nous devons plier vers le haut.

Par exemple si nous voulons effectuer le motif 0,1,1 cela voudrait dire qu'il faudrait effectuer : " bas, haut, haut "



Motif du pli : 0 1 1 [1]

III. Notations et définitionns

On utilise une bande de papier marquée sur une des deux extrémités (la marque fait face au plafond).

On doit toujours garder cette marque dans notre main gauche.[2]

On traduit les plis par un code binaire :

1 <=> pli vers le haut

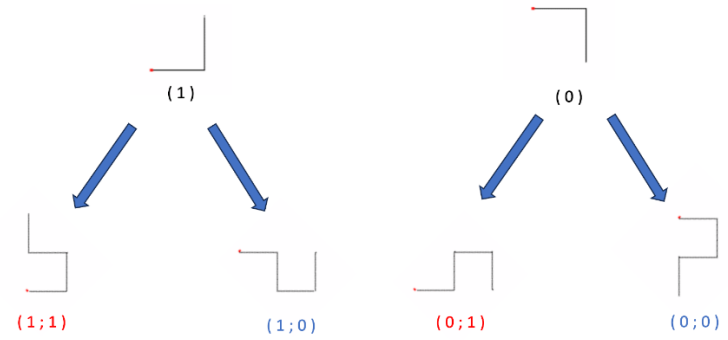
0 <=> pli vers le bas

On définit le verbe « **Retourner** » : par prendre une série et inverser la position de chaque caractère.
Par exemple, “110” devient “011”.

IV. Arbre des possibles

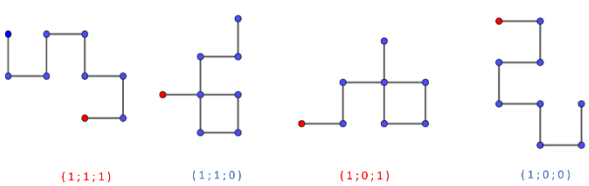
Avant de se lancer directement dans la création d'un programme pour explorer le sujet, nous avons tout d'abord consacré plusieurs semaines à l'élaboration d'un arbre des possibilités.
En effet, plutôt que de nous précipiter vers la complexité, nous avons tout simplement fait plusieurs tests pour réaliser notre arbre en examinant toutes les combinaisons possibles de plis "0" et "1".
Bien entendu, nous nous sommes limités à des arbres de trois plis, car au-delà, la taille de l'arbre devenait excessive (d'où l'idée de concevoir ensuite un code Python pour automatiser cette tâche).

Voici donc un aperçu des différentes combinaisons figurant dans notre arbre des possibilités :

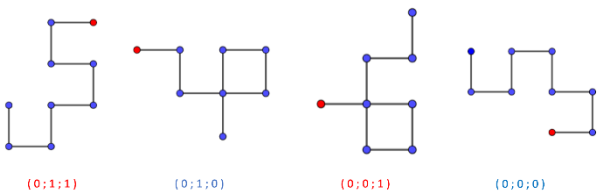


* Le point rouge représente la marque qui s'est trouvée à gauche lors des pliages. Nous la gardons à gauche pour mieux représenter l'arbre des possibles.

Arbre des possibles de trois pliages



Arbre des possibles de trois pliages



V- Figures symétriques

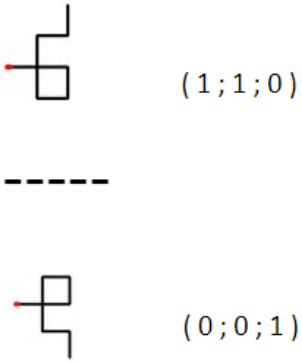
Après avoir passé du temps à nous familiariser avec le pliage des figures, nous avons pu rapidement observer une certaine symétrie entre les figures.

C'est-à-dire que pour n'importe quel pli, il y aura toujours un autre pli qui lui sera symétrique. En d'autres mots, lorsqu'on plie la figure avec n'importe(-s) quel(-s) pli(-s) (par exemple 1, 1, 0), ce pli possédera forcément un autre pli qui lui sera identique mais fait de façon opposée par rapport à un axe de symétrie (voir schéma ci-dessous).

On peut facilement trouver ce pli symétrique, il s'agit du pli inverse. C'est-à-dire que si on change chaque 1 en 0 et chaque 0 en 1 (1, 1, 0 devient alors 0, 0, 1) la figure qui sera formée en second sera symétrique par rapport à celle formée en premier (voir schéma ci-dessous). [3]

Pour rappel, on définit « Retourner » par :

- Prendre une série et inverser la position de chaque caractère
- 110 devient 011 [4]



VI. Modélisation mathématique du problème

Pour modéliser cette situation, on note n le nombre de plis effectués.

Nous pouvons donc définir :

- Le nombre de côtés se calcule par la suite géométrique (C_n) de raison 2 et de premier terme C_0 , c'est à dire $C_n = 2^n$

- La longueur des côtés par la suite géométrique(L_n) de raison $\frac{1}{2}$ et L_0 la longueur initiale de la bande de papier.
  c'est à dire $L_n = \left(\frac{1}{2}\right)^n \times L_0$

Il existe tout de même des limites physiques :

- la taille des bandes est finie

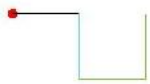
- il semble impossible de plier plus de sept fois une feuille de papier standard

VII. Pistes de réflexion

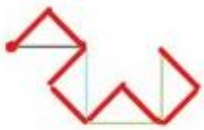
A. Prédiction par les triangles

Pli 1

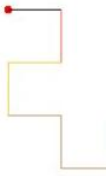
Prédiction du 1,0



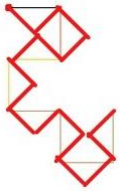
Pli 1,0



Prédiction du 1,0,0



Pli 1,0,0



Prédiction du 1,0,0,1

Pli 1,0,0,1

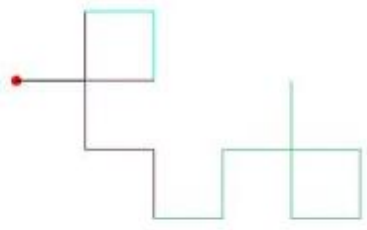
Notre première piste de réflexion pour pouvoir prédire le prochain pli a été celle de la prédiction par des triangles. [5]

Afin de pouvoir prédire le prochain pli avec des triangles il faut tout d'abord commencer avec un pli déjà existant. Ensuite on va construire des triangles sur chacun des côtés du pli. Attention, il doit obligatoirement n'y avoir qu'un triangle par coté.

Ensuite pour construire les triangles, nous allons nous aider de l'emplacement de la marque pour déterminer où sera le premier triangle. Le coté avec la marque sera toujours le premier qui aura un triangle construit car il va permettre de construire les prochains triangles. Nous avons 2 options pour le premier triangle, nous pouvons soit le construire du côté de la marque ou du côté opposé. Pour cela nous allons regarder quel sera le prochain pli, s'il s'agit d'un 0 alors le triangle sera du côté de la marque, s'il s'agit d'un 1 alors le triangle sera du côté opposé à la marque. Ensuite nous allons alterner entre coté marque et côté opposé pour déterminer les prochains triangles (par exemple si on construit le premier triangle du côté de la marque alors le second sera coté opposer puis le 3ème côté de la marque etc.).

Lorsque l'on a fini de construire tous les triangles, il ne nous reste plus qu'à prendre la figure construite par les triangles, attention il ne faut pas prendre en compte la figure déjà existante a la base mais seulement celle crée par les triangles (voir exemples situer à gauche). La figure qui sera alors former devra donc être la figure du prochain pli.

Nous avons donc réussi à prévoir la figure crée par le prochain pli. Cependant bien que cette méthode soit fiable et fonctionne à 100% ; elle trouve des limites dans son exécution qui devient rapidement compliqué à exécuter au fur et à mesure que le nombre de plis devient grand. Et qui devient physiquement



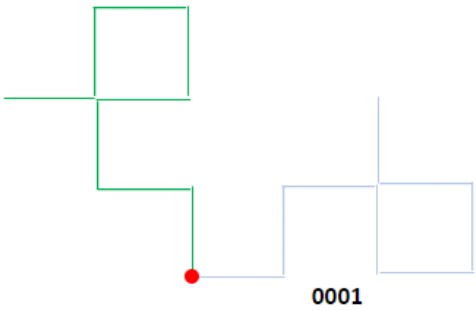
impossible au-dessus de 7 plis. De plus le script crée avec cette méthode ne peut prédire que 3 plis. C'est pour cela que nous nous sommes penchés sur d'autres pistes de réflexion.

B. Hypothèse de la rotation

Après avoir expérimenté avec nos plis de papier de nombreuses fois, nous avons remarqué plusieurs choses.

Premièrement, on a observé que chaque pli qu'on effectuait avait un centre, c'est-à-dire qu'il avait un certain motif répété équitablement 2 fois dans la totalité du pli.

Deuxièmement, on a également constaté que le motif répété une seconde fois était toujours placé après une rotation de 90° vers la droite (dans le sens des aiguilles d'une montre).



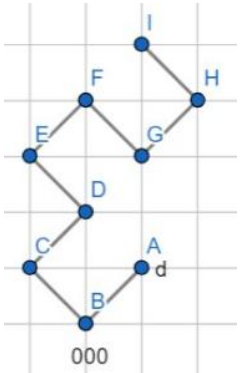
Ici, le motif du pli en vert est copié après une rotation de 90° vers la droite.

Ainsi, notre équipe a pensé à pouvoir prédire les plis de cette façon-là : en prenant un pli et lui ajouter le même motif après une rotation de 90° vers la (gauche/droite).

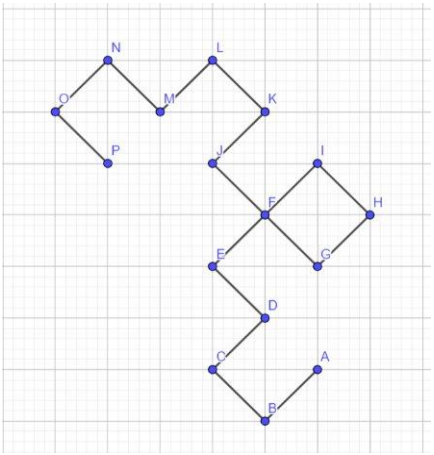
Nous avons découvert que cette hypothèse ne fonctionnait que partiellement. En effet, elle fonctionne si et seulement si le pli est homogène, c'est à dire qu'il ne comporte que des 0 ou des 1. Dès que les 0 et les 1 se mélangent, cette théorie n'est plus valable.

C'est ainsi que nous avons abandonné cette piste de réflexion pour trouver une autre façon de prédire des plis.

Pli 000



Pli 0000



Ici, on remarque que le motif de 000 est le même que la moitié du pli 0000.

C. La technique de dépliage

Il s'agit de modéliser un pli de type "0" qui représente une rotation de +90º, soit dans le sens des aiguilles d'une montre par rapport à la direction précédente. De manière similaire, un pli de type "1" correspond à une rotation de -90º, soit dans le sens contraire des aiguilles d'une montre.

Initialement, nous avons tenté de trouver une méthode pour représenter le dessin en suivant l'ordre des plis introduits par l'utilisateur. Cependant, chaque nouveau pli nécessitait des modifications sur ce qui avait déjà été dessiné.

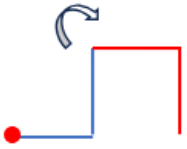
Cette approche impliquait de localiser le centre du bout de papier à chaque nouveau pli, de représenter le pli à nouveau, puis de redessiner l'intégralité du motif. Cette méthode complexifiait progressivement le dessin. Nous avons donc opté pour une nouvelle approche algorithmique.

Après avoir testé différentes options sur papier, nous avons conclu que la méthode la plus simple consistait à commencer depuis la fin des plis, c'est-à-dire en partant d'un papier complètement plié pour le déplier progressivement étape par étape.

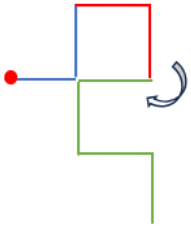
Cela se traduit par l'inversion de la série de plis fournie par l'utilisateur : si l'utilisateur saisit la série "0 0 1", la série inversée sera "1 0 0", qui sera donc prise en compte pour les étapes suivantes.

Prenons pour exemple trois plis : 0,0,1

Voici le dernier pli de la série, représenté par le chiffre 1 en bleu.
La marque rouge représente point de départ du programme. Le trait se déplace horizontalement vers la droite sur une distance préétablie en pixels (ici, 50 pixels). Ensuite, le programme réalise une rotation de - 90º conformément au pli "1", puis avance à nouveau de 50 pixels.



Voici l'avant-dernier pli, soit le pli 0 en rouge.
Le programme a repris la figure précédente en y ajoutant la rotation correspondante : + 90º



Pour le pli précédent, c'est-à-dire le pli 0 en vert, le protocole se répète avec une nouvelle rotation correspondante au pli 1 : + 90º

Afin de faire comprendre cela à un programme au langage python, il faudra premièrement créer une série qui enregistre les chiffres donnés par l'utilisateur, puis vérifier qu'elle comporte uniquement du code binaire. Ensuite, il s'agira de créer une série inversée afin de lire ses valeurs une par une. Après cela, il faudra créer une nouvelle série qui va suivre l'exemple de l'image suivante :

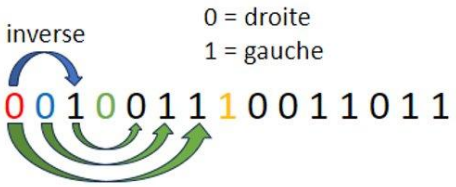
Pour expliquer cela à un programme en Python, il est d'abord nécessaire de créer une série qui enregistre les chiffres fournis par l'utilisateur, puis de vérifier qu'elle ne contient que des valeurs binaires. Ensuite, il faut inverser cette série pour lire ses valeurs une par une. Enfin, il convient de créer une nouvelle série suivant l'exemple de l'image suivante :

Exemple :

Série = 1 0 0 0

Série inversée = 0 0 0 1

Nouvelle série (pour le programme)= 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1



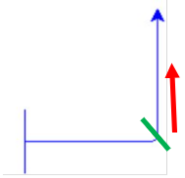
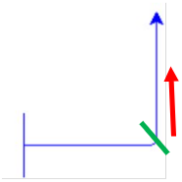
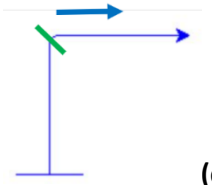
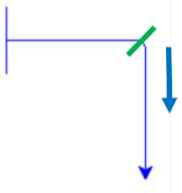
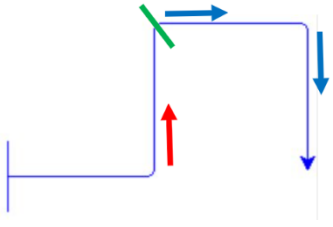
Comme nous pouvons le voir, le processus commence par ajouter le premier chiffre de la série inversée (0 rouge) à la nouvelle série. Ensuite, puisque aucun chiffre ne précède celui-ci, nous ajoutons le suivant (0 bleu). Après cela, nous ajoutons le chiffre qui précède le 0 bleu dans la nouvelle série (c'est-à-dire le 0 rouge) en l'inversant, ce qui donne un 1. Nous continuons ainsi en ajoutant tous les chiffres de la série inversée.

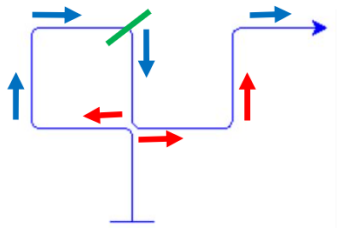
Une fois ces étapes terminées, le programme peut lire chaque valeur de la nouvelle série. Lorsqu'il rencontre un "0", il effectue une rotation vers la droite, et lorsqu'il rencontre un "1", il effectue une rotation vers la gauche, avançant d'un nombre défini de pixels à chaque étape. Cela permet de dessiner la même figure mais avec un angle différent.

Le Code python du programme se trouve dans la rubrique Annexes.

D. À gauche ou à droite

Nous avons aussi pensé à trouver une solution à ce problème en utilisant une représentation visuelle : on suit le chemin représenté par le motif labyrinthique.

<table>
<tr><td></td><td> 
</td><td>Légende</td></tr>
<tr><td rowspan="1">Etape 1</td><td>Pour “1” :
  </td><td>Premièrement, nous devons regarder le dernier chiffre de la série demandée.
Nous visualiserons alors le problème en allant à droite si le dernier chiffre est un “0” et à gauche s’il est un “1”.</td></tr>
<tr><td rowspan="4">Etape 2</td><td>Pour “1” :
 
(gauche)</td><td>Deuxièmement, lorsque l’on continue la série, en allant de droite à gauche, il y a trois sous-étapes à suivre :
Réutiliser les instructions de déplacement trouvées précédemment.</td></tr>
<tr><td>+
Plus “0” :
 
(droite)</td><td>Ajouter le milieu de cette nouvelle série que nous sommes en train de construire. En suivant la méthode précédente ; si le chiffre de notre étape est un “0” nous allons à “droite” et si c’est “1” nous allons à “gauche”, nous déterminons l’instruction centrale de notre code qui représente le pli du milieu.</td></tr>
<tr><td>+
La figure retournée :
  </td><td>Finalement, pour prévoir la partie droite de notre labyrinthe, il faut aussi prendre la suite d’instructions trouvée précédemment et l’inverser, c’est à dire que pour chacune des instructions, aller à “gauche” signifie maintenant aller à “droite”, et réciproquement. Après cela, il suffit de “retourner” la série.
Pour rappel, retourner signifie prendre une série et inverser la position de chaque caractère.</td></tr>
<tr><td>  </td><td>Le résultat pour “01”</td></tr>
</table>

Etape 3	 **En suivant le même format, pour “001”:**	Troisièmement, en allant de droite vers la gauche et pour tous les chiffres à partir du chiffre pénultième, il suffit de répéter l’étape 2.

Ainsi, pour une série comme “1011”, nous obtiendrons ; “ggddgddgggdggdd” (avec “g” représentant la gauche et “d” la droite). Après, cette série d’instructions pourra être donnée à notre robot “turtle” sur Python, qui interprètera ces lettres comme un angle de +90° pour aller à droite ou de -90° pour tourner à gauche.

Le Code python du programme se trouve dans la rubrique Annexes.

VIII. Conclusion

Vous pourriez très bien vous demander à quoi notre projet pourrait bien servir ?

Ce projet permet d’appréhender le principe de la fractale, car notre projet se rapproche de celle-ci. Une fractale est un objet géométrique qui représente des formes découpées, fragmentaires, laissant apparaître des motifs similaires à des échelles d'observation de plus en plus fines.

Prenons l’exemple de l'éponge de Sierpinsky :

Modèle d’une éponge de Sierpinsky en 3D.

Autrement dit, on peut reconnaître un objet fractal s'il a une forme plus ou moins fragmentée, et si on retrouve cette forme en changeant d'échelle. Notre projet est peut-être comparé à ce principe, c'est-à-dire que chaque pli que l'on fera mènera à un motif complexe et répétitif qui pourrait théoriquement s'étendre à l'infini.

NOTES DE L'ÉDITION

[1] Cette figure correspond au motif (0 ; 0), et non pas à (0 ; 1 ; 1).

[2] La description du jeu devrait être plus claire. Il faudrait ajouter que, lorsque la bande de papier est dépliée pour former le labyrinthe, la face marquée doit être orientée vers le joueur et la première partie doit aller de gauche à droite.

Il existe en effet plusieurs façons d'orienter le labyrinthe et il faudrait en fixer une afin d'avoir un traitement plus rigoureux du sujet. Par exemple, les motifs (1 ; 0 ; 0) et (0 ; 1 ; 1) considérés plus loin sont symétriques par rapport à un axe vertical dans la figure, tandis qu'ils sont symétriques par rapport à un axe horizontal si l'on suit la nouvelle règle.

[3] Ce fait est assez évident. Il n'aurait pas été difficile d'en donner aussi une explication formelle. On peut aussi observer que, si l'on note les pliages par 1 et –1 (au lieu de 1 et 0), on obtient la figure symétrique simplement en multipliant le motif par –1.

[4] Y a-t-il un rapport entre un labyrinthe et ce que l'on obtient après l'opération de "Retourner" ? Les figures dans l'arbre des possibilités semblent suggérer qu'il n'y en a pas. On peut donc se demander pourquoi cette opération a été envisagée.

[5] Là encore, quelque explication aurait été opportune. Construire un triangle sur les côtés du labyrinthe correspond évidemment à un pliage : chaque côté est divisé en deux parties. Il faut noter que les figures n'indiquent pas avec précision la longueur des côtés. Dans les figures en rouge, les côtés ont une longueur de $\frac{L}{\sqrt{2}}$, où L est la longueur du côté de départ, tandis qu'en réalité les nouveaux côtés ont une longueur de $\frac{L}{2}$.

IX. Annexe

Code Python de la partie C : La technique de dépliage

```
from turtle import * #on va utiliser la librérie de la tortue pour dessiner

from random import randint #et le random pour des valeurs au hasard


def zero(): #on définit le mouvement de la tortue lorsque la valeur est 0
    right(90) #la tortue tourne 90º vers la droite
    forward(50) #elle avance de 50 pixels


def un(): #de même lorsque la valeur est 1
    left(90) #la tortue tourne 90º vers la gauche
    forward(50) # elle avance de 50 pixels


serie = [] #il faut créer une série qu'on demandera à l'utilisateur plus tard

incorrecte = True #permet de créer le while loop


while incorrecte: # = pendant que ça soit Vrai
    test = input("Saisissez votre série en séparant chaque chiffre (0 ou 1) par un
espace \n") #on demande à l'utilisateur la série de son choix
    serie = test.split(' ') #on sépare chaque valeur de la série pour les
reconnaître individuellement

    for h in range(len(serie)): #si l'utilisateur ajoute des espaces en trop, le
programme les enlève

        if "" in serie:
            serie.remove("")


    for i in range(len(serie)): # = pour chaque valeur de la série (elle se répète
autant de fois que sa longueur)

        if serie[i] != "0" and serie[i]!="1": #on vérifie que c'est du code binaire
            incorrecte = True #la valeur n'est pas autorisée
            print("Veuillez mettre seulement du code binaire (0 - 1) \n") #le
message qui apparaît lorsque l'utilisateur insère autre chose que du code binaire

            break #on sort du "for" car une valeur non autoriée est saisie. On
redemande de saisir la série.

        else: # = si la valeur saisie est binaire

            incorrecte = False #la valeur saisie est autorisée. On sort du while une
fois toutes les valeurs sont vérifiées.


serie_inv = [] #on va créer une autre série pour l'invertir et la programmer de la
fin vers le début

serie_inv = serie[::-1] #on inverse la série
```

```
#on va faire un cercle pour définir où se trouve la marque du papier

shape("turtle") #cela change l'apparence de la tortue (on peut l'enlever pour
laisser une flèche)

penup() #cela est pour éviter de dessiner quand on place la tortue

goto(0,-3) #déplace la tortue à ces coordonnées

pendown() #replace le stylo pour dessiner

begin_fill() #active la manière de remplire le cercle

color("red") #change la couleur pour le distinguer, et la tortue sera verte

circle(3) #crée le cercle avec son rayon

end_fill() #complète le cercle

color("black", "green") #remet la couleur en noir. La tortue reste verte

penup() #relève le stylo

goto(0,0) #replace la tortue à sa position initiale

pendown() #remet le stylo pour dessiner

colormode(255) #permet que les valeurs de la couleur arrivent à 255



forward(50) #elle avance de 50 pixels

dessin = [] #cette nouvelle série va sauvegarder les valeurs dessinées précédemment

for i in range(len(serie)): # = pour chaque valeur de la série donnée par
l'utilisateur (pour sa longueur et non son ordre)

    pencolor(randint(0, 255), randint(0, 255), randint(0, 255)) #pour chaque
répétition on change la couleur au hasard

    if serie_inv[i] == "0": #cela se répétera pour chaque valeur
        zero() #si dans la série inversée (la dernière valeur de la série) c'est 0,
on dessine la fonction zéro (qui est définie au début du programme)

    else: #si la valeur de la série est égale à 1
        un() #de même si c'est un 1

    dessin_temp = dessin[::-1] #on va inverser cette série dans une série temporelle
pour reprendre chaque valeur à l'envers comme si on faisait le chemin vers le début

    dessin.append(serie_inv[i]) #on sauvegarde alors la valeur dessinée dans dessin

    for j in range(len(dessin_temp)): #on reprend chaque valeur déjà dessinée

        if dessin_temp[j] == "0": #ici on va les inverser car il s'agit de symétrie
centrale
            un() #on dessine le contraire de ce qu'il y a écrit
```

```
                dessin.append("1") #on le rajoute à la série pour le sauvegarder

            else:
                zero() #on fait la même chose lorsque la valeur est 1
                dessin.append("0")



    hideturtle() #cache le curseur

    exitonclick() #cela permet au dessin de rester dessiné et de ne pas disparaître une
    fois le programme finit
```

Code Python de la partie D : À gauche ou à droite

```
import turtle as t
code = input("Donnez une chaîne caractères constitué de 0 et de 1\n")
motif = ""
if code[-1] == "1": #on regarde le dernier plie et 1 = "g"
    motif += "g"
elif code[-1] == "0":             #et 0 = "d"
    motif += "d"
else:
    print("0 ou 1")

def inverse(motif): #creation de la fonction qui inverse les valeurs
    inverse = "" #initialisation de la variable qui récupère l'inverse
    for i in motif: #inverse, si c'est "g" ca devient "d" et inversement pour tout le
motif initiale
        if i == "g":
            inverse += "d"
        elif i == "d":
            inverse += "g"
    return inverse

for i in reversed(code[:-1]): #on passe à travers toutes les valeurs du code de base
    if i == "1": #avec 1 on prend le motif de base, on met "g" et l'inverse des nombre
et le reverse
        motif3 = inverse(motif)
        motif += "g"
        motif += motif3[::-1]
    elif i == "0": #avec 0 on prend le motif de base, on met "d" et l'inverse et le
reverse des nombres
        motif3 = inverse(motif)
        motif += "d"
        motif += motif3[::-1]

t.tracer(100)
taille = 100
if len(motif)>6:
```

```
    taille = 60
if len(motif)>14:
    taille = 50
if len(motif)>28:
    taille = 30
if len(motif)>48:
    taille = 10
t.color("blue")
t.left(90)
t.fd(taille/4)
t.backward(taille/2)
t.fd(taille/4)
t.right(90)
t.fd(taille)
for i in motif:
    if i == "g":
        t.circle(5,90)
        t.fd(taille)
    elif i == "d":
        t.circle(-5,90)
        t.fd(taille)
t.update()
t.mainloop()
```