❏ **Q5** Proposer une requête en SQL pour extraire les noms des familles qui disposent de polices et leur nombre de polices, classés par ordre alphabétique.

Pour la suite, la requête de la question 4 est supposée paramétrée et encapsulée dans une fonction `glyphe(c, p, r)` qui renvoie la description vectorielle du caractère `c` dans la police `p` en roman ou italique selon le booléen `r`, de sorte que l'appel à `glyphe("a", "Helvetica", False)` répond à la question 4.

Partie III – Manipulation de descriptions vectorielles de glyphes

L'avantage de la description vectorielle de glyphes est qu'il est possible de réaliser des opérations sur les glyphes sans perte d'information. On peut réaliser simplement un agrandissement des glyphes, une déformation de glyphe pour en créer un nouveau etc. Cette partie propose des fonctions pour analyser et modifier des descriptions vectorielles.

Dans un premier temps, des fonctions sont créées pour extraire des informations sur des glyphes. Deux fonctions utilitaires sont implémentées.

❏ **Q6** Implémenter la fonction utilitaire `points(v:[[[float]]])->[float]` qui renvoie la liste des points qui apparaissent dans les multi-lignes de la description vectorielle `v` d'un glyphe.

```
v = [ [ [ 0, 0 ], [ 1, 1 ] ], [ [ 0, 1 ], [ 1, 0 ] ] ]
print(points(v))              # affiche la liste [ [ 0, 0 ], [ 1, 1 ], [ 0, 1 ], [ 1, 0 ] ]
```

❏ **Q7** Implémenter la fonction utilitaire `dim(l:[[float]], n:int)->[float]` qui renvoie la liste des éléments d'indice n (en commençant à 0) des sous listes de flottants, dont on supposera qu'ils existent toujours.

```
l = [ [ 1, 2 ], [ 3, 4 ], [ 5, 6 ], [ 7, 8 ] ]
print(dim(l, 1))                              # affiche la liste [ 2, 4, 6, 8 ]
```

On cherche à déterminer les dimensions (largeur et hauteur) d'un glyphe donné de manière à pouvoir les modifier par la suite si nécessaire.

❏ **Q8** Implémenter la fonction `largeur(v:[[[float]]])->float` qui renvoie la largeur de la description vectorielle `v`. Il faudra utiliser les fonctions utilitaires précédentes ainsi que les fonctions `max` et `min` de Python appliquées à des listes.

❏ **Q9** Implémenter la fonction `obtention_largeur(police:str)->[float]` qui renvoie une liste de largeurs pour toutes les lettres minuscules romanes et italiques (uniquement les 26 lettres non accentuées de a à z) de la police `police` dans l'ordre a roman, a italique, b roman, b italique...

On souhaite dériver automatiquement de nouvelles représentations vectorielles de glyphes à partir de représentations existantes.

Python permet de passer simplement des fonctions en paramètre d'autres fonctions. Par exemple, la fonction `applique` ci-après renvoie une nouvelle liste constituée en appliquant la fonction `f` à tous les éléments de la liste `l`.

```
def applique(f:callable, l:[])->[]:
    return [ f(i) for i in l ]

def incremente(i:int)->int:
    return i + 1

print(applique(incremente, [ 0, 5, 8 ]))     # affiche la liste [ 1, 6, 9 ]
```

❏ **Q10** En se basant sur l'exemple de la fonction `applique`, implémenter une fonction utilitaire `transforme(f:callable, v:[[[float]]])->[[[float]]]` qui prend en paramètres une fonction `f`, une description vectorielle `v` et qui renvoie une nouvelle description vectorielle construite à partir de `v` en appliquant la fonction `f` à chacun des points et en préservant la structure des multi-lignes. La fonction `f` passée en argument transforme un point en un autre point.

Soit la fonction `zzz` qui renvoie un nouveau point calculé de la façon suivante :