# TP1 MV55 Piana Victor

## SOMMAIRE

# Question 1 : Fonctions de Rotation et de Symétrie

## 1a — Fonction de Rotation

La première fonction à réaliser est la fonction **rote** qui effectue une rotation d'un point m(x,y) autour de l'origine, par un angle θ. Cette fonction doit fournir les nouvelles coordonnées (x′,y′) du point après la rotation.

Voici l'illustration de la rotation par un angle θ :

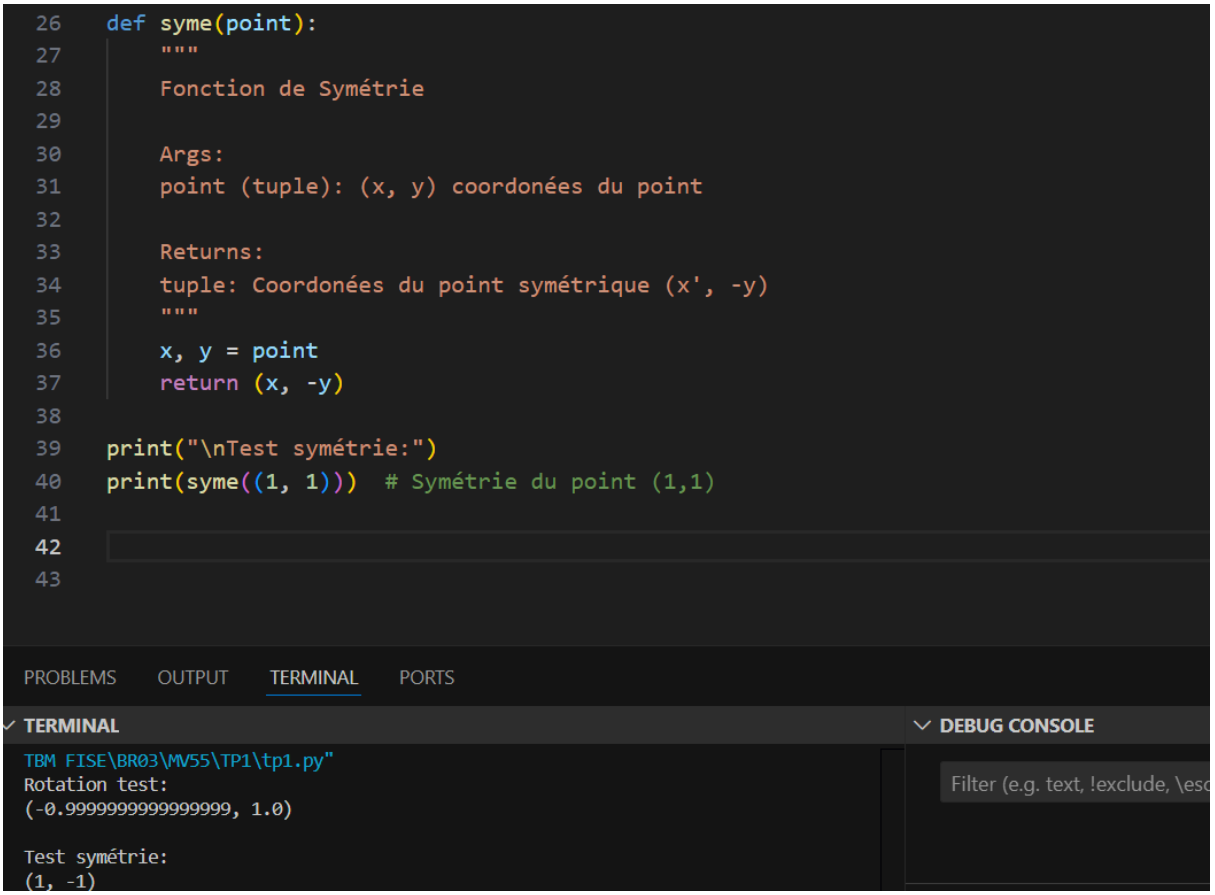
```python
def rote(theta, point):
    """
    Fonction de rotation

    Args:
    theta (float): Angle en radian
    point (tuple): les coordonées (x,y) du point

    Returns:
    tuple: coordonnées après rotation : (x', y')
    """
    x, y = point
    x_rotated = x * math.cos(theta) - y * math.sin(theta)
    y_rotated = x * math.sin(theta) + y * math.cos(theta)
    return (x_rotated, y_rotated)

print("Rotation test:")
print(rote(math.pi/2, (1, 1)))  # Rotation de 90 degrée par rapport au point (1,1)
```

```
PROBLEMS  OUTPUT  TERMINAL  PORTS
TERMINAL                                          DEBUG CONSOLE
(-0.9999999999999999, 1.0)
PS C:\Users\victo\Desktop\UTBM FISE\BR03\MV55> python -u "c:\Users\victo\Desktop\UTBM FISE\BR03\MV55\TP1\tp1.py"
Rotation test:
(-0.9999999999999999, 1.0)
```

## 1b — Fonction de Symétrie

La fonction **syme** applique une symétrie orthogonale par rapport à l'axe des abscisses (axe x). Cette symétrie transforme un point m(x,y) en un point symétrique m′(x,−y).

Cela signifie que seules les coordonnées y sont inversées, tandis que x reste inchangé.




# Question 2 : Etude du groupe diédral D8 pour n=4

## 2a — Traduction entre base 4 et base décimale

On commence par considérer l'ensemble des transformations du plan D8, avec n=4 et rθ=rπ/2.

Pour manipuler cet ensemble, il est nécessaire de traduire entre les représentations en base 4 et les entiers décimaux. Ainsi, nous avons défini deux fonctions :

- `trad_dec_qu(k)` : Cette fonction permet de convertir un nombre décimal k en une représentation en base 4 sous la forme d'un couple (α,β), où α et β sont les chiffres de la base 4 représentant k.

- `trad_qu_dec((α,β))` : Cette fonction permet de convertir une représentation en base 4 sous la forme d'un couple (α,β) en un entier décimal k.

```python
def trad_dec_qu(k):
    """
    Traduit un nombre décimal en représentation en base 4

    Arguments :
    k (int) : Nombre décimal (de 0 à 7)

    Retourne :
    tuple : Représentation en base 4 (a, B)
    """
    alpha = k // 4
    beta = k % 4
    return (alpha, beta)

def trad_qu_dec(base4_tuple):
    """
    Traduit une représentation en base 4 en nombre décimal

    Arguments :
    base4_tuple (tuple) : Représentation en base 4 (a, B)

    Retourne :
    int : Nombre décimal
    """
    alpha, beta = base4_tuple
    return 4 * alpha + beta
```

```python
print("Conversion décimal vers base 4:")
print(trad_dec_qu(5))  # Example avec 5
print("\nConversion base 4 vers décimal")
print(trad_qu_dec((1, 1)))  # Example: base 4 (1,1)
```

```
PROBLEMS  OUTPUT  TERMINAL  PORTS
TERMINAL                                  DEBUG CONSOLE
PS C:\Users\victo\Desktop\UTBM FISE\BR03\MV55> python -u "c:\Users\victo\Desktop\U

Conversion décimal vers base 4:
(1, 1)

Conversion base 4 vers décimal
5
```

- `trad_dec_qu(k)` :

  Cette fonction prend un entier k (compris entre 0 et 7), et le convertit en base 4. Le nombre k est décomposé en deux parties :

  - α, qui est le quotient de la division de k par 4.
  - β, qui est le reste de cette division.
  - `trad_qu_dec(α,β)` :

    Cette fonction prend un couple (α,β) représentant un nombre en base 4, et le reconvertit en un entier décimal. La formule utilisée est :

    $k=4\cdot\alpha+\beta$

## 2b — Composition dans le groupe D8

Dans cette partie, il s'agit de construire la table de composition `tab(i, j)` pour tous les indices `i` et `j` entre 0 et 7, qui correspond à la loi de composition dans le groupe diédral D8. On utilise les notations et les résultats vus en travaux dirigés.

On rappelle que les éléments de D8 sont codés par des entiers de 0 à 7, où chaque transformation `σ_k` correspond à une rotation ou une symétrie. Ces éléments peuvent être représentés par un couple `(alpha, beta)` en base 4, où :

- `alpha = 0` correspond à une rotation `r_beta`
- `alpha = 1` correspond à une symétrie `s o r_beta`

La fonction `compose(i, j)` permet de calculer la composition `σ_i o σ_j` et de retrouver l'indice `k` de la transformation résultante, c'est-à-dire `σ_k = σ_i o σ_j`.

---

**Code de la fonction compose**

```
    # Convertir les indices en paires (α, β)
    alpha_i, beta_i = trad_dec_qu(i)
    alpha_j, beta_j = trad_dec_qu(j)

    # Règles de composition du groupe diédral
    if alpha_i == 0 and alpha_j == 0:
        # Rotation ∘ Rotation = Rotation
        alpha_result = 0
        beta_result = (beta_i + beta_j) % 4
    elif alpha_i == 0 and alpha_j == 1:
        # Rotation ∘ Symétrie = Symétrie
        alpha_result = 1
        beta_result = (beta_j - beta_i) % 4
    elif alpha_i == 1 and alpha_j == 0:
        # Symétrie ∘ Rotation = Symétrie
        alpha_result = 1
        beta_result = (beta_i + beta_j) % 4
    else:  # alpha_i == 1 and alpha_j == 1
        # Symétrie ∘ Symétrie = Rotation
        alpha_result = 0
        beta_result = (beta_j - beta_i) % 4

    # Convertir le résultat en indice k
    k = trad_qu_dec((alpha_result, beta_result))
    return k
```

---

**Exemples de tests :**


Ces résultats confirment que la fonction respecte les règles de composition du groupe D8.

---

## 2c — Inverses dans le groupe D8 et vérification groupe abélien

Dans cette partie, l'objectif est de déterminer l'inverse de chaque élément du groupe D8. Un élément `σ_i` doit avoir un inverse `σ_j` tel que `σ_i o σ_j = σ_0`, où `σ_0` est l'élément neutre (la rotation identique).

Les inverses des éléments du groupe D8 peuvent être déterminés de la manière suivante, à partir de la représentation de chaque transformation en base 4 `(alpha, beta)` :

- Si `alpha = 0` (rotation), l'inverse de `r^beta` est `r^{(4 - beta)}` (c'est-à-dire `r^(-beta mod 4)`).
- Si `alpha = 1` (symétrie), l'inverse de `s o r^beta` est `s o r^(-beta mod 4)` (puisque `(s o r^beta)^2 = id`).

J'ai implémenté une fonction `inverse(i)` qui calcule l'inverse d'une transformation `σ_i` dans le groupe D8. Le code de cette fonction est le suivant :

**Code de la fonction inverse :**

```python
def inverse(i):
    """
    Trouve l'inverse de la transformation o_i dans le groupe D₈

    Args:
        i (int): Indice de la transformation (0 à 7)

    Returns:
        int: Indice j de l'inverse de o_i
    """
    alpha, beta = trad_dec_qu(i)

    # Pour les rotations (alpha = 0): l'inverse de r^k est r^(4-k) ou r^(-k mod 4)
    if alpha == 0:
        inverse_beta = (-beta) % 4
        return trad_qu_dec((0, inverse_beta))

    # Pour les symétries (alpha = 1): l'inverse de s∘r^k est s∘r^(-k mod 4)
    # Car (s∘r^k)∘(s∘r^k) = id
    else:
        inverse_beta = (-beta) % 4
        return trad_qu_dec((1, inverse_beta))
```

**Explication de la fonction :**

- La fonction prend un indice `i` (de 0 à 7) représentant une transformation dans D8.
- Selon la valeur de `alpha` dans la représentation `(alpha, beta)`, l'inverse est calculé en ajustant l'angle de rotation ou en inversant la symétrie.
- Le résultat est converti à nouveau en indice pour obtenir la transformation inverse.

Les résultats pour les inverses de chaque élément de D8 sont les suivants :

**Affichage dans le terminal :**

```python
print("Inverses dans D₈:")
for i in range(8):
    inv = inverse(i)
    print(f"L'inverse de σ_{i} est σ_{inv}")
```

PROBLEMS 3 OUTPUT TERMINAL PORTS

TERMINAL

```
r¹ ∘ r² = 3
s ∘ r¹ = 5
Inverses dans D₈:
L'inverse de σ_0 est σ_0
L'inverse de σ_1 est σ_3
L'inverse de σ_2 est σ_2
L'inverse de σ_3 est σ_1
L'inverse de σ_4 est σ_4
L'inverse de σ_5 est σ_7
L'inverse de σ_6 est σ_6
L'inverse de σ_7 est σ_5
```

Pour déterminer si le groupe D8 est abélien ou non, nous devons vérifier si la composition des transformations est commutative. En d'autres termes, il faut vérifier si pour deux transformations `σ_i` et `σ_j`, on a bien `σ_i o σ_j = σ_j o σ_i`.

J'ai écrit une fonction `test_abelien()` qui effectue cette vérification en prenant deux transformations de D8 et en comparant leur composition dans les deux sens. Voici le principe :

1. On choisit deux transformations `σ_i` et `σ_j` (par exemple, `σ_1` et `σ_4`).
2. On calcule leur composition dans les deux sens (`σ_i o σ_j` et `σ_j o σ_i`).
3. Si les deux résultats sont différents, cela signifie que le groupe n'est pas abélien.

**Code de la fonction test_abelien :**

```python
def test_abelien():
    """Teste si D8 est abélien ou non"""
    # Prenons deux transformations
    i, j = 1, 4  # r¹ et s

    # Calculons leur composition dans les deux sens
    comp_ij = compose(i, j)
    comp_ji = compose(j, i)

    print(f"σ_{i} ∘ σ_{j} = σ_{comp_ij}")
    print(f"σ_{j} ∘ σ_{i} = σ_{comp_ji}")

    if comp_ij != comp_ji:
        print("D8 n'est pas abélien car la composition n'est pas commutative")
    else:
        print("Ces éléments commutent")

test_abelien()
```

```
PROBLEMS 3  OUTPUT  TERMINAL  PORTS
TERMINAL
L'inverse de σ_6 est σ_6
L'inverse de σ_7 est σ_5
σ_1 ∘ σ_4 = σ_7
σ_4 ∘ σ_1 = σ_5
D8 n'est pas abélien car la composition n'est pas commutative
PS C:\Users\victo\Desktop\UTBM FISE\BR03\MV55>
DEBUG CONSOLE
Filter (e.g. text, !exclu
```

**Résultat dans le terminal :**

Cela montre que D8 n'est pas abélien, car la composition des transformations n'est pas commutative.

---

## 2d — Table de composition pour le groupe diédral $D_8$

Dans cette partie, nous avons construit la table de composition du groupe diédral $D_8$, qui indique les résultats de la composition de chaque transformation avec toutes les autres.

**Code pour la construction de la table du groupe diédral D8:**

```python
def construire_table_groupe():
    """Construit et affiche la table du groupe diédral D8"""
    n = 4  # n=4 pour D8
    taille = 2*n

    # Initialiser la table
    table = [[0 for _ in range(taille)] for _ in range(taille)]

    # Remplir la table
    for i in range(taille):
        for j in range(taille):
            table[i][j] = compose(i, j)

    # Afficher la table
    print("Table du groupe diédral D8:")
    print("    | " + " ".join(f"σ_{j}" for j in range(taille)))
    print("---+" + "-"*(3*taille))

    for i in range(taille):
        ligne = f"σ_{i} | "
        ligne += " ".join(f"{table[i][j]:2d}" for j in range(taille))
        print(ligne)

    return table

table_D8 = construire_table_groupe()
```

**Explication :**

- La fonction `construire_table_groupe()` initialise une table vide de taille 8×8 pour le groupe D8.
- Ensuite, pour chaque paire d'éléments du groupe (σ_i et σ_j), la fonction calcule la composition à l'aide de la fonction `compose(i, j)`.
- Enfin, la table est affichée dans un format lisible.

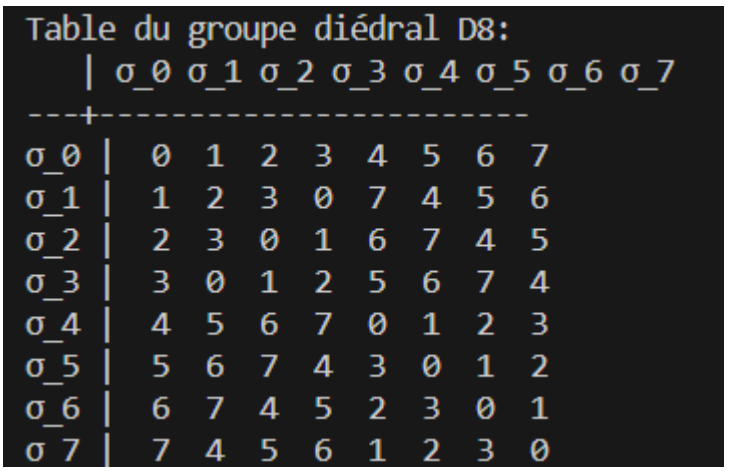


Table du groupe diédral D8:

| | σ_0 | σ_1 | σ_2 | σ_3 | σ_4 | σ_5 | σ_6 | σ_7 |
|---|---|---|---|---|---|---|---|---|
| σ_0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| σ_1 | 1 | 2 | 3 | 0 | 7 | 4 | 5 | 6 |
| σ_2 | 2 | 3 | 0 | 1 | 6 | 7 | 4 | 5 |
| σ_3 | 3 | 0 | 1 | 2 | 5 | 6 | 7 | 4 |
| σ_4 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |
| σ_5 | 5 | 6 | 7 | 4 | 3 | 0 | 1 | 2 |
| σ_6 | 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 |
| σ_7 | 7 | 4 | 5 | 6 | 1 | 2 | 3 | 0 |

Caque ligne et chaque colonne correspond à l'indice d'une transformation du groupe D8, et les cases indiquent l'indice du résultat de la composition des transformations.

## 2e — Sous-groupes de $D_8$ et diagramme de Hasse

Pour cette partie, j'ai conçu plusieurs fonctions Python pour étudier et visualiser les sous-groupes du groupe diédral $D_8$. L'objectif était de générer **automatiquement** tous les sous-groupes de $D_8$ et d'en afficher la structure d'inclusion via un diagramme de Hasse.

---

**Fonctions principales utilisées:**

- `sous_groupe_engendre()`

  Cette fonction reçoit une liste de générateurs (indices des éléments du groupe) et construit **le plus petit sous-groupe** contenant ces éléments, en fermant l'ensemble sous la composition.

- `trouver_tous_sous_groupes()`

  Elle explore toutes les combinaisons de 1 à 3 générateurs possibles parmi les éléments de $D_8$ pour construire **l'ensemble des sous-groupes distincts**. Chaque sous-groupe est ajouté seulement s'il est nouveau (non déjà généré).

- `diagramme_hasse()`

  Cette fonction affiche dans le terminal les relations d'inclusion directe entre les sous-groupes (sans sous-groupe intermédiaire). Elle respecte l'ordre de

construction et détecte les inclusions strictes.

- `dessiner_hasse_vertical()`

  Grâce à networkx et matplotlib, elle construit une **visualisation graphique orientée** du diagramme de Hasse. Chaque nœud représente un sous-groupe, et une flèche pointe vers un sous-groupe contenant strictement l'autre.

---

**1. Liste des sous-groupes de $D_8$ :**

Les sous-groupes de $D_8$ sont les ensembles engendrés par les éléments du groupe et tous leurs sous-ensembles, fermés sous la composition. Voici la liste des sous-groupes de $D_8$, classée par ordre croissant de taille :

```
Liste des sous-groupes de D8:
Sous-groupe 0: [0]
Sous-groupe 1: [0, 2]
Sous-groupe 2: [0, 4]
Sous-groupe 3: [0, 5]
Sous-groupe 4: [0, 6]
Sous-groupe 5: [0, 7]
Sous-groupe 6: [0, 1, 2, 3]
Sous-groupe 7: [0, 2, 4, 6]
Sous-groupe 8: [0, 2, 5, 7]
Sous-groupe 9: [0, 1, 2, 3, 4, 5, 6, 7]
```

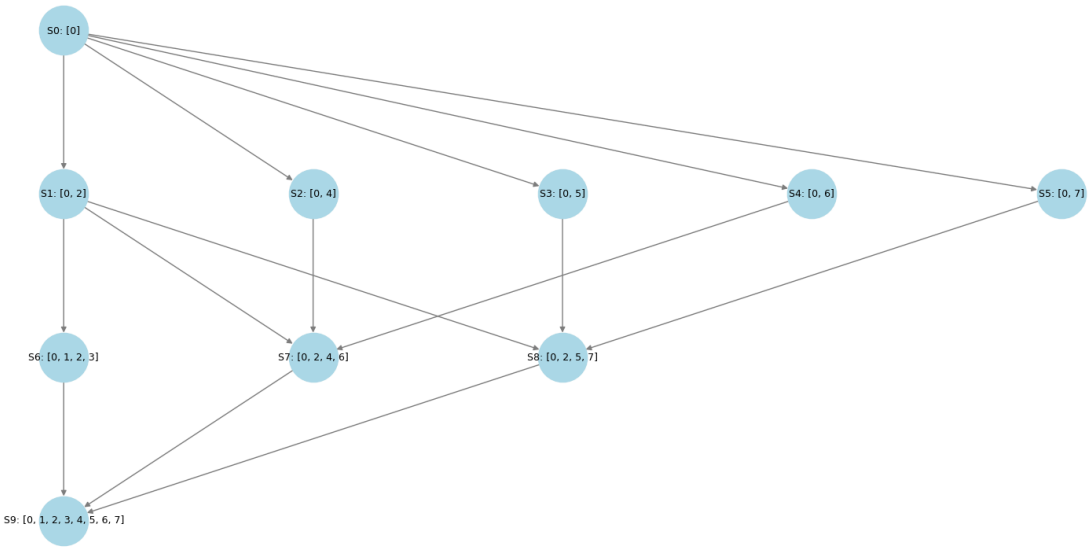
**2. Diagramme de Hasse des sous-groupes de $D_8$ :**

Voici le diagramme de Hasse des sous-groupes de $D_8$, où chaque sous-groupe est inclus dans un autre selon la relation d'inclusion directe. Les inclusions directes sont celles où un sous-groupe est inclus dans un autre sans passer par un sous-groupe intermédiaire.

```
Diagramme de Hasse:
Diagramme de Hasse des sous-groupes de D8:

Sous-groupe 0: [0]
  Inclus directement dans: Sous-groupe 1: [0, 2], Sous-groupe 2: [0, 4], So
us-groupe 3: [0, 5], Sous-groupe 4: [0, 6], Sous-groupe 5: [0, 7]

Sous-groupe 1: [0, 2]
  Inclus directement dans: Sous-groupe 6: [0, 1, 2, 3], Sous-groupe 7: [0,
2, 4, 6], Sous-groupe 8: [0, 2, 5, 7]

Sous-groupe 2: [0, 4]
  Inclus directement dans: Sous-groupe 7: [0, 2, 4, 6]

Sous-groupe 3: [0, 5]
  Inclus directement dans: Sous-groupe 8: [0, 2, 5, 7]

Sous-groupe 4: [0, 6]
  Inclus directement dans: Sous-groupe 7: [0, 2, 4, 6]

Sous-groupe 5: [0, 7]
  Inclus directement dans: Sous-groupe 8: [0, 2, 5, 7]

Sous-groupe 6: [0, 1, 2, 3]
  Inclus directement dans: Sous-groupe 9: [0, 1, 2, 3, 4, 5, 6, 7]

Sous-groupe 7: [0, 2, 4, 6]
  Inclus directement dans: Sous-groupe 9: [0, 1, 2, 3, 4, 5, 6, 7]

Sous-groupe 8: [0, 2, 5, 7]
  Inclus directement dans: Sous-groupe 9: [0, 1, 2, 3, 4, 5, 6, 7]

Sous-groupe 9: [0, 1, 2, 3, 4, 5, 6, 7]
```

### 3. Graphique du diagramme de Hasse :

Le diagramme de Hasse des sous-groupes de $D_8$ est également représenté sous forme graphique pour mieux visualiser les inclusions. Voici la visualisation générée par le programme :



# Question 3 – Généralisation automatique à $D_{2n}$

Dans cette question, l'objectif est d'**étendre automatiquement** l'étude du groupe diédral $D_{2n}$ pour tout entier n ≥ 2.

Le but est de pouvoir analyser les groupes $D_6$, $D_8$, $D_{10}$... sans réécrire manuellement chaque fonction. Pour cela, j'ai adapté tout le code précédent à une variable **n** paramétrable, ce qui permet une **généralisation complète**.

---

### 🔧 Fonctions généralisées :

Voici les fonctions que j'ai réécrites pour qu'elles fonctionnent avec n'importe quel **n ≥ 2** :

- `trad_dec_qu(k, n)` et `trad_qu_dec((α, β), n)` : conversion entre codage décimal et codage (rotation/symétrie).
- `compose(i, j, n)` : applique la bonne règle de composition selon le type de chaque transformation.
- `inverse(i, n)` : calcule l'inverse d'un élément du groupe $D_{2n}$.
- `construire_table_groupe(n)` : génère et affiche la table de Cayley pour le groupe $D_{2n}$.
- `sous_groupe_engendre(generateurs, n)` et `trouver_tous_sous_groupes(n)` : identifient automatiquement tous les sous-groupes de $D_{2n}$.
- `diagramme_hasse()` et `dessiner_hasse_vertical()` : construisent et affichent le diagramme de Hasse associé aux sous-groupes.

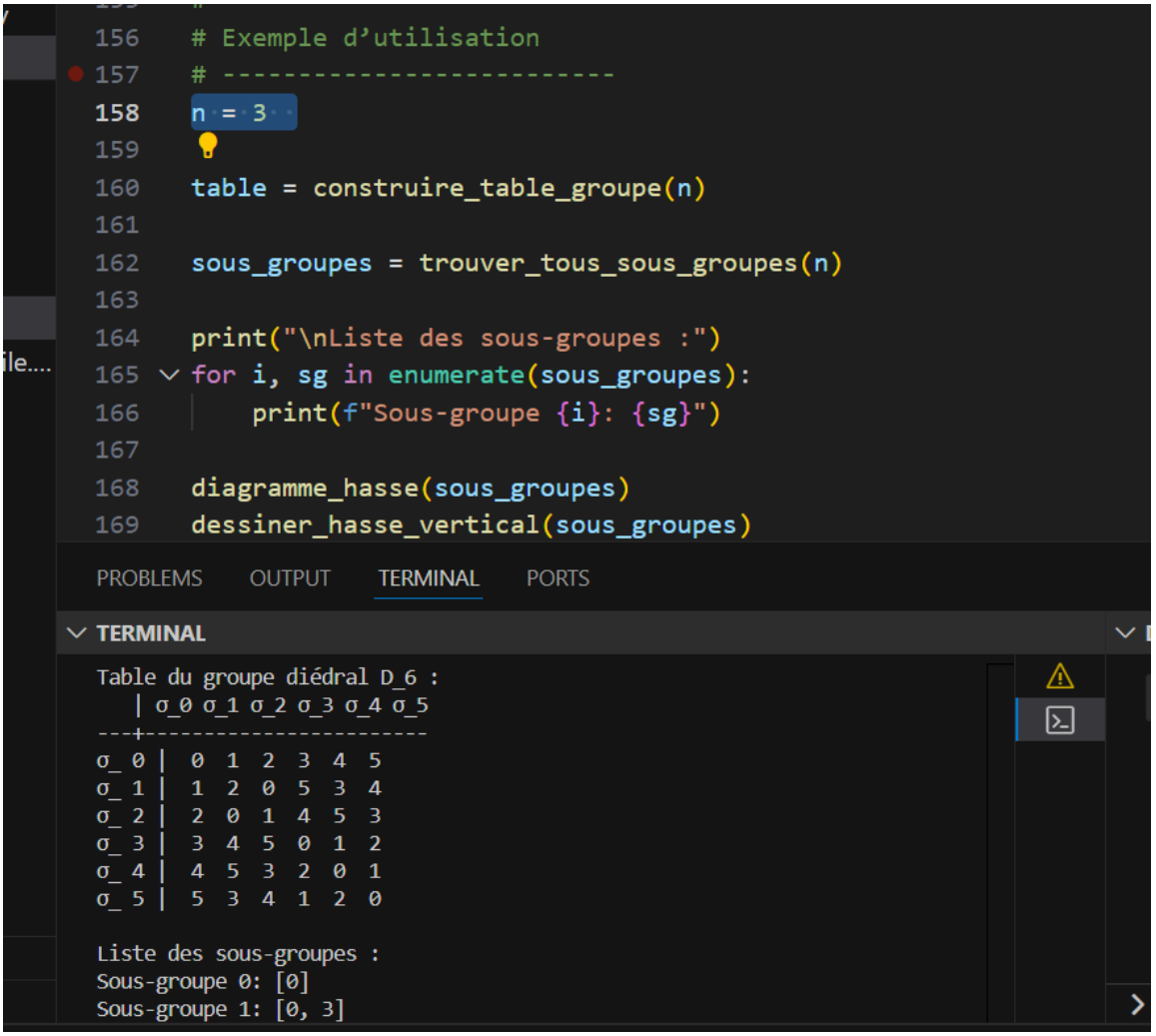
---

### 🧪 Exemple : Étude du groupe $D_6$ (n = 3) :

Pour tester la généricité du programme, j'ai utilisé n = 3, ce qui correspond au groupe $D_6$.

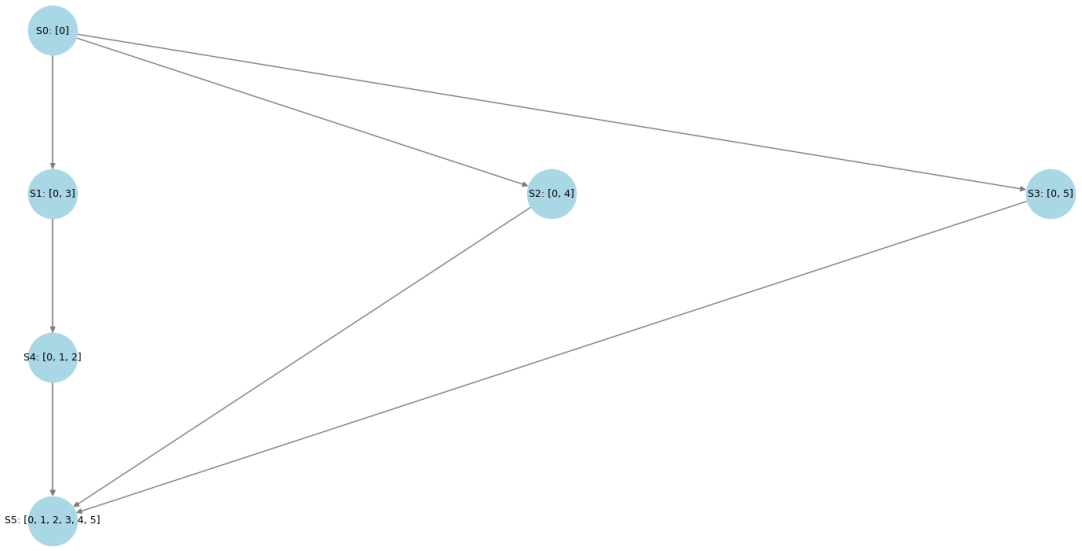
Résultats obtenus :

- **Table de composition correcte** (6 éléments = 3 rotations + 3 symétries),




- **6 sous-groupes identifiés automatiquement**, dont :
  - Le sous-groupe trivial `[0]`,
  - Le sous-groupe des rotations `[0, 1, 2]`,
  - Le groupe complet `[0, 1, 2, 3, 4, 5]`,
  - Trois sous-groupes d'ordre 2 formés d'une symétrie et de l'identité.

La **structure de Hasse** a aussi été générée automatiquement.




# Question 4 – Application des symétries du groupe $D_{2n}$ sur un motif géométrique

Dans cette partie, nous étudions **l'action géométrique du groupe diédral $D_{2n}$** sur un motif du plan.

Chaque transformation du groupe (rotation ou symétrie) est appliquée à un motif de base $M_o$, et les copies transformées sont affichées.

Cela permet de **visualiser la structure du groupe** à travers ses effets sur une figure géométrique.

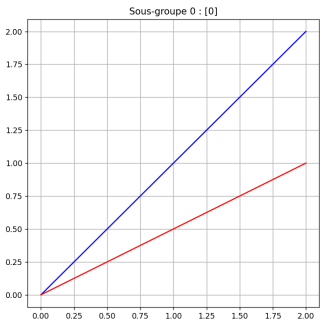
## 4a – Représentation d'un motif transformé par un sous-groupe de $D_{2n}$

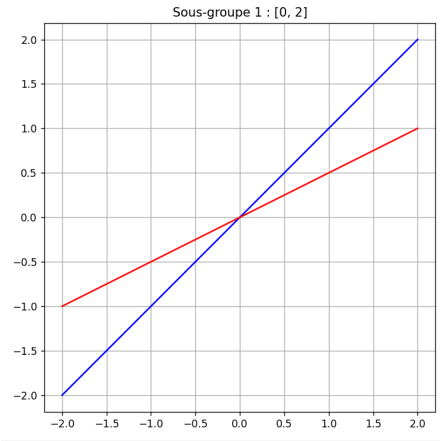
Le motif de base $M_o$ est composé de deux segments partant de l'origine :

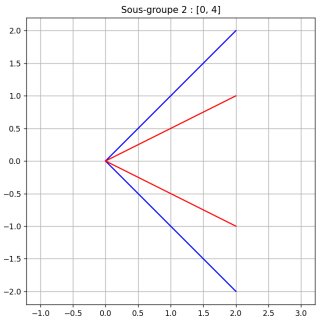
- OA avec A = (2, 2), représenté en bleu,
- OB avec B = (2, 1), représenté en rouge.

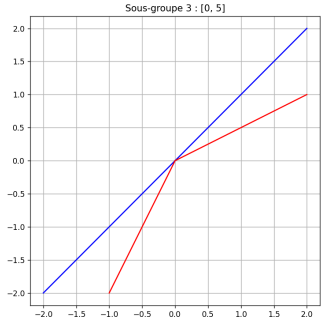
Pour chaque **sous-groupe H** de $D_8$, nous avons appliqué toutes les transformations $\sigma_i \in H$ au motif $M_o$, puis tracé l'ensemble des motifs transformés pour obtenir M_H.

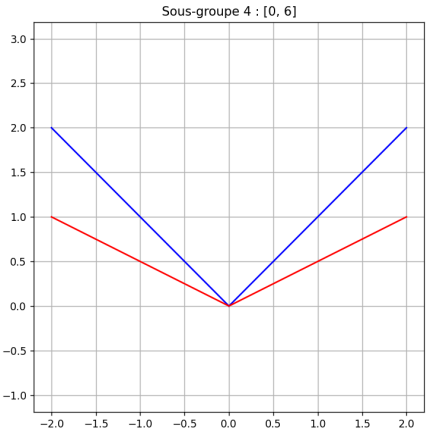
```
Affichage du motif pour le sous-groupe 0 : [0]
Affichage du motif pour le sous-groupe 1 : [0, 2]
Affichage du motif pour le sous-groupe 2 : [0, 4]
Affichage du motif pour le sous-groupe 3 : [0, 5]
Affichage du motif pour le sous-groupe 4 : [0, 6]
Affichage du motif pour le sous-groupe 5 : [0, 7]
Affichage du motif pour le sous-groupe 6 : [0, 1, 2, 3]
Affichage du motif pour le sous-groupe 7 : [0, 2, 4, 6]
Affichage du motif pour le sous-groupe 8 : [0, 2, 5, 7]
Affichage du motif pour le sous-groupe 9 : [0, 1, 2, 3, 4, 5, 6, 7]
```

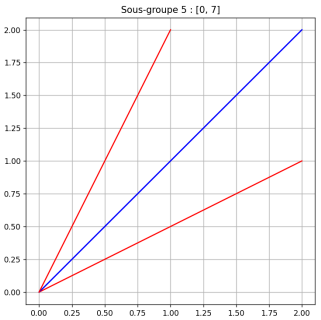


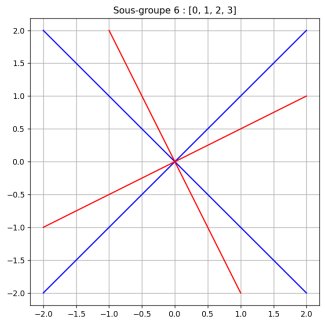



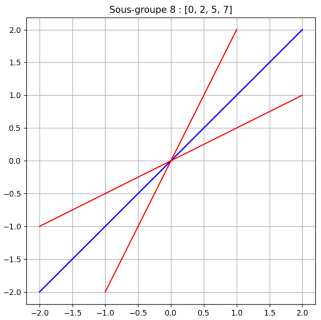



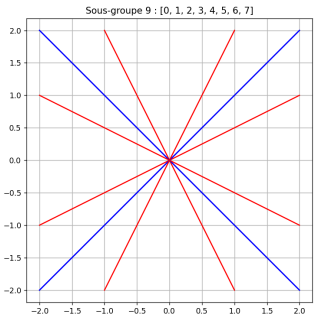



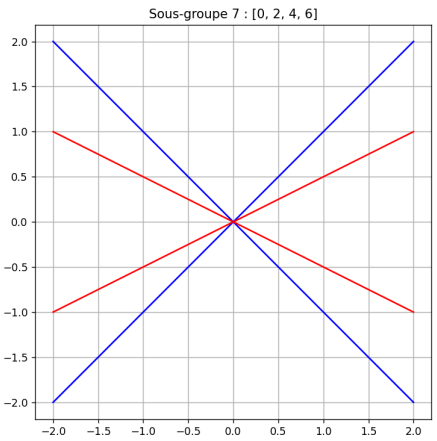






Les figures obtenues montrent que :

- Les **rotations** conservent l'orientation du motif,
- Les **symétries** inversent la direction des segments, visibles sur le segment OB,
- Les **compositions** produisent des figures symétriques autour de l'origine.

---

# 4b – Stratégie de transfert d'un motif

Pour appliquer un groupe de transformations G à un motif $M_o$, nous utilisons la méthode suivante :

1. Représenter chaque transformation du groupe comme une fonction agissant sur les points du plan (rotation ou rotation + symétrie).
2. Appliquer chaque transformation $\sigma_i \in G$ au motif $M_o$.
3. Afficher toutes les copies transformées pour obtenir la figure globale M_G.

Ce processus est systématique et généralisable à tout groupe fini d'isométries.

---

# 4c – Prolongement : motif plus élaboré

Pour aller plus loin, on peut remplacer le motif $M_o$ par une figure plus complexe:

- Une flèche, une lettre asymétrique, un petit polygone orienté...

Cela permettrait d'observer plus clairement les effets de symétries, notamment les inversions ou retournements causés par les réflexions.

La stratégie d’application reste la même, mais le motif permettrait de mieux percevoir visuellement les transformations.

---

# Conclusion du TP

Ce TP avait pour objectif d'étudier **le groupe diédral $D_{2n}$** à travers des outils **algébriques, programmatiques et géométriques**.

Nous avons d'abord :

- Implémenté les **isométries élémentaires du plan** (rotation et symétrie),
- Codé les transformations du groupe $D_8$ sous forme d’indices et de couples $(\alpha, \beta)$,
- Étudié la **composition**, les **inverses**, et la **structure non abélienne** du groupe.

Ensuite, nous avons :

- Généré la **table de composition** du groupe,
- Identifié automatiquement tous les **sous-groupes de $D_8$**,
- Et représenté leur organisation à l’aide d’un **diagramme de Hasse** textuel et graphique.

Enfin, nous avons :

- Appliqué les transformations de $D_8$ à un **motif géométrique**,
- Visualisé l’action des sous-groupes sur ce motif,
- Et démontré que le **groupe de symétries du motif** est **isomorphe à $D_8$**.

Ce travail a permis de lier de façon claire la **théorie des groupes finis**, leur **programmation** et leur **visualisation graphique**, tout en posant les bases d’une approche généralisable à tout groupe $D_{2n}$.

Remarque : le rapport a été rédigé sur Notion, puis exporté en PDF. La conversion a altéré la mise en page (sauts de page, marges...). Je privilégierai Word à l’avenir.