Universidad de Granada

Los grupos cristalográficos en Python

Alex Jurado González

Tutor: Pedro A. García Sánchez
Cotutor: Manuel Bullejos Lorenzo
Máster en Matemáticas
Departamento de Álgebra
Universidad de Granada, Febrero 2016.

Índice general

Parte I

Introducción

Desde siempre me ha apasionado el mundo de la geometría, sobre todo relacionada con el arte y con la naturaleza. Las simetrías y los patrones son algo que nos rodea, que convive con nosotros en nuestro día a día, algo que dota de belleza nuestro entorno. Por este motivo he indagado y profundizado en este tema.

Cuando acabé el grado en matemáticas, realicé un trabajo final de grado sobre la creación de teselaciones, propiedades y matemáticas inmersas en ellas. Ahora realizando el máster he decidido continuar con este estudio, llevando dichas teselaciones a la computación.

En el ambiente laboral, me dedico al mundo del arte y el tatuaje, más concretamente al tatuaje geométrico. Utilizo varios programas específicos de creación de arte geométrico, pero utilizando y agotando recursos me doy cuenta que ninguno de ellos te permite teselar utilizando todos los grupos cristalograficos, es decir, con todas las posibilidades existentes para teselar.

Este TFM me parecía una oportunidad para unir varios aspectos a aprender como matemático y como diseñador, con el objetivo de crear un programa que me permitiera abordar este problema, permitiéndome construir teselaciones con todos los grupos cristalográficos. Como matemático, estoy muy interesado en el aprendizaje de la programación. Es algo que pensando en un futuro laboral, ofrece gran variedad de posibilidades a graduados en matemáticas como es mi caso.

Por tanto, este TFM tiene dos objetivos claros, uno de ellos es aprender a programar con criterio y rigor, y otro de ellos es generar un programa que me sea de utilidad en mi ámbito laboral, la creación artística.

El trabajo se compone de dos partes; una fundamentación teórica sobre los grupos cristalográficos y una implementación de estos grupos en python.

La fundamentación teórica ha sido gran parte extraída de mi TFG realizado para el grado, pero me parecía necesaria notarla para poder entender la programación que se realiza a posteriori. El centro de trabajo de este TFM es la programación. Hemos decidido realizarlo en lenguaje python por varios motivos, es un lenguaje muy utilizado en la actualidad, y con nuevas versiones de python nos facilitarían el trabajo. Antes de decidirnos, probamos y barajamos otros lenguajes, los cuales fueron descartados por problemas que iban surgiendo durante la ejecución. Observamos que existía un modulo de python que nos permitía procesar imágenes de una forma sencilla y óptima. Esto ha sido un reto por el hecho de nunca haber trabajado en programación con algo distinto a la matemática numérica o simbólica.

En el apartado de programación hemos creado una clase y un tutorial de uso. Todo este material esta subido en mi perfil de github, y es libre para todo el mundo que desee utilizarlo. No obstante he decidido escribir el código fuente en la memoria, debido a que es de fácil comprensión y explican por si

solos los procedimientos computacionales utilizados.

Parte II

Grupos

Capítulo 1

Grupos Ornamentales del plano

En este capítulo haremos un estudio de lo que llamamos *grupos ornamentales del plano*. Reciben este nombre debido a que cuando hacemos una acción de estos grupos sobre una región del plano, obtenemos una teselación de $\mathbb{R}^2$. Usando estos grupos para teselar se consigue una ornamentación artística del plano.

1.1. Grupos de Rosáceas o Rosetones

Una isometría Euclidea es una transformación del plano del tipo:

$$f(x) = Ax + a$$

con A una matriz regular. Las isometrías Euclideas forma un grupo bajo la composición [4], [2].

A este grupo le llamamos grupo de simetrías del plano y lo denotaremos como

$$Sim = \langle \mathrm{Sim}(\mathbb{R}^2),\ \circ \rangle.$$

Con

$$\mathrm{Sim}(\mathbb{R}^2) = \{f : f(x) = Ax + b, b \in \mathbb{R}^2, A \in O(2)\}.$$

Un subgrupo muy importante de Sim con el que trabajaremos en nuestro estudio, es el que llamaremos *grupo Puntual* [4], [2], [18].

Este grupo esta generado por las Isometrías Euclídeas de $\mathbb{R}^2$ que no admiten subgrupos de traslaciones. Consecuentemente contiene al menos un punto fijo. Denotamos a este grupo como

$$Punt = \langle \mathrm{Punt}(\mathbb{R}^2),\ \circ \rangle.$$

Con

$$\text{Punt}(\mathbb{R}^2) = \{f \in \text{Sim}(\mathbb{R}^2) : f(x) = Ax, A \in O(2)\}.$$

Nuestro primer objetivo es estudiar los subgrupos finitos de *Punt*. Para ello, comenzamos recordando dos grupos bien conocidos.

Definición 1.1.1. El grupo cíclico de orden n es un grupo generado por un elemento de orden n. Lo denotamos como

$$C_n = \langle r \rangle = \{r : r^n = Id\} = \{Id, r, r^2, ..., r^{n-1}\}.$$

Definición 1.1.2. El grupo diédrico [2], [4] de orden $2n$ está generado por dos elementos r y s y lo presentamos de la forma:

$$D_n = \{r, s : r^n = s^2 = Id, sr = r^{-1}s\} = \{Id, r, r^2, ..., r^{n-1}, s, sr, sr^2, ..., sr^{n-1}\}.$$

Estos dos grupos son conocidos con el nombre de *Grupos de Leonardo da Vinci*. Son llamados así debido a que los aplicó en el diseño de capillas, de manera que al añadir nuevos elementos a la capilla inicial, se conservase la simetría de la misma. Leonardo descubrió que los únicos grupos de isometrías en el plano son los grupos cíclicos C_n y diédricos D_n, para todo n natural. Se le atribuye el nombre del siguiente teorema.

Teorema 1.1.3 (Leonardo da Vinci)**.** *Todo subgrupo finito de* $\text{Punt}(\mathbb{R}^2)$ *es isomorfo o bien a un grupo cíclico* C_n *o a un grupo diédrico* D_n

Dem: Sea $G \leqslant \text{Punt}(\mathbb{R}^2)$ un subgrupo. Denotaremos $|G|$ a su orden, que supondremos es finito.

Por otro lado, dado un punto $p \in \mathbb{R}^2$, denotaremos G_p a la órbita de p. Este es un subconjunto de puntos del plano cuyo cardinal es un cociente del orden de G. De manera que si G es un grupo finito cualquier punto del plano tiene una órbita finita. Notemos también que si $H \leq G$ es un subgrupo entonces la órbita de H en un punto p, H_p, es un subconjnto de G_p.

Observamos ahora que las traslaciones tienen orden infinito y por tanto G no puede tener traslaciones. Tampoco puede tener simetrías deslizantes ya que estas son composición de traslaciones y simetrías axiales.

No obstante, G puede tener rotaciones. Pero si suponemos que G tiene dos rotaciones, $r_1, r_2 \in G$, con centros de rotación distintos $c_1 = (a, 0)$ y $c_2 = (a', 0)$ (que podemos suponer en el eje $y = 0$), $c_1 \neq c_2$, y consideramos el subgrupo $H = \langle r_1, r_2 \rangle \leq G$ veamos que obtenemos una contradicción.

Tomamos los semiplanos

$$\pi_1 = \{(x, y) \in \mathbb{R}^2 : x < a\} \quad \text{y} \quad \pi_2 = \{(x, y) \in \mathbb{R}^2 : x > a'\}.$$

Podemos encontrar un natural $n_1 \in \mathbb{N}$ tal que $c_3 = r_2^{n_1}(c_1) \in \pi_2$. De manera que obtenemos las siguientes relaciones entre las distancias

$$d(c_2, c_3) = d(c_2, c_1) \quad \text{y} \quad d(c_3, c_1) > d(c_2, c_1).$$

De forma análoga, podemos encontrar un natural $n_2 \in \mathbb{N}$ tal que $c_4 = r_1^{n_2}(c_3) \in \pi_2$ con $d(c_2, c_4) > d(c_1, c_3) > d(c_1, c_2)$. Estamos generando así una sucesión de centros en la órbita de c_2, $\{c_n\}_n \subset H_{c_2}$, tal que las distancias a c_1 determinan una sucesión creciente $\{d(c_1, c_n)\}_n \longrightarrow \infty$.

Esto implica que la órbita H_{c_2} es infinita, con lo que llegamos a una contradicción. Por tanto si existen rotaciones en G han de tener el mismo centro (véase [18]).

Si G contiene dos simetrías axiales, podría ocurrir:

- Que los ejes de simetría se intersequen en un punto p, o bien,
- que los ejes de simetría sean paralelos.

En el primer caso G tendría una rotación con centro p (véase justificación en [6], [13]) y como G no puede tener rotaciones con centros distintos, todas las simetrías en G tienen ejes que han de interceptarse en un único punto p. En el otro caso G tendría una traslación (véase justificación en [6],[13]) que hemos visto no puede ocurrir.

Por tanto G no puede contener dos simetrías distintas con ejes paralelos.

Así, G contiene únicamente rotaciones con el mismo centro, en cuyo caso es isomorfo a un grupo cíclico C_n según el orden de rotación. O bien, G contiene rotaciones y además simetrías (con ejes de simetría que todos se cortan en un mismo punto) en cuyo caso es isomorfo a un grupo diédrico D_n (ver [12], [4], [2], [18] para una explicación más detallada). □

Dentro de los grupos ornamentales del plano, tenemos los *Grupos de Rosáceas*. Estos grupos reciben este nombre debido a que son los que generan las llamadas *Rosáceas*.

Definición 1.1.4. Llamamos *grupo de rosáceas* a cualquier subgrupo finito del grupo de puntual

$$R \subset \text{Punt}(\mathbb{R}^2).$$

Por el Teorema de Leonardo todo *grupo de rosáceas* es isomorfo a un grupo cíclico C_n o a un grupo diédrico D_n, para algún n. Existen infinitos *grupos de rosáceas* y puesto que tanto los grupos cíclicos como los diédricos tienen un punto fijo (al actuar sobre el plano) podemos afirmar que todo *grupo de rosáceas* tiene al menos un punto fijo (ver [3], [5]).

1.2. Grupos de Frisos

En esta sección hablaremos de los llamados *grupos de frisos*. Estos grupos son los que generan los llamados *frisos*, unos patrones de gran importancia artística y arquitectónica a lo largo de la historia. (Las referencias consultadas para el estudio general de grupos de frisos son las siguientes [6], [18], [5], [8] [14], [3], [13], [2], [4].)

Definición 1.2.1. Un grupo de frisos es un subgrupo del grupo de Simetrías $F \subset \mathrm{Sim}(\mathbb{R}^2)$ que admite un único subgrupo maximal que es cíclico y está generado por una traslación T_v. Luego

$$F \ \subset \ \mathrm{Punt}(\mathbb{R}^2) \oplus \langle T_v \rangle.$$

Como observación directa de la definición, tenemos que:

Todo grupo de frisos F deja invariante una recta l.

- **Rotaciones**: En F existen únicamente rotaciones de ángulo π con centro en l.

  Observamos que si tomamos una rotación r de ángulo $\theta \neq (2k+1)\pi, k \in \mathbb{N}$ tenemos que $r(l)$ es de la forma

  $$r(l) = \begin{pmatrix} cos(\theta) & sen(\theta) \\ -sen(\theta) & cos(\theta) \end{pmatrix} (vx + p) + v$$

  por tanto $r(l) \cap l \neq l$.

- **Simetrías**: Existen dos tipos de simetrías que dejan invariante la recta l. Una a la que llamaremos *simetría horizontal*, la cual tiene como eje de simetría la propia recta l, y otra que llamaremos *simetría vertical*, la cual tiene como eje una recta s donde s es perpendicular a l.

  Si tomamos cualquier otra simetría con ejes distintos a s y l, al simetrizar cualquier punto de l, la imagen de este no pertenece a l.

- **Simetrías con deslizamiento**: Una simetría con deslizamiento es una simetría seguida de una traslación. Como las únicas traslaciones en F son T_v y sus potencias. Tenemos que si F tiene simetrías con deslizamiento, estás están generadas por la que tiene eje de simetría la recta l y a T_v como traslación.

Para poder trabajar con estos elementos de una manera práctica, introducimos la siguiente notación:

Id: Identidad.

T: Traslación T_v.

r: Rotación de orden 2.

sh: Simetría horizontal.

sv: Simetría vertical.

sd: Simetría deslizante.

Claramente se tiene:

$$sd = sh \circ T, \quad sh^2 = Id, \quad sv^2 = Id.$$

La siguiente proposición nos da unas relaciones no tan obvias.

Proposición 1.2.2 ([13, 6]). *Tenemos las siguientes identidades:*

1. $T \circ sv = sv \circ T^{-1}$.
2. $T \circ sh = sh \circ T$.
3. $sv \circ sh = r$.

Dem:

1. Sea $(x, y) \in \mathbb{R}^2$, $T(x, y) = (x + c, y)$ y $sv(x, y) = (-x, y)$, entonces

$$T(sv(x, y)) = T((-x, y)) = (-x + c, y)$$

y

$$sv(T^{-1}(x, y)) = sv(x - c, y) = (-x + c, y).$$

2. Sea $(x, y) \in \mathbb{R}^2$, $T(x, y) = (x + c, y)$ y $sh(x, y) = (x, -y)$, entonces

$$T(sh(x, y)) = T((x, -y)) = (x + c, -y)$$

y

$$sh(T(x, y)) = sh(x + c, y) = (x + c, -y).$$

3. Sea $(x, y) \in \mathbb{R}^2$,

$$r(x,y) = \begin{pmatrix} cos(\pi) & sen(\pi) \\ -sen(\pi) & cos(\pi) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix},$$
$$sh(x,y) = (x,-y),$$
$$sv(x,y) = (-x,y).$$

Entonces:

$$sv(sh(x,y)) = sv((x,-y)) = (-x,-y)$$

y

$$r((x,y)) = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = (-x,-y).$$

□

1.2.1. Identificación de los grupos

Veamos algunos ejemplos de *grupos de Frisos*.

Nos ajustamos a una notación llamada *Fejes Tóth* (véase [19]) en la que se utiliza la letra F, debido al nombre $Frieze$, para los ornamentos del plano generados bajo la acción de estos grupos. Además se utilizan los índices i y j, de manera que el grupo será denotado F_j^i donde:

- El índice j denota el mayor orden de rotación y
- el índice i toma valores:
  - 1 si contiene simetría horizontal,
  - 2 si contiene simetría vertical o
  - 3 si contiene simetría deslizante.

Así

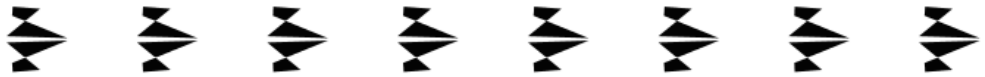
- $F_1 = \langle T \rangle = \{T^k : k \in \mathbb{Z}\}$.
- $F_1^1 = \langle T, sh \rangle = \{sh^m T^k : m \in \mathbb{Z}_2, k \in \mathbb{Z}\}$.
- $F_1^2 = \langle T, sv \rangle = \{sv^m T^k : m \in \mathbb{Z}_2, k \in \mathbb{Z}\}$.
- $F_1^3 = \langle sd \rangle = \{sd^k : k \in \mathbb{Z}\}$.
- $F_2 = \langle T, r \rangle = \{r^m T^k : m \in \mathbb{Z}_2, k \in \mathbb{Z}\}$.
- $F_2^1 = \langle T, sh, r \rangle = \{sh^m r^l T^k : m, l \in \mathbb{Z}_2, k \in \mathbb{Z}\}$.

- $F_2^2 = \langle sd, r \rangle = \{r^l sd^k : l \in \mathbb{Z}_2, k \in \mathbb{Z}\}$.

Estos grupos, en ocasiones, están notados con la que llamada *notación cristalográfica*. Esta la especificaremos cuando hablemos de *grupos cristalográficos*. Para verlos de una manera más intuitiva, veamos un ejemplo gráfico de cada uno de ellos (véase [14], [3], [13], [2], [4], [6], [18], [5], [8]):

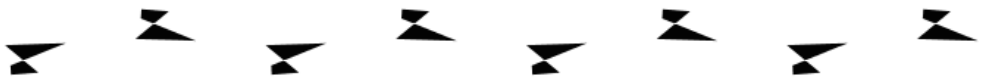
- El grupo F_1 se compone de una única traslación T en la dirección de la recta invariante l.

- El grupo F_1^1 se compone de una simetría horizontal sh, seguida de una traslación T.



- El grupo F_1^2 se compone de una simetría vertical sv, seguido de una traslación.

- El grupo F_1^3 se compone de una simetría deslizante sd.



- El grupo F_2 se compone de una rotación r, seguida de una traslación T.

- El grupo F_2^1 se compone de una simetría horizontal sh seguida de una rotación r, seguida de una traslación T.

- El grupo F_2^2 se compone de una rotación r seguida de una simetría deslizante sd.

El siguiente teorema nos determinará cuales son exactamente los *grupos de frisos*.

Teorema 1.2.3. *[7] Si F es un grupo de friso, es necesariamente uno de los nombrados anteriormente. Luego existen 7 grupos de frisos.*

Dem: Para la demostración del teorema consideramos $sh, sv, sd, r \in \mathrm{Sim}(\mathbb{R}^2)$. Puesto que estos son las únicas simetrías que pueden estar en un *grupo de friso*, procedemos a estudiar todas las simetrías que se pueden generar a partir de estas.

Nos restringimos a 4 elementos posibles ya que por definición, todo *grupo de friso* debe de contener a T. Por consiguiente el número de combinaciones posibles es el siguiente:

$$\binom{4}{0} + \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 1 + 4 + 6 + 4 + 1 = 16.$$

Estudiamos cada una de las posibilidades, incluyendo 1, 2, 3 o 4 elementos al grupo sin contar la traslación, que como hemos notado, debe de pertenecer a todos los grupos.

- En el primer caso, estudiamos las combinaciones con ningún elemento. Obtenemos un único grupo.
  - $\langle T \rangle = F_1$.

- Estudiamos ahora grupos con un elemento.
  - $\langle T, sv \rangle = F_1^2$
  - $\langle T, sh \rangle = F_1^1$
  - $\langle sd \rangle = F_1^3$
  - $\langle T, r \rangle = F_2$

- En este caso utilizaremos las propiedades probadas anteriormente para distinguir los grupos con dos elementos.
  - $sd \circ sv = ((T \circ sh) \circ sv) = T \circ r$, por tanto $\langle T, r \rangle = F_2$
  - $sd \circ sh = ((T \circ sh) \circ sh) = T$, por tanto $\langle T \rangle = F_1$
  - $\langle sd, r \rangle = F_2^2$
  - $sv \circ sh = r$, por tanto$\langle T, r \rangle = F_2$
  - $sv \circ r = (sv \circ (sv \circ sh) = sh$, por tanto $\langle T, sh \rangle = F_1^1$
  - $sh \circ r = (sh \circ (sh \circ sv) = sv$, por tanto $\langle T, sv \rangle = F_1^2$

- Casos con tres elementos, son los siguientes.
  - $sd \circ sh \circ sv = ((T \circ sh) \circ sh \circ sv) = T \circ sv$, por tanto $\langle T, sv \rangle = F_1^2$
  - $sd \circ sv \circ r = (T \circ sh) \circ sv \circ (sv \circ sh) = T$, por tanto $\langle T \rangle = F_1$
  - $sd \circ sh \circ r = (T \circ sh) \circ sh \circ r = T \circ r$, por tanto $\langle T, r \rangle = F_2$
  - $sv \circ sh \circ r = sv \circ sh \circ (sh \circ sv) = Id$, por tanto$\langle T \rangle = F_1$

- Casos con cuatro elementos solo tenemos el siguiente.
  - $sd \circ sv \circ sh \circ r = sd \circ Id = sd$, por tanto $\langle sd \rangle = F_1^3$.

□

1.3. Grupos Cristalográficos

En esta sección trabajaremos los *grupos cristalográficos*. Se trata de una serie de subgrupos del grupo de simetrías tales que contienen dos traslaciones independientes. Las teselaciones generadas por estos grupos han sido estudiadas a lo largo de la historia. Estos grupos reciben el nombre de *cristalográficos* debido a que se trata de una restricción al plano del estudio de la formación de cristales en tres dimensiones.

En la formación de cristales en la naturaleza se observarón ciertos patrones, que pueden encajarse dentro de unos grupos de simetría en el espacio. En el desarrollo de este estudio, se observó, qué sucedía, cuando se trasladaban al plano dichos grupos, observando que daban lugar a 17 grupos. Dichos grupos generan muchas de las teselaciones que observamos en la naturaleza, en tantas obras de arte y en construcciones arquitectónicas.

Definición 1.3.1. Llamamos grupo cristalográfico a cualquier subgrupo del grupo de simetrías $W \subset \mathrm{Sim}(\mathbb{R}^2)$ que contienen dos traslaciones independientes, es decir, existen traslaciones T_u y T_v en W, con u y v dos vectores linealmente independientes y tal que el grupo $\langle T_u, T_v \rangle$ es discreto.

$$W \subset \mathrm{Punt}(\mathbb{R}^2) \oplus \langle T_u, T_v \rangle.$$

Antes de entrar en el estudio de dichos grupos definimos el concepto de *región fundamental* y *tesela*.

Definición 1.3.2. Llamamos *región fundamental* de un grupo de simetrías G a la menor región del plano sobre la que actúa G para generar una teselación.

Definición 1.3.3. Llamamos *tesela* a la menor región del plano sobre la que actúa el subgrupo de traslaciones para generar una teselación.

Para identificar los *grupos cristalográficos* vamos a usar la notación estándar [19, 20], llamada *notación cristalográfica*, basada en la cristalografía y estandarizada por la Unión Internacional de Cristalografía. Esta se basa en expresar cada *grupo cristalográfico* usando cuatro caracteres a_1, a_2, a_3, a_4.

- $\mathbf{a_1}$:
  • Toma el valor p si la región fundamental es primitiva. Esto es, los dos ejes que forman las traslaciones coinciden con los lados de la región fundamental.
  •Toma el valor c si la región fundamental es centrada. Esto es si alguno de los dos ejes que forman las traslaciones atraviesa la región fundamental.

- $\mathbf{a_2}$: Indica el mayor orden de rotación de las rotaciones que contiene el grupo , 1(ausencia de rotaciones), 2, 3, 4 o 6

- $\mathbf{a_3}$: Indica ejes de simetría perpendiculares al eje de la región fundamental:
  • Toma el valor m si tiene un eje de simetría perpendicular a un lado

de la región fundamental.
•Toma el valor g si tiene un eje de simetría con deslizamiento perpendicular a un lado de la región fundamental.
• Toma el valor 1 en el resto de los casos.

- $\mathbf{a_4}$: Indica ejes de simetría con ángulo θ respecto los lados de la región fundamental. El orden de rotación n debe ir relacionado con θ de la siguiente manera:$[n, \theta]$=$[2, \frac{\pi}{2}]$, $[4, \frac{\pi}{4}]$ o bien $[3\ o\ 6, \frac{\pi}{6}]$
  • Toma el valor m si contiene simetrías axiales cumpliendo la relación $[n, \theta]$
  • Toma el valor g si contiene simetrías deslizantes cumpliendo la relación $[n, \theta]$
  • Toma el valor 1 en el resto de los casos.

Notamos que en algunos casos se pueden suprimir algún término de la notación, cuando por ejemplo este proporciona información redundante. A esta se le llama notación cristalográfica reducida.

1.3.1. Identificación de los grupos

Para identificar los elementos que componen estos *grupos cristalográficos*, vamos a ver un teorema de gran importancia en el estudio de dichos grupos. Este es, el *teorema de restricción cristalográfica* [4], [18] el cual nos determina el orden de las simetrías rotacionales que caracterizan estos grupos.

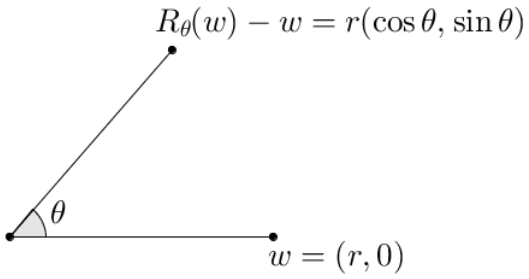
Teorema 1.3.4 (Restricción Cristalográfica)**.**
Sea W un grupo cristalográfico, las rotaciones contenidas en W son de orden 2,3,4 o 6.

Dem: Sea $w \in \langle T_u, T_v \rangle$ minimal con $\| w \| = r$, $r > 0$. Tomamos el conjunto

$$S = \{w \in \langle T_u, T_v \rangle : \ \| w \| = r\}.$$

Tomamos $\overline{B}(0, r)$ como la bola de radio r y centro 0. Tenemos que S es discreto en $\overline{B}(0, r)$. Luego aplicamos el *teorema de Bolzano-Weierstrass*, el cual nos dice que todo conjunto discreto en un compacto es finito. Por tanto, puesto que $\langle T_u, T_v \rangle$ es discreto en $\overline{B}(0, r)$, por el *teorema de Bolzano-Weierstrass*, S es finito. Además al hacer actuar el grupo puntual de cualquier W, un *grupo cristalográfico*, en el conjunto S, deja a este invariante. Por consiguiente obtenemos que las rotaciones de W tienen orden finito.

Veamos ahora el orden de dichas rotaciones. Tomamos R_θ una rotación minimal, luego $\theta \in [0, \pi]$. Tenemos que $R_\theta(w) - w \in \langle T_u, T_v \rangle$, con $\| R_\theta(w) - w \| \geqslant r$.



Entonces $\| R_\theta(w) - w \| = \| (r(\cos\theta - 1), r\,\mathrm{sen}\,\theta) \| = r\sqrt{2(1-\cos\theta)} \geqslant r \Rightarrow 1 - \cos\theta \geqslant \frac{1}{2} \Rightarrow \cos\theta \leq \frac{1}{2} \Rightarrow \theta \in [\frac{\pi}{3}, \pi]$. Pero θ debe verificar $\theta = \frac{2\pi}{n}$ luego $\theta \in [\frac{\pi}{3}, \pi] \Leftrightarrow n = 2, 3, 4, 5, 6$.
Procedemos ahora a rechazar el caso $n = 5$:
Sea $R_{\frac{2\pi}{5}}$ una rotación de orden 5. Si tomamos $R^2_{\frac{2\pi}{5}}(w) + w \in \langle T_u, T_v \rangle$, tenemos que: $\| R^2_{\frac{2\pi}{5}}(w) + w \| = r\sqrt{(\cos\frac{2\pi}{5} - 1)^2 + \mathrm{sen}^2\,\frac{2\pi}{5}} \geqslant r \Leftrightarrow \cos\frac{2\pi}{5} - 1 \geqslant \frac{1}{2} \Leftrightarrow \cos\frac{2\pi}{5} \geqslant \frac{3}{2}$ con lo que llegamos a una contradicción. Lo que descarta el caso $\theta = \frac{2\pi}{5}$.
□

Procedemos ahora a describir cada uno de los 17 *grupos cristalográficos*. Para ello damos un gráfico conceptual de la tesela de cada uno interpretando la región fundamental inmersa. La notación usada se basa en la imagen de una elipse, un triángulo, un cuadrado o un hexágono para denotar orden de rotación 2, 3, 4 y 6 respectivamente. Nótese que las rotaciones de orden 4 tienen contenidas rotaciones de orden 2 y las rotaciones de orden 6 tienen contenidas rotaciones de orden 2 y 3. Par notar las simetrías, se le asigna un segmento rojo a los ejes de simetría de una simetría axial, y un segmento rojo discontinuo para los ejes de simetrías deslizantes. Los segmentos negros delimitan la región fundamental. Hemos seguido el procedimiento dado en [10].

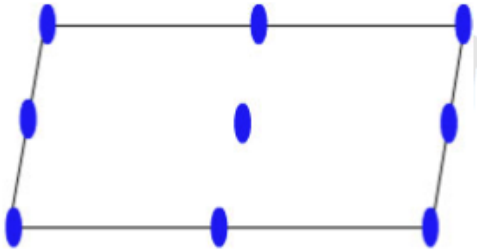
- **p1** (**p111**): Es el grupo generado por dos traslaciones independientes, luego podemos notarlo como $p1 = \langle T_u, T_v \rangle$.

- **pm** (**p1m1**): Es el grupo que contiene dos traslaciones independientes y simetrías axiales con ejes paralelos al eje de una de estas traslaciones. Luego podemos notarlo como $pm = \langle T_u, T_v, sh \rangle$, siendo sh dicha simetría.

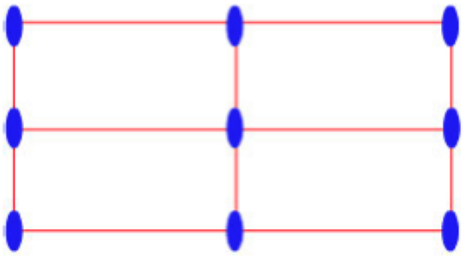
- **pg** (**p1g1**): Es el grupo que contiene dos traslaciones independientes y simetrías deslizantes con ejes paralelos al eje de una de estas traslaciones. Luego podemos notarlo como $pg = \langle T_u, sd \rangle$, siendo sd dicha simetría.

- **cm** (**c1m1**): Es el grupo que contiene dos traslaciones independientes, simetrías axiales y simetrías deslizantes, con ejes paralelos al eje de una de estas traslaciones. Luego podemos notarlo como $cm = \langle T_u, T_v, sh, sd \rangle$, siendo sh, sd dichas simetrías.

- **p2** (**p211**): Es el grupo que contiene dos traslaciones independientes y rotaciones de orden dos. Luego podemos notarlo como $p2 = \langle T_u, T_v, r_2 \rangle$, siendo r_2 dicha rotación.

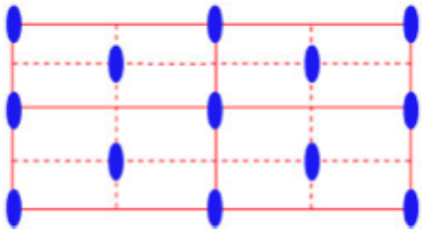


- **pmm** (**p2mm**): Es el grupo que contiene dos traslaciones independientes, simetrías axiales y rotaciones de orden 2. Notamos T_u y T_v a las dos traslaciones. En este caso existen simetrías axiales unas con eje l en la dirección de T_u y otras con eje l' en la dirección de T_v, las cuales notamos como sl y sl' respectivamente. Las rotaciones de orden 2, contienen el centro de giro en la intersección de l y l', las cuales notamos $r2$. Por tanto notamos el grupo como $p2mm = \langle T_u, T_v, sl, sl', r_2 \rangle$

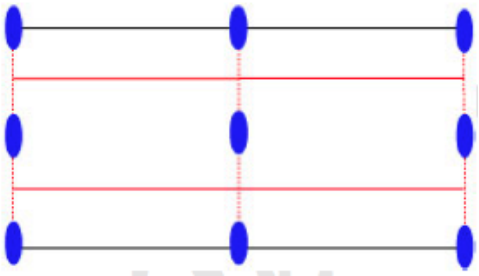


- **cmm** (**c2mm**): Es el grupo que contiene dos traslaciones independientes, simetrías axiales, simetrías deslizantes y rotaciones de orden 2. Notamos T_u y T_v a las dos traslaciones. En este caso existen simetrías axiales y deslizantes, con ejes l en la dirección de T_u y con ejes l' en la dirección de T_v, las cuales notamos como sl y sl'para las simetrías

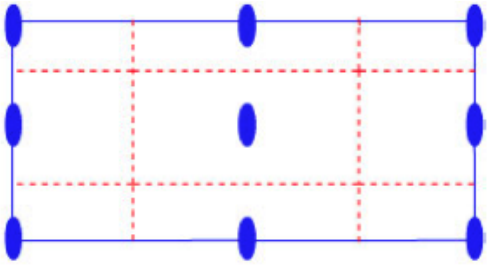
axiales y sdl y sdl' para las simetrías deslizantes. Las rotaciones de orden 2, contienen el centro de giro o bien en la intersección de los ejes no paralelos de las simetrías axiales, o bien en la intersección de los ejes no paralelos de las simetrías deslizantes. Estas las notamos como r_2. Por tanto notamos el grupo como $c2mm = \langle T_u, T_v, sl, sl', sdl, sdl', r_2 \rangle$



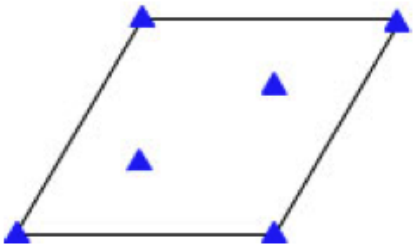
- **pmg** (**p2mg**):Es el grupo que contiene dos traslaciones independientes, simetrías axiales, simetrías deslizantes y rotaciones de orden 2. Notamos T_u y T_v a las dos traslaciones. En este caso existen simetrías axiales con ejes l en la dirección de T_u y simetrías deslizantes con ejes l' en la dirección de T_v. Notamos como sl las simetrías axiales y sdl' las simetrías deslizantes. Las rotaciones de orden 2, contienen el centro de giro en los ejes de las simetrías y en el centro de la región del plano sobre la que actúan. Estas las notamos como $r2$. Por tanto notamos el grupo como $p2mg = \langle T_u, T_v, sl, sdl', r_2 \rangle$



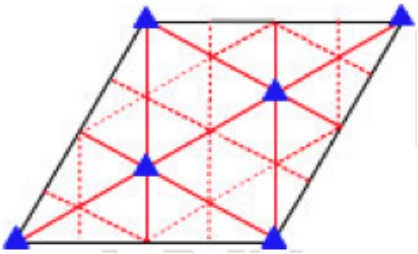
- **pgg** (**p2gg**):Es el grupo que contiene dos traslaciones independientes, simetrías deslizantes y rotaciones de orden 2. Notamos T_u y T_v a las dos traslaciones. En este caso existen simetrías deslizantes unas con eje l en la dirección de T_u y otras con eje l' en la dirección de T_v, las cuales notamos como sdl y sdl' respectivamente. Las rotaciones de orden 2, contienen el centro de giro en los ejes que delimitan la región del plano sobre la que actúan y en el centro de esta, las cuales notamos $r2$. Por tanto notamos el grupo como $p2gg = \langle T_u, T_v, sdl, sdl', r_2 \rangle$



- **p3** (**p311**): Es el grupo que contiene dos traslaciones independientes y rotaciones de orden tres. Luego podemos notarlo como $p3 = \langle T_u, T_v, r_3 \rangle$, siendo r_3 dicha rotación.

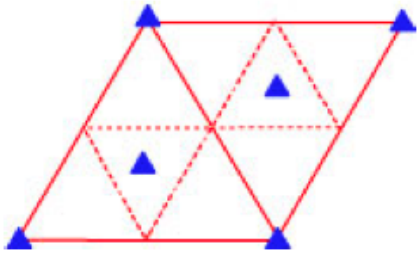


- **p3m1**: Es el grupo que contiene dos traslaciones independientes, simetrías axiales, simetrías deslizantes y rotaciones de orden 3. Notamos T_u y T_v a las dos traslaciones. En este caso existen simetrías axiales y deslizantes, con ejes l, l', l'' donde l lleva la dirección de $T_u \circ T_v$, ademas estos ejes forman un ángulo de $\frac{\pi}{3}$ dos a dos. Notamos sl, sl' y sl'' a las simetrías axiales y sdl, sdl' y sdl'' a las simetrías deslizantes. Las rotaciones de orden 3, contienen el centro de giro en la intersección de los ejes de las simetrías axiales. Estas las notamos como $r3$. Por tanto notamos el grupo como $p3m1 = \langle T_u, T_v, sl, sl', sl'', sdl, sdl', sdl'', r_3 \rangle$

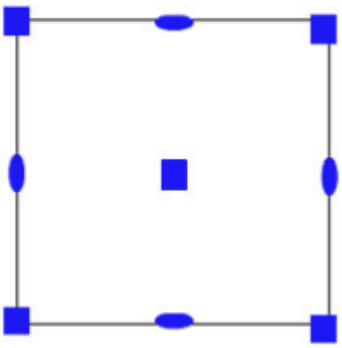


- **p31m**: Es el grupo que contiene dos traslaciones independientes, simetrías axiales, simetrías deslizantes y rotaciones de orden 3. Notamos

T_u y T_v a las dos traslaciones. En este caso existen simetrías axiales y deslizantes, con ejes l, l', l'' donde l lleva la dirección de T_u, l' la dirección de T_v y l'' forma un ángulo de $\frac{\pi}{3}$ con l y l''. Notamos sl, sl' y sl'' a las simetrías axiales y sdl, sdl' y sdl'' a las simetrías deslizantes. Las rotaciones de orden 3, contienen el centro de giro en la intersección de los ejes de las simetrías axiales y en el centro de la región que delimitan las simetrías deslizantes. Estas las notamos como $r3$. Por tanto notamos el grupo como $p31m = \langle T_u, T_v, sl, sl', sl'', sdl, sdl', sdl'', r_3 \rangle$

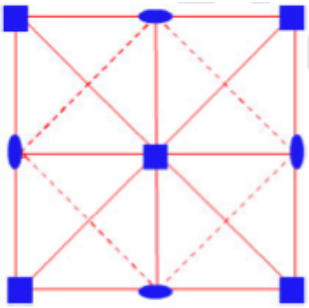


- **p4** (**p411**): Es el grupo que contiene dos traslaciones independientes y rotaciones de orden dos y cuatro. Luego podemos notarlo como $p4 = \langle T_u, T_v, r_2, r_4 \rangle$, siendo r_2, r_4 las rotaciones de orden dos y orden 4 respectivamente.

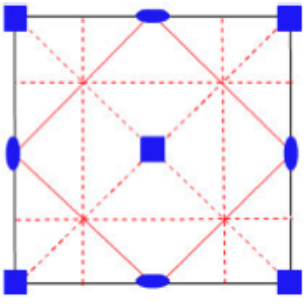


- **p4m** (**p4mm**): Es el grupo que contiene dos traslaciones independientes, simetrías axiales, simetrías deslizantes, rotaciones de orden 2 y rotaciones de orden 4. Notamos T_u y T_v a las dos traslaciones. En este caso existen simetrías axiales y deslizantes, con ejes l, l', l'', l''', donde l lleva la dirección de T_u, l' la dirección de T_v, l'' la dirección de $T_u \circ T_v$ y l''' la dirección de $T_u \circ T_v^{-1}$. Notamos sl, sl', sl'' y sl''' a las simetrías axiales

y sdl'' y sdl''' a las simetrías deslizantes. Las rotaciones de orden 2, contienen el centro de giro en la intersección de los ejes de las simetrías deslizantes. Mientras que las rotaciones de orden 4, contienen el centro de giro en la intersección de los ejes de las simetrías axiales. Por tanto notamos el grupo como $p4mm = \langle T_u, T_v, sl, sl', sl'', sl''', sdl'', sdl''', r_2, r_4 \rangle$, siendo r_2, r_4 las rotaciones de orden dos y orden 4 respectivamente.

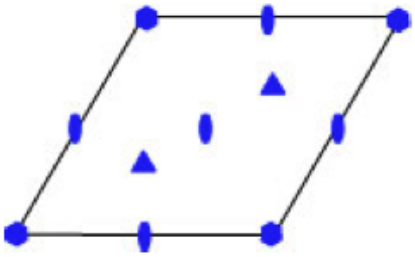


- **p4g** (**p4gg**): Es el grupo que contiene dos traslaciones independientes, simetrías axiales, simetrías deslizantes, rotaciones de orden 2 y rotaciones de orden 4. Notamos T_u y T_v a las dos traslaciones. En este caso existen simetrías axiales y deslizantes con ejes l, l', l'', l''', donde l lleva la dirección de T_u, l' la dirección de T_v, l'' la dirección de $T_u \circ T_v$ y l''' la dirección de $T_u \circ T_v^{-1}$. Notamos sl'' y sl''' a las simetrías axiales y sdl, sdl', sdl'' y sdl''' a las simetrías deslizantes. Las rotaciones de orden 2, contienen el centro de giro en la intersección de los ejes de las simetrías axiales. Mientras que las rotaciones de orden 4, contienen el centro de giro en los ejes de las simetrías deslizantes con ejes l'' y l'''. Por tanto notamos el grupo como $p4mm = \langle T_u, T_v, sl'', sl''', sdl, sdl', sdl'', sdl''', r_2, r_4 \rangle$, siendo r_2, r_4 las rotaciones de orden 2 y 4 respectivamente.

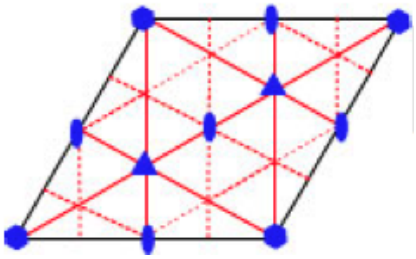


- **p6** (**p611**): Es el grupo que contiene dos traslaciones independientes y rotaciones de orden 2, 3 y 6. Luego podemos notarlo como $p6 =$

$\langle T_u, T_v, r_2, r_3, r_6 \rangle$, siendo r_2, r_3 y r_6 las rotaciones de orden 2, 3 y 6 respectivamente.



- **p6m** (**p6mm**): Es el grupo que contiene dos traslaciones independientes, simetrías axiales, simetrías deslizantes y rotaciones de orden 2, 3 y 6. Notamos T_u y T_v a las dos traslaciones. En este caso existen simetrías axiales y deslizantes, con ejes l, l', l'' donde l lleva la dirección de $T_u \circ T_v$, ademas estos ejes forman un ángulo de $\frac{\pi}{3}$ dos a dos. Notamos sl, sl' y sl'' a las simetrías axiales y sdl, sdl' y sdl'' a las simetrías deslizantes. Las rotaciones de orden 3, contienen el centro de giro en la intersección de los ejes de las simetrías axiales. Por tanto notamos el grupo como $p6mm = \langle T_u, T_v, sl, sl', sl'', sdl, sdl', sdl'', r_3 \rangle$, siendo r_2, r_3 y r_6 las rotaciones de orden 2, 3 y 6 respectivamente.



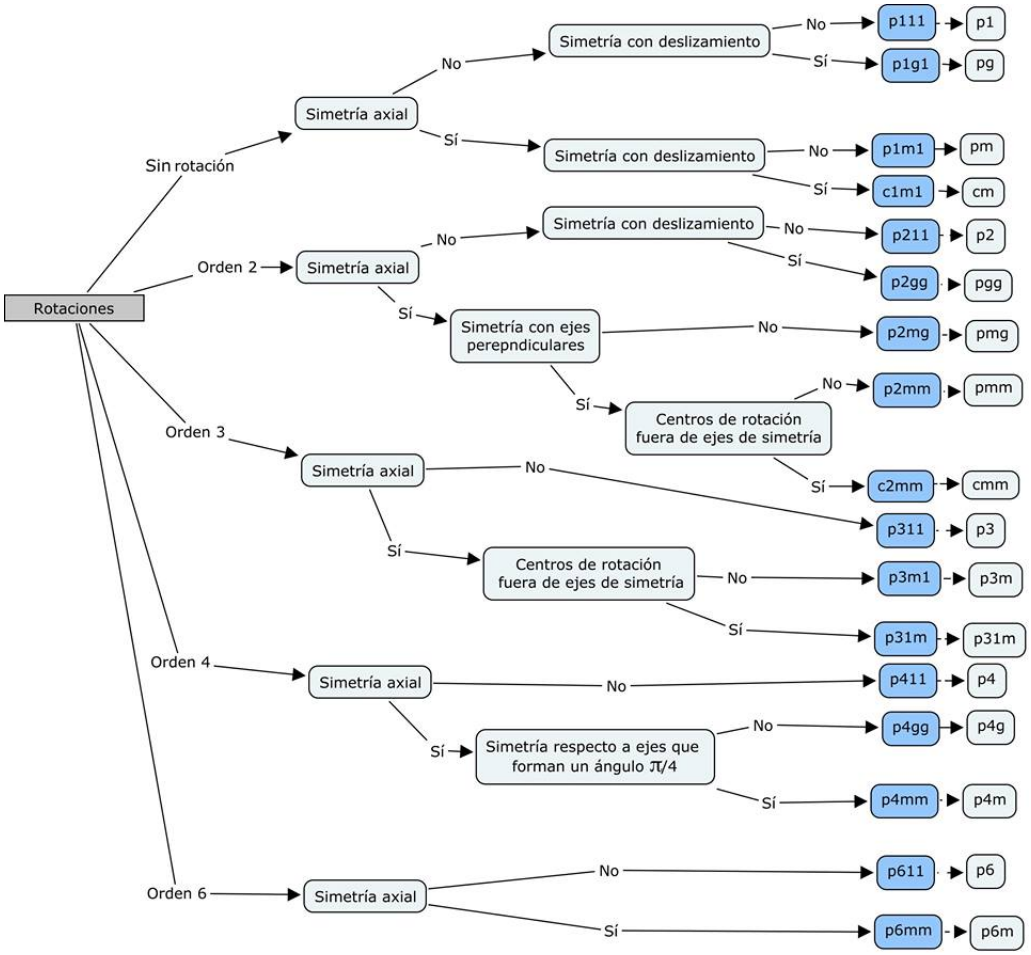
Una vez identificados los grupos cristalográficos, enunciamos el teorema de Federov (véase [11], [5], [12], [9], [4], [6], [18], [8], [2], [1]), que fue demostrado por el propio Federov en 1891.

Teorema 1.3.5 (Federov). *Si G es un grupo cristalográfico, es necesariamente uno de los nombrados anteriormente. Luego existen 17 y solo 17 grupos cristalográficos del plano.*

1.4. Algoritmos de clasificación

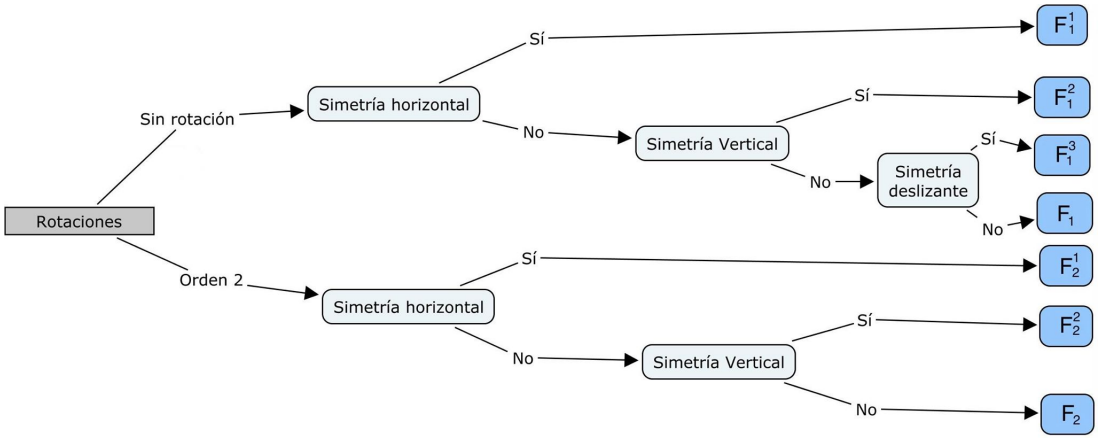
1.4.1. Algoritmo de clasificación de grupos cristalográficos

Hemos estudiado como generar teselaciones. Procedemos ahora a el estudio de como identificar la creación de una teselación. Esto es, identificar el grupo cristalográfico que ha actuado para generarla. Para ello tenemos el siguiente algoritmo de clasificación [8] [5]:



1.4.2. Algoritmo de clasificación de frisos

En el caso de los frisos, el algoritmo de clasificación es el siguiente


Rotaciones
Sin rotación
Orden 2
Simetría horizontal
Sí
No
Simetría Vertical
Sí
No
Simetría deslizante
Sí
No
Simetría horizontal
Sí
No
Simetría Vertical
Sí
No
F_1^1
F_1^2
F_1^3
F_1
F_2^1
F_2^2
F_2

Capítulo 2

Implementación de los grupos en python

En este capítulo nos centramos en la implementación en *python* de cada uno de los grupos estudiados en el capítulo anterior.

Una vez seleccionado el lenguaje de programación para implementar el proyecto, tuvimos que elegir las librerías y herramientas más idóneas para la escritura del código. Debido a que este trabajo trata imágenes, es fundamental tener métodos eficientes para manipularlas. Estuvimos haciendo pruebas con distintas librerías de python, entre ellas *matplotlib*, *pygame* y *pillow* (PIL). La primera de ellas está orientada a mostrar gráficas de funciones, y aunque es bastante versátil, no es la adecuada para manipular imágenes: simetrías, rotaciones, escalado, ... La segunda está orientada a juegos, y su instalación es engorrosa, así como su uso. No nos quedó claro que pudiese ser de utilidad directa en nuestro proyecto. La tercera sin embargo es una librería estándar que permite de forma nativa hacer las transformaciones de imágenes que nos pueden servir como primitivas para nuestras funciones. Estuvimos haciendo pruebas con estas librerías, y la que mejor se ajustaba al proyecto fue pillow. Aún así las rotaciones han sido difíciles de tratar, como ya comentamos más adelante.

Pillow permite almacenar las imágenes en una matriz. Cada entrada de la matriz se corresponde con un pixel de la imagen. Existen varias posibilidades de almacenamiento, y nosotros hemos usado RGBA, que añade a RGB (rojo, verde, azul) una cuarta entrada para el canal α (transparencias). Así, cada pixel se almacena como una tupla de cuatro elementos, y la posición del pixel en la imagen es la posición de éste en la matriz en la que se guarda. El canal α es fundamental para poder solapar imágenes.

Para facilitar la implementación, y pensando en la posibilidad de que otros usuarios pudieran utilizar en el futuro las funciones que hemos implementado,

creamos una clase llamada *Teselaciones*. En esta clase se incluye un método específico para cada grupo (Puntual, Frisos y Cristalográficos).

Se ha creado un repositorio público, alojado en github, donde está tanto el código como un tutorial sobre su uso. Estos documentos también están adjuntados en formato digital a esta memoria. El código está preparado para versiones de *python 3* o posteriores. No es difícil adaptarlo a python 2.7, el usuario interesado no encontrará problemas para hacerlo (las divisiones y otros pequeños detalles cambian de una versión a otra).

Las fuentes consultadas para la programación son las siguientes [15, 16, 17, 21, 22, 23].

2.1. La clase Teselaciones

El método `__init__` de la clase (inicialización), así como las librerías que usamos son las siguientes. Todas las funciones están documentadas el código, y por tanto al invocar a la ayuda de python obtendremos una breve descripción y, en algunos casos, ejemplos de uso (que se corresponde con el código que hay entre tres comillas dobles).

```
from PIL import Image
from math import sqrt
from math import floor
from sympy import *
class Teselaciones:
    """
    Functions that generate crystallographic group order.
       Functions have name as
    the name of the group and supported as input image and range
        .
    Example:
    >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
    >>> t=Teselaciones(imagen)
    >>> t.p1(2)

    """

    def __init__(self, im):
        self.image = im
```

A lo largo de las siguientes secciones mostraremos el código fuente de los métodos correspondientes a cada grupo, así como la salida que devuelven. Para la ejecución cargaremos la siguiente imagen como ejemplo para cada llamada al método.

```
>>> imagen = Image.open("RAIMUNDO.jpg")
>>> t=Teselaciones(imagen)
```

2.2. Grupo Puntual

En esta sección mostramos el código para los elementos generadores del grupo diédrico, simetrías y rotaciones.

Para la implementación de los distintos métodos de la clase, hemos tenido que crear varias funciones auxiliares. La primera es la función *mini()*, nos devuelve una miniatura de la imagen dada, encajada en un cuadrado, de tamaño 150x150 pixeles. Esto lo hacemos para que al crear una teselación de un rango alto, la cantidad de pixeles no sea demasiado elevada.

```
def mini(self):
        """
        Creates a 150x150 pixels thumbnail of the given image,
           and then squares it
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.mini()
        """
        imag=self.image
        imagaux=imag
        miniatura = (150, 150)
        imagaux.thumbnail(miniatura)
        if imagaux.size[0]>=imagaux.size[1]:
            caja=(0,0,imagaux.size[1],imagaux.size[1])
            cuadrado=imagaux.crop(caja)
        else:
            caja=(0,0,imagaux.size[0],imagaux.size[0])
            cuadrado=imagaux.crop(caja)
```

```
        return(cuadrado)
```

La llamada al método t.mini() produce la siguiente salida

2.2.1. Simetrías

Temenos las siguientes simetrías: horizontales, verticales. Hemos añadido también simetrías con deslizamiento.

El siguiente método nos devuelve una simetría vertical de la imagen con la que se ha inicializado la clase.

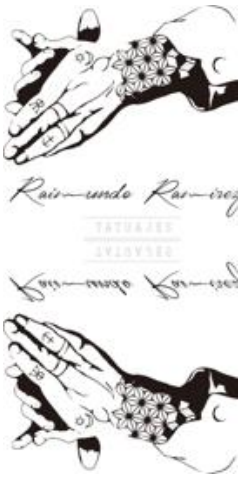
```
def SimVertical(self):
        """
        Creates a vertical symmetry of the given image, return
            image and symmetry image
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.SimVertical()
        """
        imag=self.mini()
        mp=imag.load()
        imagaux=Image.new("RGB",(imag.size[0]*2,imag.size[1]),"
            white")
        mpaux=imagaux.load()
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                mpaux[i,j]=mp[i,j]
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                mpaux[imag.size[0]+i,j]=mpaux[imag.size[0]-i-1,j
                    ]
        return(imagaux)
```

Llamamos al método t.SimVertical(), y produce

La siguiente función nos devuelve una simetría horizontal de la imagen (esta notación puede variar según el autor, pero con la imagen que mostramos queda claro lo que pretendemos).

```
def SimHorizontal(self):
        """
        Creates a horizontal symmetry of the given image, return
            image and symmetry image
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.SimHorizontal()
        """
        imag=self.mini()
        mp=imag.load()
        imagaux=Image.new("RGB",(imag.size[0],imag.size[1]*2),"
            white")
        mpaux=imagaux.load()
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                mpaux[i,j]=mp[i,j]
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                mpaux[i,imag.size[1]+j]=mpaux[i,imag.size[1]-j
                    -1]
        return(imagaux)
```

La salida de t.SimHorizontal() es



Mostramos ahora una simetría horizontal con deslizamiento.

```
def SimHoDesliz(self):
        """
        Creates a horizontal glide reflection of the given image
            , return image and symmetry image
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.SimHoDesliz()
        """
        imag=self.mini()
        mp=imag.load()
        imagaux=Image.new("RGB",(imag.size[0]*2,imag.size[1]*2),
            "white")
        mpaux=imagaux.load()
        imsim=Image.new("RGB",(imag.size[0],imag.size[1]),"white
            ")
        mpsim=imsim.load()
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                mpsim[i,j]=mp[i,imag.size[1]-j-1]
        imagaux.paste(imag, (0,0))
        imagaux.paste(imag, (0,imag.size[1]))
        imagaux.paste(imsim, (imag.size[0],0))
        imagaux.paste(imsim, (imag.size[0],imag.size[1]))
        return(imagaux)
```

Llamamos al método t.SimHoDesliz() y obtenemos la siguiente salida (la salida original serían las imágenes correspondientes a la diagonal, hemos añadido las otras dos por cuestiones estéticas)

La siguiente función nos devuelve una simetría vertical deslizante de la imagen.

```
def SimVeDesliz(self):
        """
        Creates a horizontal glide reflection of the given image
            , return image and symmetry image
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.SimVeDesliz()
        """
        imag=self.mini()
        mp=imag.load()
        imagaux=Image.new("RGB",(imag.size[0]*2,imag.size[1]*2),
            "white")
        mpaux=imagaux.load()
        imsim=Image.new("RGB",(imag.size[0],imag.size[1]),"white
            ")
        mpsim=imsim.load()
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                mpsim[i,j]=mp[imag.size[0]-i-1,j]
        imagaux.paste(imag, (0,0))
        imagaux.paste(imsim, (0,imag.size[1]))
        imagaux.paste(imag, (imag.size[0],0))
        imagaux.paste(imsim, (imag.size[0],imag.size[1]))
        return(imagaux)
```

Llamamos al método t.SimVeDesliz():

2.2.2. Rotaciones

En el caso de las rotaciones hemos usado la función que viene implementada en Pillow por defecto. Hemos creado varias funciones para rotar imágenes, pero ninguna de ellas nos daba un resultado mejor que la que contiene Pillow. El problema que surge con las rotaciones cuando se trabaja pixel a pixel, es que actuamos sobre posiciones y al hacer el producto por una matriz de rotación (con senos y cosenos) los números que obtenemos no son enteros en muchas ocasiones. Entonces es necesario hacer un pequeño redondeo de ajuste. Este hecho produce pequeñas variaciones en la imagen, sobre todo al trabajar con miniaturas. Si la imagen con la que trabajamos es grande, estos redondeos son inapreciables, pero en las miniaturas observamos ciertas imperfecciones y ruidos.

2.3. Grupos de Frisos

En esta sección describimos los métodos de la clase correspondientes a los grupos de frisos.

2.3.1. Friso F_1

El grupo F_1 se explicó en la Página 17. El código que lo genera es el siguiente.

```
def F1(self,rang):
        """
        Group F1
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.F1(2)
        """
        imag=self.mini()
        im1=Image.new("RGB",(imag.size[0]*rang,imag.size[1]),"
            white")
        for i in range(0,rang):
                im1.paste(imag, (i*imag.size[0],0))
        return(im1)
```

Llamamos al método t.F1(10), obteniendo diez copias consecutivas de la imagen desplazado horizontalmente:

2.3.2. Friso F_1^1

El grupo F_1^1 se explicó en la Página 17. El código asociado es el siguiente.

```
def F11(self,rang):
        """
        Group F11
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.F11(2)
        """
        imag=self.mini()
        imag=self.SimHorizontal()
        im1=Image.new("RGB",(imag.size[0]*rang,imag.size[1]),"
            white")
        for i in range(0,rang):
                im1.paste(imag, (i*imag.size[0],0))
        return(im1)
```

Al invocar t.F11(10), muestra lo siguiente

El rango (segundo argumento en la llamada al método) hace referencia al número de copias de la tesela, y no de la región fundamental. Es por eso que en la imagen anterior aparecen 20 copias de la imagen original (algunas simetrizadas). La tesela en este caso se corresponde con la primera de las 10 columnas de la imagen.

2.3.3. Friso F_1^2

El grupo F_1^2 se explicó en la Página 17, y nuestra implementación es la siguiente.

```
def F12(self,rang):
        """
        Group F12
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.F12(2)
        """
        imag=self.mini()
        imag=self.SimVertical()
        im1=Image.new("RGB",(imag.size[0]*rang,imag.size[1]),"
            white")
        for i in range(0,rang):
                im1.paste(imag, (i*imag.size[0],0))
        return(im1)
```

Llamamos al método t.F12(10)

2.3.4. Friso F_1^3

El grupo F_1^3 se explicó en la Página 17. El código que lo genera es el siguiente.

```
def F13(self,rang):
        """
        Group F13
```

```
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.F13(2)
        """
        imag=self.mini()
        imag=self.SimHoDesliz()
        im1=Image.new("RGB",(imag.size[0]*rang,imag.size[1]),"
            white")
        for i in range(0,rang):
                im1.paste(imag, (i*imag.size[0],0))
        return(im1)
```

Por ejemplo, t.F13(10) da como resultado:

2.3.5. Friso F_2

La definición del grupo F_2 la dimos en la Página 17. Mostramos a continuación su código asociado.

```
def F2(self,rang):
        """
        Group F2
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.F2(2)
        """
        imag=self.mini()
        imag=self.equilatero()
        imagr=imag.rotate(180)
        im1=Image.new("RGBA",(imag.size[0]*2,imag.size[1])
            ,(255,255,255,0))
        im1.paste(imag, (0,0))
        im1.paste(imagr, (floor(imag.size[0]/2),0),imagr)
        im2=Image.new("RGBA",(imag.size[0]*rang*2,imag.size[1])
            ,(255,255,255,0))
        for i in range(0,rang):
                im2.paste(im1, (i*imag.size[0],0),im1)
        return(im2)
```

Llamamos al método t.F2(10), obteniendo

2.3.6. Friso F_2^1

El grupo F_2^1 se explicó en la Página 18. El código que lo genera es el siguiente.

```
def F21(self,rang):
        """
        Group F21
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.F21(2)
        """
        imag=self.mini()
        imag=self.SimHorizontal()
        imagr=imag.rotate(180)
        im1=Image.new("RGB",(imag.size[0]*2,imag.size[1]),"white
            ")
        im1.paste(imag, (0,0))
        im1.paste(imagr, (imag.size[0],0))
        im2=Image.new("RGB",(imag.size[0]*rang*2,imag.size[1]),"
            white")
        for i in range(0,rang):
                im2.paste(im1, (i*im1.size[0],0))
        return(im2)
```

Llamamos al método t.F21(10)

2.3.7. Friso F_2^2

En la Página 18 dimos la definición de este grupos. El código que lo genera es el siguiente.

```
def F22(self,rang):
        """
        Group F22
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.F21(2)
        """
        imag=self.mini()
        imagr=imag.rotate(180)
        imagrs=Teselaciones(imagr).SimV()
        imags=self.SimV()
        im1=Image.new("RGB",(imag.size[0]*2,imag.size[1]*2),"
            white")
        im1.paste(imag, (0,0))
        im1.paste(imagr, (0,imag.size[1]))
        im1.paste(imags, (imag.size[0],0))
        im1.paste(imagrs,(imag.size[0],imag.size[1]))
        im2=Image.new("RGB",(imag.size[0]*rang*2,imag.size[1]*2)
            ,"white")
        for i in range(0,rang):
                im2.paste(im1, (i*im1.size[0],0))
        return(im2)
```

Si ejecutamos t.F22(10), obtenemos

2.4. Grupos Cristalográficos

En esta sección veremos los métodos utilizados para la generación de teselaciones bajo la acción de un grupo cristalográfico.

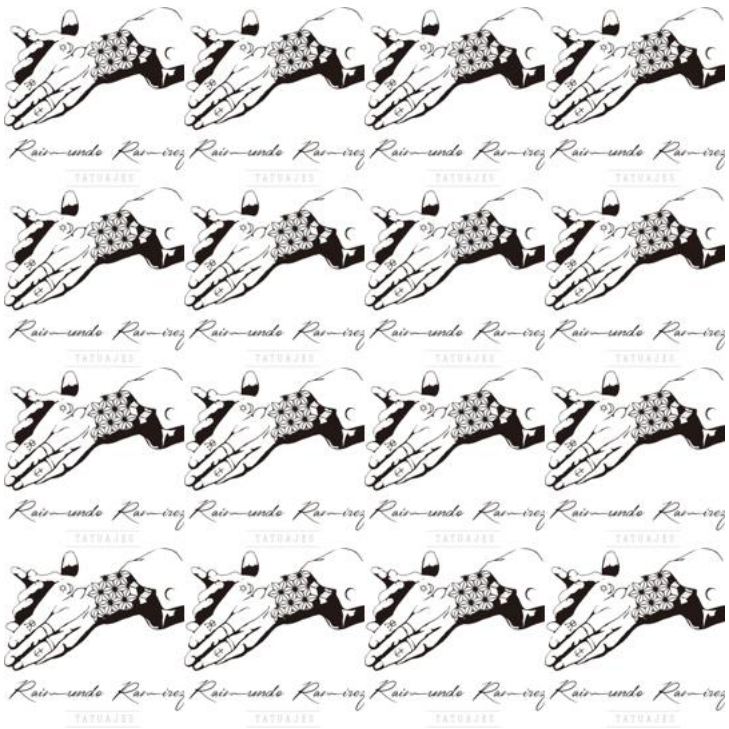
2.4.1. Grupo P1

El grupo $p1$ se explicó en la página 22. El código que lo genera es el siguiente.

```
def p1(self,rang):
        """
        p1 Group containing two linearly independent
            translations
```

```
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p2(2)
        """
        imag=self.mini()
        im1=Image.new("RGB",(imag.size[0]*rang,imag.size[1]*rang
            ),"white")
        for i in range(0,rang):
            for j in range(0,rang):
                im1.paste(imag, (i*imag.size[0],j*imag.size[1]))
        return(im1)
```

Llamamos al método t.p1(4) (en este caso el rango el orden de la matriz, en este caso nos da una matriz 4×4)



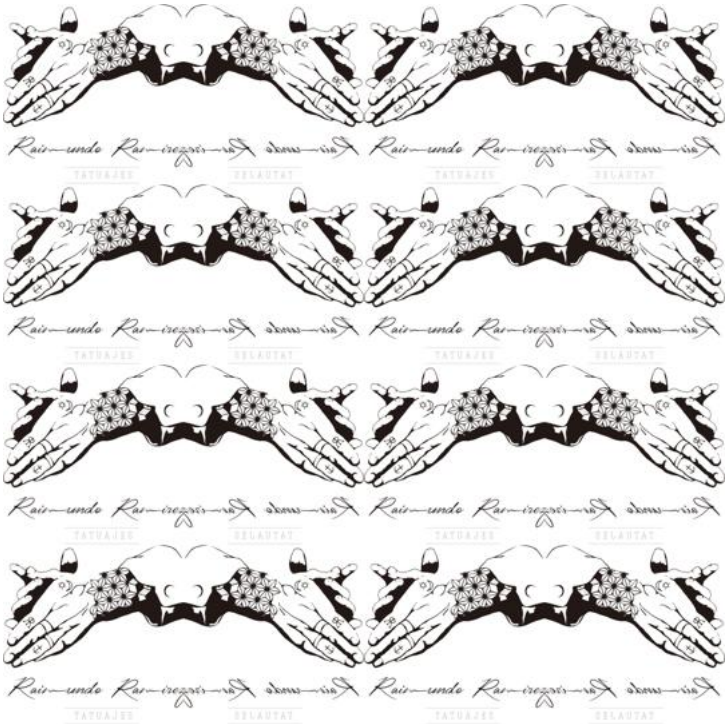
2.4.2. Grupo Pm

La explicación del grupo *pm* podemos encontrarla en la página 23. El código que lo genera es el siguiente.

```
def pm(self,rang):
        """
        Group pm
```

```
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.pm(2)
        """
        imag=self.mini()
        im1=Image.new("RGB",(imag.size[0]*rang*2,imag.size[1]*
            rang*2),"white")
        im2=self.SimVertical()
        for i in range(0,rang):
            for j in range(0,rang*2):
                im1.paste(im2, (i*2*imag.size[0],j*imag.size[1])
                    )
        return(im1)
```

Al ejecutar el método t.pm(2) obtenemos la siguiente imagen.



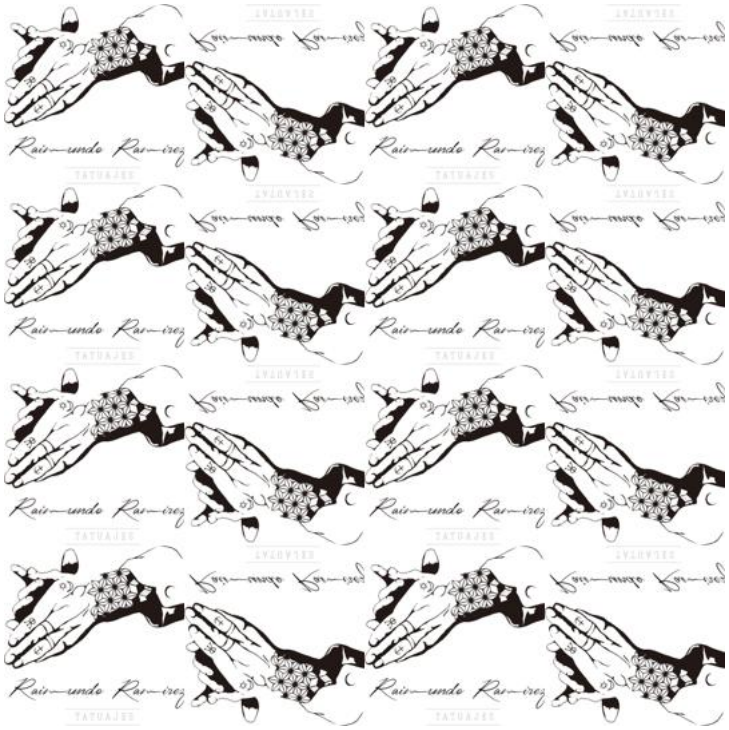
2.4.3. Grupo Pg

El grupo *pg* y su definición se encuentra en la Página 23. Mostramos el código a continuación.

```
def pg(self,rang):
        """
```

```
        Group pg
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.pg(2)
        """
        imag=self.mini()
        im1=Image.new("RGB",(imag.size[0]*rang*2,imag.size[1]*
            rang*2),"white")
        im2=self.SimHoDesliz()
        for i in range(0,rang*2):
            for j in range(0,rang*2):
                im1.paste(im2, (i*2*imag.size[0],j*2*imag.size
                    [1]))
        return(im1)
```

Llamamos al método t.pg(2)



2.4.4. Grupo Cm

El grupo *cm* lo encontramos en la página 23. El código que lo genera es el siguiente.

```
def cm(self,rang):
        """
```

```
Group cm
Example:
>>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
>>> t=Teselaciones(imagen)
>>> t.cm(2)
"""
imag=self.mini()
imagaux=Image.new("RGB",(imag.size[0]*2,imag.size[1]*2),
    "white")
im1=Image.new("RGB",(imag.size[0]*2*rang,imag.size[1]*2*
    rang),"white")
imsim=self.SimV()
imsimH=self.SimH()
rot = imag.rotate(180)
imagaux.paste(imag, (0,0))
imagaux.paste(imsimH, (0,imag.size[1]))
imagaux.paste(rot, (imag.size[0],0))
imagaux.paste(imsim, (imag.size[0],imag.size[1]))
for i in range(0,rang*2):
    for j in range(0,rang*2):
        im1.paste(imagaux, (i*2*imag.size[0],j*2*imag.
            size[1]))
return(im1)
```

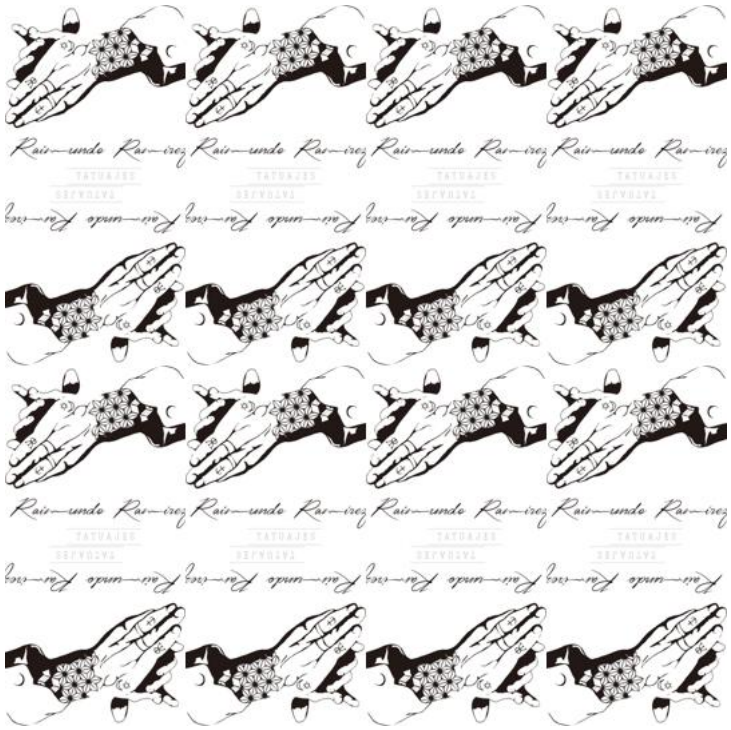
Ejecutamos el método t.cm(2) de la siguiente manera.

2.4.5. Grupo P2

El grupo de rotaciones de orden 2, $p2$, se explica en la página 24. El código que lo genera es el siguiente.

```
def p2(self,rang):
        """
        Group containing two linearly independent translations
            and rotations of order two.
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p2(2)
        """
        imag=self.mini()
        im1=Image.new("RGB",(imag.size[0]*rang*2,imag.size[1]*
            rang*2),"white")
        im2=Image.new("RGB",(imag.size[0],imag.size[1]*2),"white
            ")
        rot = imag.rotate(180)
        im2.paste(imag, (0,0))
        im2.paste(rot, (0,imag.size[1]))
        for i in range(0,rang*2):
            for j in range(0,rang):
                im1.paste(im2, (i*imag.size[0],j*2*imag.size[1])
                    )
        return(im1)
```

Llamamos al método t.p2(2)



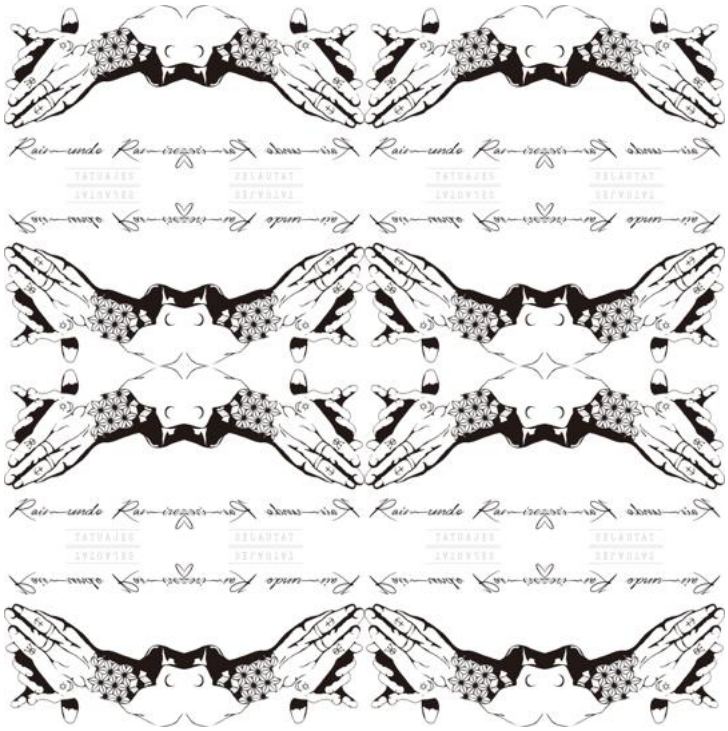
2.4.6. Grupo Pmm

Podemos encontrar información sobre el grupo *pmm* en la página 24. El código que lo genera es el siguiente.

```
def pmm(self,rang):
        """
        Group pmm
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.pmm(2)
        """
        imag=self.mini()
        imagaux=Image.new("RGB",(imag.size[0]*2,imag.size[1]*2),
            "white")
        im1=Image.new("RGB",(imag.size[0]*2*rang,imag.size[1]*2*
            rang),"white")
        imsim=self.SimV()
        imsimH=self.SimH()
        rot = imag.rotate(180)
        imagaux.paste(imag, (0,0))
        imagaux.paste(imsimH, (0,imag.size[1]))
        imagaux.paste(imsim, (imag.size[0],0))
        imagaux.paste(rot, (imag.size[0],imag.size[1]))
```

```
        for i in range(0,rang*2):
            for j in range(0,rang*2):
                im1.paste(imagaux, (i*2*imag.size[0],j*2*imag.
                    size[1]))
        return(im1)
```

Llamamos al método t.pmm(2)



2.4.7. Grupo Cmm

Para generar el grupo cmm, necesitamos que la región fundamental sea un triángulo, para poder teselar con las simetrías que requiere. Para ello generamos el siguiente método.

```
def triangulo(self):
        """
        Creates a triangle of the image.
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.triangulo(2)
        """
        imag=self.image
        mp=imag.load()
```

```
        imagaux=Image.new("RGBA",(imag.size[0],imag.size[1]),"
            white")
        mpaux=imagaux.load()
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                if j>=(-2*imag.size[1]/imag.size[0])*i+imag.size
                    [1] and j>=(2*imag.size[1]/imag.size[0])*i-
                    imag.size[1]:
                    mpaux[i,j]=mp[i,j]
                else:
                    mpaux[i,j]=(255,255,255,0)
        return(imagaux)
```

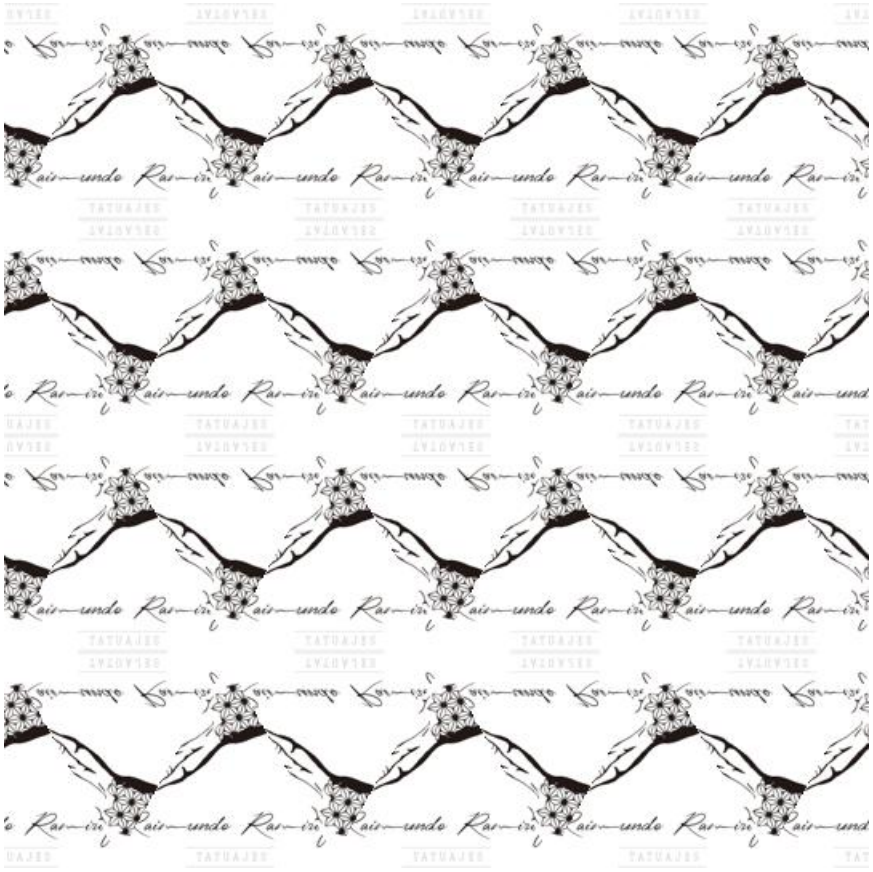
Que nos proporciona la siguiente transformación cuando ejecutamos t.triangulo().

Una vez tenemos la región fundamental, mostramos el código del grupo *cmm*. La información del grupo la obtenemos en la página 24.

```
def cmm(self,rang):
        """
        Group cmm
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.cmm(2)
        """
        imag=self.mini()
        mp=imag.load()
        imagaux=self.triangulo()
        mpaux=imagaux.load()
        im1=Image.new("RGBA",(imag.size[0]*2,imag.size[1]*2),"
            white")
        fin=Image.new("RGB",(imag.size[0]*2*rang,imag.size[1]*2*
            rang),"white")
        mpf=fin.load()
        mpim1=im1.load()
        imsim=Image.new("RGBA",(imag.size[0],imag.size[1]),"
            white")
        mpsim=imsim.load()
        s=Image.new("RGBA",(imag.size[0],imag.size[1]),"white")
        mps=s.load()
        for i in range(imag.size[0]):
```

```
            for j in range(imag.size[1]):
                mpsim[i,j]=mpaux[i,imag.size[1]-j-1]
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                mps[i,j]=mpsim[i,imag.size[1]-j-1]
        im1.paste(imagaux, (0,0))
        im1.paste(imsim, (floor(imag.size[0]/2),0), imsim)
        im1.paste(imsim, (0,imag.size[1]))
        im1.paste(s, (floor(imag.size[0]/2),imag.size[1]),s)
        for i in range(-1,rang*2):
            for j in range(0,rang*2):
                fin.paste(im1, (i*imag.size[0],j*2*imag.size[1])
                    ,im1)
        return(fin)
```

Llamamos al método t.cmm(4)



2.4.8. Grupo Pmg

El grupo *pmg* se encuentra en la página 25. Veamos el código que lo genera.

```
def pmg(self,rang):
        """
        Group pmg
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.pmg(2)
        """
        imag=self.mini()
        imagaux=Image.new("RGB",(imag.size[0]*2,imag.size[1]*2),
            "white")
        im1=Image.new("RGB",(imag.size[0]*2*rang,imag.size[1]*2*
            rang),"white")
        imsim=self.SimV()
        imsimH=self.SimH()
        rot = imag.rotate(180)
        imagaux.paste(imag, (0,0))
        imagaux.paste(imsimH, (0,imag.size[1]))
        imagaux.paste(rot, (imag.size[0],0))
        imagaux.paste(imsim, (imag.size[0],imag.size[1]))
        for i in range(0,rang*2):
            for j in range(0,rang*2):
                im1.paste(imagaux, (i*2*imag.size[0],j*2*imag.
                    size[1]))
        return(im1)
```

Ejecutamos el método t.pmg(2)

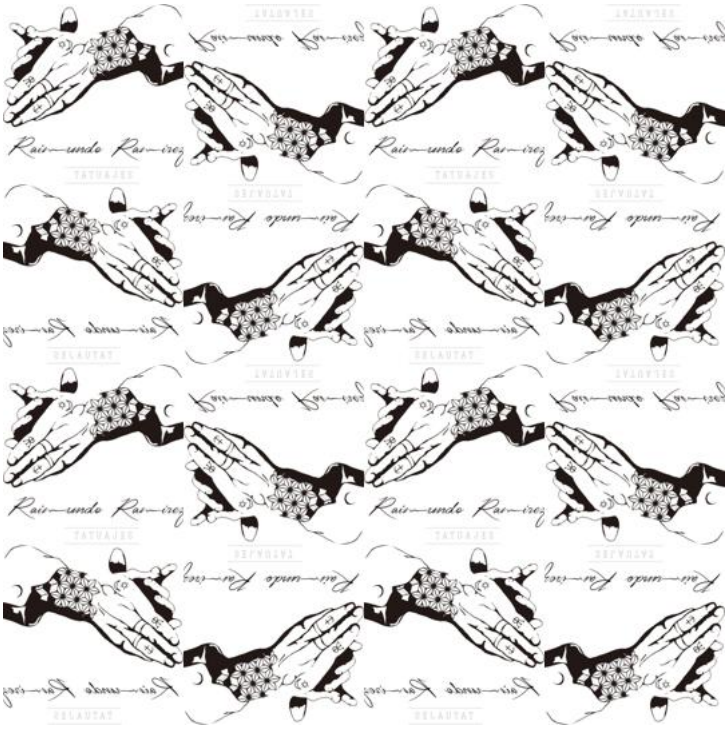
2.4.9. Grupo Pgg

Encontramos el grupo *pgg* en la página 25. Tenemos para generarlo el siguiente código.

```
def pgg(self,rang):
        """
        Group pgg
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.pgg(2)
        """
        imag=self.mini()
        imagaux=Image.new("RGB",(imag.size[0]*2,imag.size[1]*2),
            "white")
        im1=Image.new("RGB",(imag.size[0]*2*rang,imag.size[1]*2*
            rang),"white")
        imsim=self.SimV()
        imsimH=self.SimH()
```

```
        rot = imag.rotate(180)
        imagaux.paste(imag, (0,0))
        imagaux.paste(imsim, (0,imag.size[1]))
        imagaux.paste(imsimH, (imag.size[0],0))
        imagaux.paste(rot, (imag.size[0],imag.size[1]))
        for i in range(0,rang*2):
            for j in range(0,rang*2):
                im1.paste(imagaux, (i*2*imag.size[0],j*2*imag.
                    size[1]))
        return(im1)
```

Llamamos al método t.pgg(2)



2.4.10. Grupo P3

Habíamos creado un método que nos devolvía un triángulo de la imagen insertada. Pero para generar el grupo p3 necesitamos que la región fundamental sea un triángulo equilátero, luego generamos el siguiente método.

```
def equilatero(self):
        """
        Creates a equilater triangle of the image.
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
```

```
        >>> t.triangulo(2)
        """

        imag=self.mini()
        caja = (0, imag.size[1]-floor(sqrt(imag.size[1]**2-(imag
            .size[1]/2)**2)), imag.size[0], imag.size[1])
        modif = imag.crop(caja)
        tmod=Teselaciones(modif).triangulo()
        return(tmod)
```

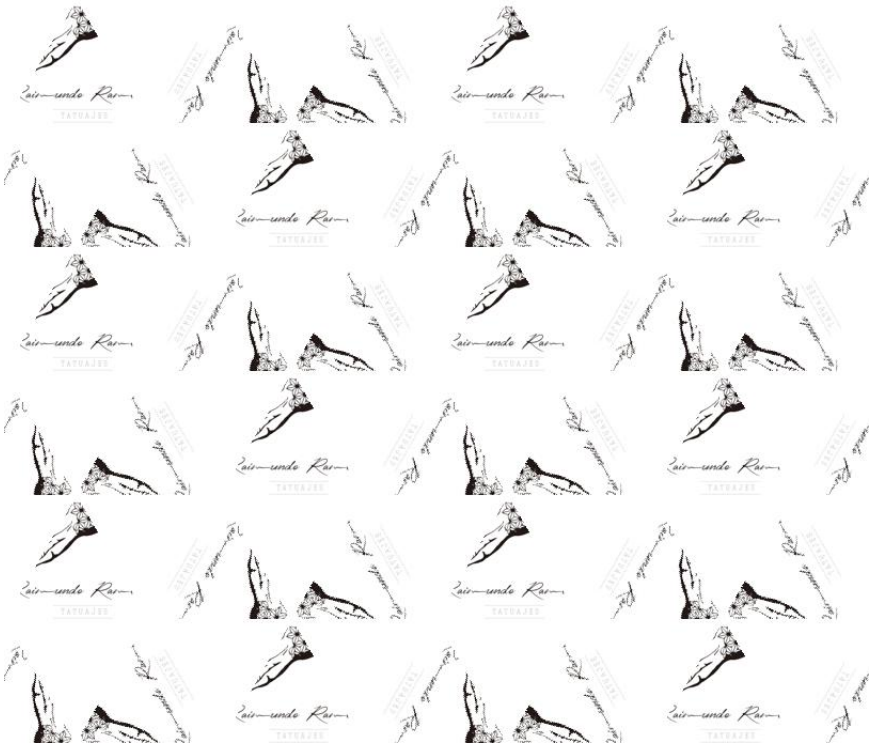
Lo ejecutamos como t.equilatero().

Estamos en condiciones de ver ahora el código que genera el grupo $p3$. Este se explicó en la Página 26.

```
def p3(self,rang):
        """
        Group p3
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p3(2)
        """
        imag=self.mini()
        mp=imag.load()
        imagaux=self.equilatero()
        mpaux=imagaux.load()
        im1=Image.new("RGBA",(imagaux.size[0]*3,imagaux.size
            [1]*3),(255,255,255,0))
        fin=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
            size[1]*3*rang),(255,255,255,0))
        fin2=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
            size[1]*3*rang),(255,255,255,0))
        mpf=fin.load()
        mpim1=im1.load()
        imsim=Image.new("RGBA",(imagaux.size[0],imagaux.size[1])
            ,"white")
        mpsim=imsim.load()
        s=Image.new("RGBA",(imagaux.size[0],imagaux.size[1]),"
            white")
        mps=s.load()
```

```
s2=Image.new("RGBA",(imagaux.size[0],imagaux.size[1]),"
    white")
mps2=s2.load()
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mpsim[i,j]=mpaux[i,imagaux.size[1]-j-1]
rc=imsim.rotate(-120, expand=1)
caja=(rc.size[0]-imagaux.size[0],rc.size[1]-imagaux.size
    [1],rc.size[0],rc.size[1])
rc2=rc.crop(caja)
mprc2=rc2.load()
rccc=imagaux.rotate(-120, expand=1)
caja3=(0,0,imagaux.size[0],imagaux.size[1])
rccc2=rccc.crop(caja3)
mprccc2=rccc2.load()
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mps[i,j]=mprc2[i,imagaux.size[1]-j-1]
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mps2[i,j]=mprccc2[i,imagaux.size[1]-j-1]
im1.paste(imagaux, (0,0))
im1.paste(s, (floor(imagaux.size[0]/2),imagaux.size[1]),
    s)
im1.paste(rccc2, (imagaux.size[0],0),rccc2)
for i in range(-2,rang*2):
    for j in range(-2,rang*2):
        fin.paste(im1, (i*3*imagaux.size[0]-i,j*2*
            imagaux.size[1]),im1)
for i in range(-2,rang*2):
    for j in range(-2,rang*2):
        fin2.paste(im1, (i*3*imagaux.size[0]-imagaux.
            size[0]-floor((1/2)*imagaux.size[0])-i,j*2*
            imagaux.size[1]-imagaux.size[1]),im1)
fin2.paste(fin, fin)
return(fin2)
```

Llamamos al método t.p3(2)



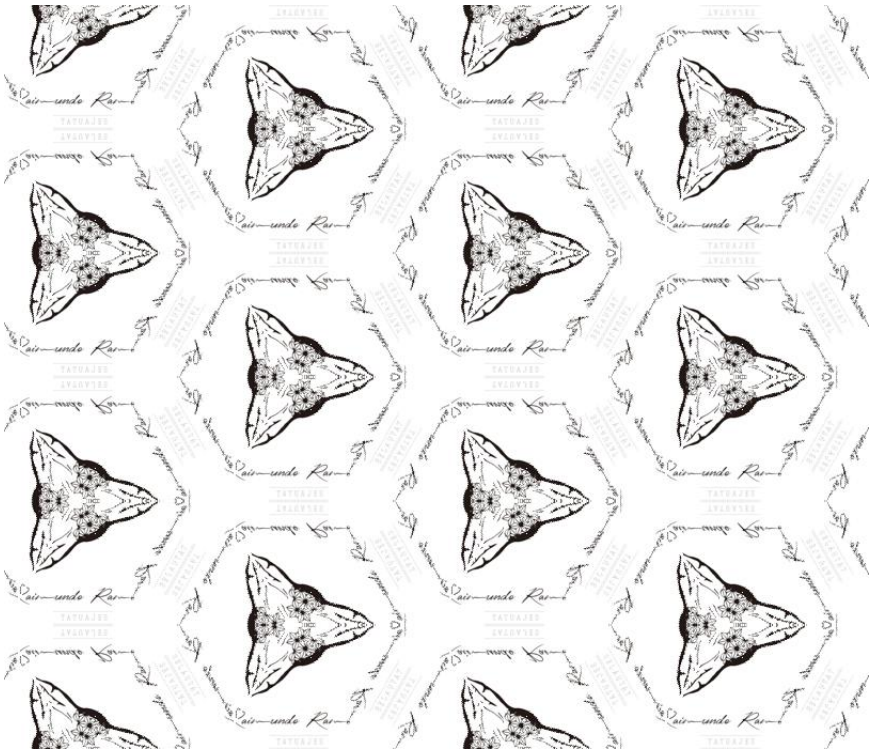
2.4.11. Grupo P31m

El grupo $p31m$ se explicó en la Página 26. El código que lo genera es el siguiente.

```
def p31m(self,rang):
        """
        Group p31m
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p31m(2)
        """
        imagaux=self.equilatero()
        mpaux=imagaux.load()
        im1=Image.new("RGBA",(imagaux.size[0]*3,imagaux.size
            [1]*3),(255,255,255,0))
        fin=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
            size[1]*3*rang),(255,255,255,0))
        fin2=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
            size[1]*3*rang),(255,255,255,0))
        mpf=fin.load()
        mpim1=im1.load()
```

```
imsim=Image.new("RGBA",(imagaux.size[0],imagaux.size[1])
    ,"white")
mpsim=imsim.load()
s=Image.new("RGBA",(imagaux.size[0],imagaux.size[1]),"
    white")
mps=s.load()
s2=Image.new("RGBA",(imagaux.size[0],imagaux.size[1]),"
    white")
mps2=s2.load()
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mpsim[i,j]=mpaux[i,imagaux.size[1]-j-1]
rc=imsim.rotate(-120, expand=1)
caja=(rc.size[0]-imagaux.size[0],rc.size[1]-imagaux.size
    [1],rc.size[0],rc.size[1])
rc2=rc.crop(caja)
mprc2=rc2.load()
rccc=imagaux.rotate(-120, expand=1)
caja3=(0,0,imagaux.size[0],imagaux.size[1])
rccc2=rccc.crop(caja3)
mprccc2=rccc2.load()
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mps[i,j]=mprc2[i,imagaux.size[1]-j-1]
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mps2[i,j]=mprccc2[i,imagaux.size[1]-j-1]
im1.paste(imagaux, (0,0))
im1.paste(rc2, (floor(imagaux.size[0]/2),0),rc2)
im1.paste(imsim, (0,imagaux.size[1]))
im1.paste(s, (floor(imagaux.size[0]/2),imagaux.size[1]),
    s)
im1.paste(rccc2, (imagaux.size[0]-2,0),rccc2)
im1.paste(s2,(imagaux.size[0]-2,imagaux.size[1]),s2)
for i in range(-2,rang*2):
    for j in range(-2,rang*2):
        fin.paste(im1, (i*3*imagaux.size[0]-i,j*2*
            imagaux.size[1]),im1)
for i in range(-2,rang*2):
    for j in range(-2,rang*2):
        fin2.paste(im1, (i*3*imagaux.size[0]-imagaux.
            size[0]-floor((1/2)*imagaux.size[0])-i,j*2*
            imagaux.size[1]-imagaux.size[1]),im1)
fin2.paste(fin, fin)
return(fin2)
```

Llamamos al método t.p31m(2)



2.4.12. Grupo P3m1

Necesitamos ajustar la región fundamental a un triángulo dividido en tres partes iguales, con centro en el circuncentro, para ello desarrollamos los siguientes métodos.

```
def fundam(self):
        """
        divide a triangle in three same parts, an return one of
            them.
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.fundam(2)
        """
        im=self.equilatero()
        mp=im.load()
        imaux=Image.new("RGBA",(im.size[0],im.size[1]),"white")
        mpaux=imaux.load()
        for i in range(im.size[0]):
            for j in range(im.size[1]):
                if j<=((-2/3)*im.size[1]/im.size[0])*i+im.size
                    [1] and i<im.size[0]/2:
                    mpaux[i,j]=mp[i,j]
```

```
                else:
                    mpaux[i,j]=(255,255,255,0)
        return(imaux)
```

Llamamos al método t.fundam()

```
def fundam2(self):
        """
        Divide a triangle in three same parts.
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.fundam2(2)
        """
        pp=self.fundam()
        pp2=pp.rotate(-120, expand=1)
        pp3=pp.rotate(120, expand=1)
        ppaux=Image.new("RGBA",(pp3.size[0],pp3.size[1]),"white"
            )
        ppaux.paste(pp, (0,0))
        ppaux.paste(pp2, (0,0),pp2)
        ppaux.paste(pp3, (-floor(pp.size[0]/4)-1,0),pp3)
        aux=self.equilatero()
        caja=(0,0,aux.size[0],aux.size[1])
        ppaux2=ppaux.crop(caja)
        return(ppaux2)
```

Llamamos al método t.fundam2()

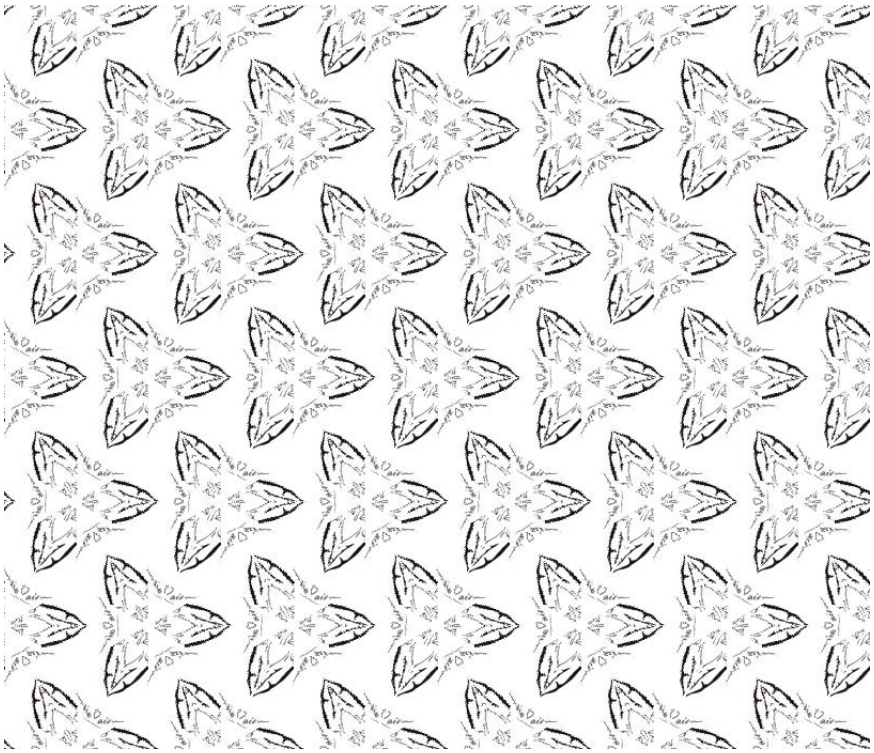
El grupo $p3m1$ lo ubicamos en la página 26. El código que lo genera es el siguiente.

```
def p3m1(self,rang):
        """
        Group p3m1
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
```

```
>>> t.p3m1(2)
"""
imag=self.mini()
imagaux=self.fundam2()
mpaux=imagaux.load()
im1=Image.new("RGBA",(imagaux.size[0]*3,imagaux.size
    [1]*3),(255,255,255,0))
fin=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
    size[1]*3*rang),(255,255,255,0))
fin2=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
    size[1]*3*rang),(255,255,255,0))
mpf=fin.load()
mpim1=im1.load()
imsim=Image.new("RGBA",(imagaux.size[0],imagaux.size[1])
    ,"white")
mpsim=imsim.load()
s=Image.new("RGBA",(imagaux.size[0],imagaux.size[1]),"
    white")
mps=s.load()
s2=Image.new("RGBA",(imagaux.size[0],imagaux.size[1]),"
    white")
mps2=s2.load()
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mpsim[i,j]=mpaux[i,imagaux.size[1]-j-1]
rc=imsim.rotate(-120, expand=1)
caja=(rc.size[0]-imagaux.size[0],rc.size[1]-imagaux.size
    [1],rc.size[0],rc.size[1])
rc2=rc.crop(caja)
mprc2=rc2.load()
rccc=imagaux.rotate(-120, expand=1)
caja3=(0,0,imagaux.size[0],imagaux.size[1])
rccc2=rccc.crop(caja3)
mprccc2=rccc2.load()
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mps[i,j]=mprc2[i,imagaux.size[1]-j-1]
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mps2[i,j]=mprccc2[i,imagaux.size[1]-j-1]
im1.paste(imagaux, (0,0))
im1.paste(rc2, (floor(imagaux.size[0]/2)-1,0),rc2)
im1.paste(rc2, (0,imagaux.size[1]),rc2)
im1.paste(imagaux, (floor(imagaux.size[0]/2),imagaux.
    size[1]),imagaux)
im1.paste(imagaux, (imagaux.size[0],0),imagaux)
im1.paste(rc2,(imagaux.size[0]-1,imagaux.size[1]),rc2)
for i in range(-2,rang*2):
    for j in range(-2,rang*2):
```

```
                fin.paste(im1, (i*3*imagaux.size[0],j*2*imagaux.
                    size[1]),im1)
        for i in range(-2,rang*2):
            for j in range(-2,rang*2):
                fin2.paste(im1, (i*3*imagaux.size[0]-imagaux.
                    size[0]-floor((1/2)*imagaux.size[0]),j*2*
                    imagaux.size[1]-imagaux.size[1]),im1)
        fin2.paste(fin, fin)
        return(fin2)
```

Llamamos al método t.p3m1(3)



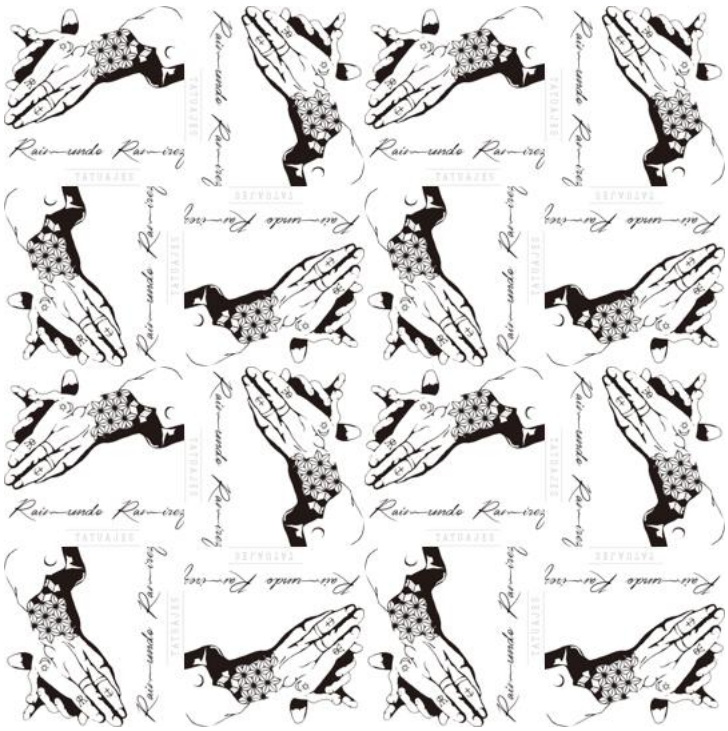
2.4.13. Grupo P4

El grupo de rotaciones de orden 4 es el grupo $p4$, se explicó en la página 27. Veamos el código que lo genera.

```
def p4(self,rang):
        """
        Group p4
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p4(2)
        """
```

```
        imag=self.mini()
        imagaux=Image.new("RGB",(imag.size[0]*2,imag.size[1]*2),
            "white")
        im1=Image.new("RGB",(imag.size[0]*2*rang,imag.size[1]*2*
            rang),"white")
        rot1 = imag.rotate(270)
        rot2 = imag.rotate(180)
        rot3 = rot1.rotate(180)
        imagaux.paste(imag, (0,0))
        imagaux.paste(rot3, (0,imag.size[1]))
        imagaux.paste(rot1, (imag.size[0],0))
        imagaux.paste(rot2, (imag.size[0],imag.size[1]))
        for i in range(0,rang*2):
            for j in range(0,rang*2):
                im1.paste(imagaux, (i*2*imag.size[0],j*2*imag.
                    size[1]))
        return(im1)
```

Llamamos al método t.p4(2)



2.4.14. Grupo P4m

Ajustamos la región fundamental con el siguiente método.

```
def divi1(self):
```

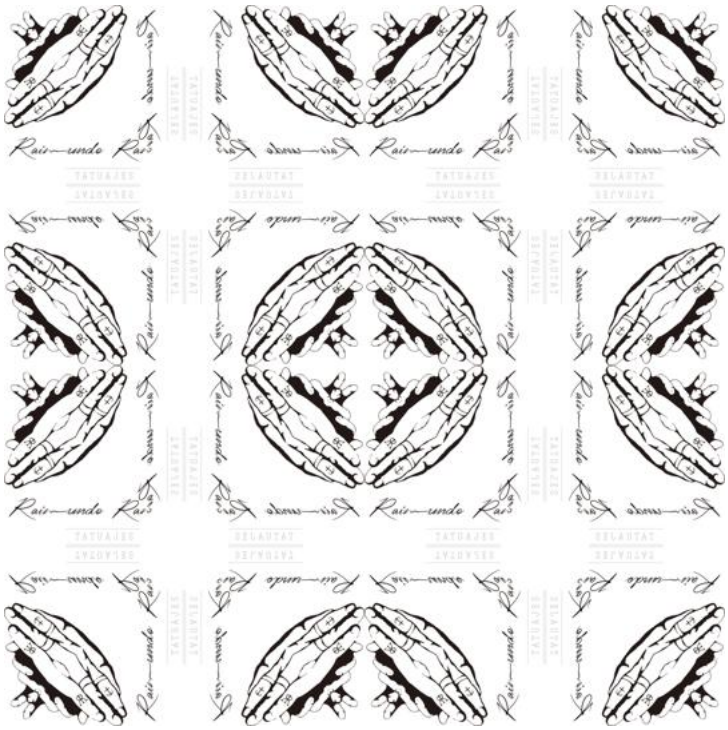
```
        """
        Divide image in diagonal
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.divi1(2)
        """
        imag=self.mini()
        mp=imag.load()
        imagaux=Image.new("RGBA",(imag.size[0],imag.size[1]),"
            white")
        mpaux=imagaux.load()
        rot=imag.rotate(90)
        srot=self.SimV()
        rot2=srot.rotate(90)
        mprot2=rot2.load()
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
                if j>=(imag.size[1]/imag.size[0])*i:
                    mpaux[i,j]=mp[i,j]
                else:
                    mpaux[i,j]=mprot2[i,j]
        return(imagaux)
```

Obtenemos en la ejecución t.divi1(2) la siguiente transformación.

Veamos ahora el grupo *p4mm*, lo ubicamos en la página 27. El código que lo genera es el siguiente.

```
def p4mm(self,rang):
        """
        Group p4mm
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p4mm(2)
        """
        imag=self.mini()
        aux=self.divi1()
        im1=Teselaciones(aux).p4(rang)
        return(im1)
```

Llamamos al método t.p4mm(2)



2.4.15. Grupo P4g

Para generar este grupo también necesitamos ajustar la región fundamental. Lo hacemos con una simetría inversa a la del grupo anterior. El método es el siguiente.

```
def divi2(self):
        """
        Divide image in diagonal
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.divi2(2)
        """
        imag=self.mini()
        mp=imag.load()
        imagaux=Image.new("RGBA",(imag.size[0],imag.size[1]),"
            white")
        mpaux=imagaux.load()
        rot=imag.rotate(-90)
        srot=self.SimV()
        rot2=srot.rotate(-90)
        mprot2=rot2.load()
        for i in range(imag.size[0]):
            for j in range(imag.size[1]):
```

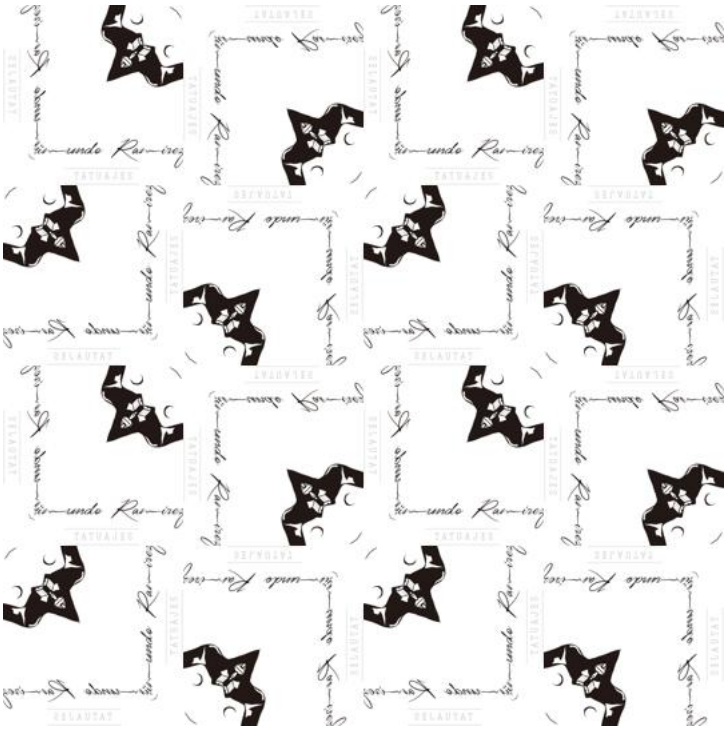
```
                if j>=(-imag.size[1]/imag.size[0])*i+imag.size
                    [1]:
                    mpaux[i,j]=mp[i,j]
                else:
                    mpaux[i,j]=mprot2[i,j]
        return(imagaux)
```

Llamamos al método t.divi(2)

Por tanto generamos el grupo *p4g* con el siguiente código. La información al respecto del grupo la encontramos en la página 28.

```
def p4g(self,rang):
        """
        Group p4g
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p4g(2)
        """
        imag=self.mini()
        aux=self.divi2()
        im1=Teselaciones(aux).p4(rang)
        return(im1)
```

Llamamos al método t.p4g(2)



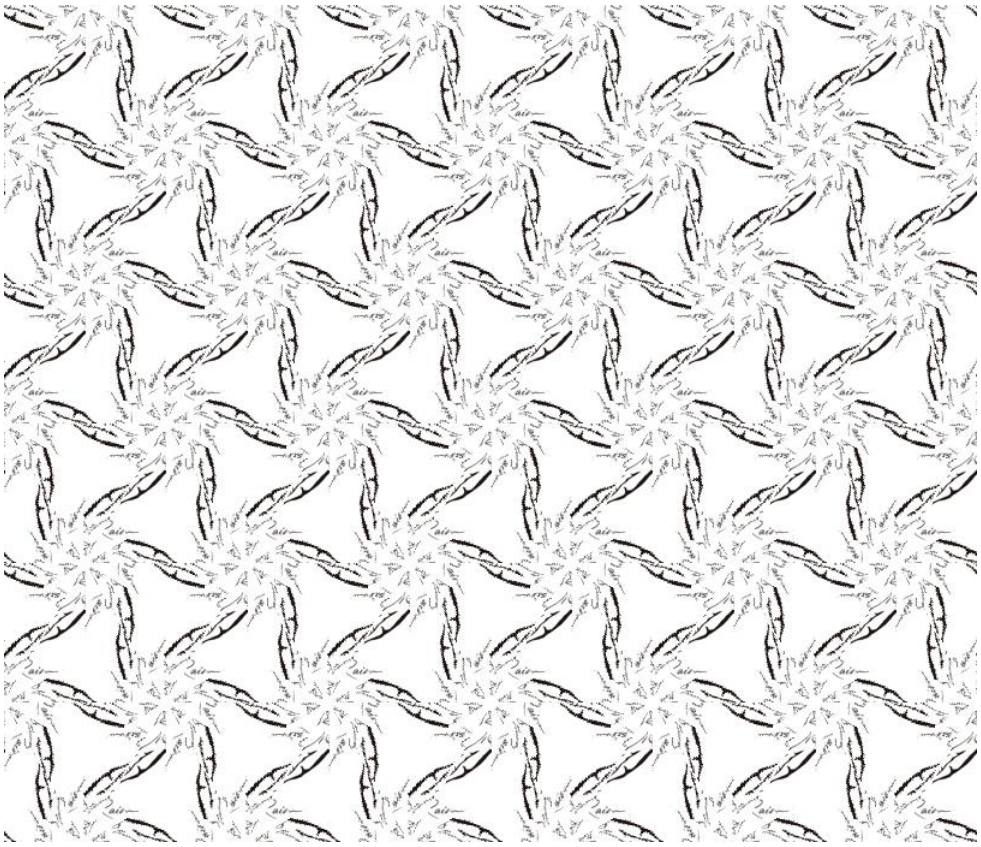
2.4.16. Grupo P6

El grupo *p6* se explicó en la Página 28. El código que lo genera es el siguiente.

```
def p6(self,rang):
        """
        Group p6
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p6(2)
         """
        imag=self.mini()
        imagaux=self.fundam2()
        mpaux=imagaux.load()
        im1=Image.new("RGBA",(imagaux.size[0]*3,imagaux.size
            [1]*3),(255,255,255,0))
        fin=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
            size[1]*3*rang),(255,255,255,0))
        fin2=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
            size[1]*3*rang),(255,255,255,0))
        mpf=fin.load()
        mpim1=im1.load()
```

```
imsim=Image.new("RGBA",(imagaux.size[0],imagaux.size[1])
    ,"white")
mpsim=imsim.load()
s=Image.new("RGBA",(imagaux.size[0],imagaux.size[1]),"
    white")
mps=s.load()
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mpsim[i,j]=mpaux[i,imagaux.size[1]-j-1]
rc=imsim.rotate(-120, expand=1)
caja=(rc.size[0]-imagaux.size[0],rc.size[1]-imagaux.size
    [1],rc.size[0],rc.size[1])
rc2=rc.crop(caja)
mprc2=rc2.load()
for i in range(imagaux.size[0]):
    for j in range(imagaux.size[1]):
        mps[i,j]=mprc2[imagaux.size[0]-i-1,j]
im1.paste(imagaux, (0,0))
im1.paste(s, (floor(imagaux.size[0]/2),0),s)
im1.paste(s, (0,imagaux.size[1]),s)
im1.paste(imagaux, (floor(imagaux.size[0]/2),imagaux.
    size[1]),imagaux)
im1.paste(imagaux, (imagaux.size[0],0),imagaux)
im1.paste(s,(imagaux.size[0],imagaux.size[1]),s)
for i in range(-2,rang*2):
    for j in range(-2,rang*2):
        fin.paste(im1, (i*3*imagaux.size[0],j*2*imagaux.
            size[1]),im1)
for i in range(-2,rang*2):
    for j in range(-2,rang*2):
        fin2.paste(im1, (i*3*imagaux.size[0]-imagaux.
            size[0]-floor((1/2)*imagaux.size[0]),j*2*
            imagaux.size[1]-imagaux.size[1]),im1)
fin2.paste(fin, fin)
return(fin2)
```

Ejecutamos el método t.p6(3)



2.4.17. Grupo P6m

Para generar el grupo *p6m* necesitamos dos métodos auxiliares previos. El primero de ellos divide un triangulo en seis partes iguales y devuelve una de ellas. El segundo método simetriza estas partes por cada uno de los laterales.

```
def fundam3(self):
    """
    divide a triangle in six same parts, an return one of them.
    Example:
    >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
    >>> t=Teselaciones(imagen)
    >>> t.fundam3(2)

    """
    im=self.equilatero()
    mp=im.load()
    imaux=Image.new("RGBA",(im.size[0],im.size[1]),"white")
    mpaux=imaux.load()
    for i in range(im.size[0]):
            for j in range(im.size[1]):
```

```
                    if j<=((-2/3)*im.size[1]/im.size[0])*i+im.
                        size[1] and j<=((2/3)*im.size[1]/im.size
                        [0])*i+im.size[1]/3 and i<im.size[0]/2:
                            mpaux[i,j]=mp[i,j]
                    else:
                            mpaux[i,j]=(255,255,255,0)
    return(imaux)
```

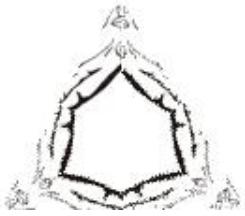
Iniciamos el método t.fundam3()

```
def fundam4(self):
        """
        divide a triangle in six same parts.
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.fundam3(2)

        """
        im1=self.fundam3()
        mpim1=im1.load()
        im2=im1.rotate(180)
        im3=Teselaciones(im2).SimVok().rotate(-60, expand=1)
        caja=(im3.size[0]-im1.size[0],0,im3.size[0],im1.size[1])
        im4=im3.crop(caja)
        mpim4=im4.load()
        imaux=Image.new("RGBA",(im1.size[0],im1.size[1])
            ,(255,255,255,0))
        mpaux=imaux.load()
        for i in range(im4.size[0]):
                for j in range(im4.size[1]):
                        if j<((-2/3)*im4.size[1]/im4.size[0])*i+
                            im4.size[1] and j>=((2/3)*im4.size
                            [1]/im4.size[0])*i+im4.size[1]/3 and
                            i<im4.size[0]/2:
                                mpaux[i,j]=mpim4[i,j]
                        else:
                                mpaux[i,j]=mpim1[i,j]
        pp2=imaux.rotate(-120, expand=1)
        pp3=imaux.rotate(120, expand=1)
        ppaux=Image.new("RGBA",(pp3.size[0],pp3.size[1])
            ,(255,255,255,0))
        ppaux.paste(imaux, (0,0))
```

```
        ppaux.paste(pp2, (1,0),pp2)
        ppaux.paste(pp3, (-floor(imaux.size[0]/4),0),pp3)
        caja=(0,0,im1.size[0],im1.size[1])
        ppaux2=ppaux.crop(caja)
        fin=Teselaciones(ppaux2).triangulo()
        return(fin)
```

Llamamos al método t.fundam4()



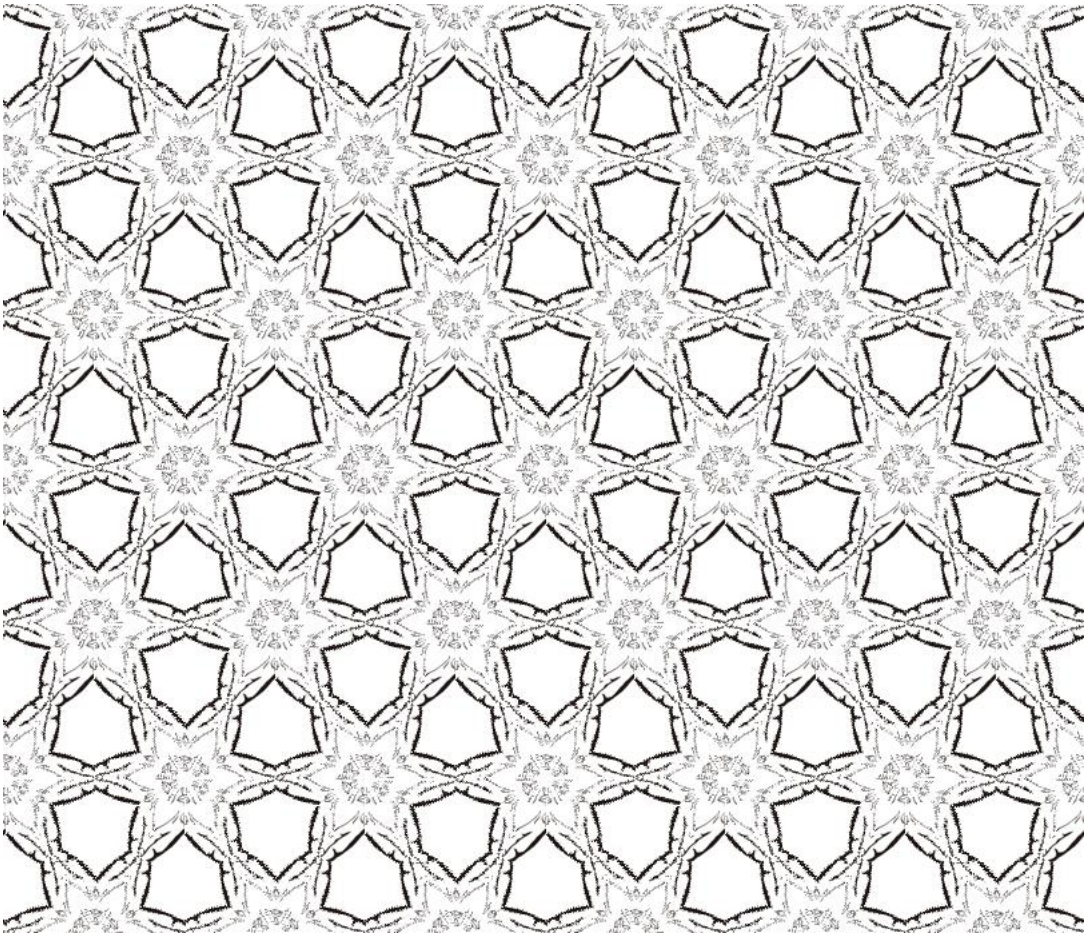
Una vez tenemos programada la región fundamental del grupo *p6m* procedemos a notar el código. El grupo se explicó en la Página 29.

```
def p6m(self,rang):
        """
        Group p6m
        Example:
        >>> imagen = Image.open("Ejemplos/salamanquesa.jpeg")
        >>> t=Teselaciones(imagen)
        >>> t.p6m(2)

        """
        imag=self.mini()
        mp=imag.load()
        imagaux=self.fundam4()
        mpaux=imagaux.load()
        im=Image.new("RGBA",(imagaux.size[0]*3,imagaux.size
            [1]*3),(255,255,255,0))
        fin=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
            size[1]*3*rang),(255,255,255,0))
        fin2=Image.new("RGBA",(imagaux.size[0]*3*rang,imagaux.
            size[1]*3*rang),(255,255,255,0))
        mpf=fin.load()
        mpim=im.load()
        imsim=Image.new("RGBA",(imagaux.size[0],imagaux.size
            [1]),(255,255,255,0))
        mpsim=imsim.load()
        s=Image.new("RGBA",(imagaux.size[0],imagaux.size[1])
            ,(255,255,255,0))
        mps=s.load()
        for i in range(imagaux.size[0]):
                for j in range(imagaux.size[1]):
                        mpsim[i,j]=mpaux[i,imagaux.size[1]-j-1]
```

```
    rc=imsim.rotate(-120, expand=1)
    caja=(rc.size[0]-imagaux.size[0],rc.size[1]-imagaux.
        size[1],rc.size[0],rc.size[1])
    rc2=rc.crop(caja)
    mprc2=rc2.load()
    for i in range(imagaux.size[0]):
            for j in range(imagaux.size[1]):
                    mps[i,j]=mprc2[imagaux.size[0]-i-1,j]
im.paste(imagaux, (0,0))
    im.paste(s, (floor(imagaux.size[0]/2)+2,0),s)
    im.paste(s, (2,imagaux.size[1]-1),s)
    im.paste(imagaux, (floor(imagaux.size[0]/2)+1,imagaux.
        size[1]),imagaux)
    im.paste(imagaux, (imagaux.size[0]+1,0),imagaux)
    im.paste(s,(imagaux.size[0]+2,imagaux.size[1]-1),s)
    for i in range(-2,rang*2):
            for j in range(-2,rang*2):
                    fin.paste(im, (i*3*imagaux.size[0],j
                        *2*(imagaux.size[1])),im)
for i in range(-2,rang*2):
            for j in range(-2,rang*2):
                    fin2.paste(im, (i*3*imagaux.size[0]-
                        imagaux.size[0]-floor((1/2)*imagaux.
                        size[0]),j*2*imagaux.size[1]-imagaux
                        .size[1]),im)
fin2.paste(fin, fin)
return(fin2)
```

Ejecutamos así el último método t.p6m(3)



Bibliografía

[1] J.M. Aarts, *Plane and Solid Geometry*, Springer Science, 2008.

[2] Andrew Baker, *Groups and Symmetry*, 2005.

[3] Y. Bossard, *Rosaces, frises et pavages. vol 1: étude pratique*. CEDIC, Paris 1997.

[4] T.K. Carne, *Geometry and Groups*. University of Cambridge, 2012.

[5] B.Grunbaum, *Tilings and Patterns*, shephard. New York, W.H. Freeman and Company,1987.

[6] Jarkko Kari, *Tilings and Patterns*, University of Turku, 2012.

[7] Rick Lakin, *Te Seven Frieze Groups*, 2006.

[8] Yanxi Liu, Robert T. Collins, *Frieze and Wallpaper Symmetry Groups Classification under Affine and Perspective Distortion*, Carnegie Mellon University, 1998.

[9] Patrick J. Morandi, *Symmetry Groups:The Classifcation of Wallpaper Patterns Mathematics 482/526*, New Mexico State University, 2007.

[10] Celia M.Pascual, *Cristalografía geométrica*.

[11] Doris Schattschneider, *The Plane Symmetry Groups: Their Recognition and Notation*, American Mathematical Monthly, Volume 85, Issue 6, 1978.

[12] Thomas Q. Sibley, *The Geometric Viewpoint: A Survey in Geometries*, 1998.

[13] Maxwell Stolarski, *Frieze Groups in* $\mathbb{R}^2$.

[14] C.Ruiz et al, *Simetría Dinamica*, Síntesis, Madrid 1989.

[15] John Kitchin, *Python Computations in Science and Engineering*, 2015.

[16] A.Downey, J.Elkner, C.Meyers, *Think in Python*, Green Tea Press, 2002.

[17] John Kerl, *Concrete abstract algebra in Python*, Springer Science,2013.

[18] http://users.monash.edu.au/ normd/documents/MATH-348-lecture-24.pdf

[19] http://www.acorral.es/notacion.htm

[20] http://www.math.arq.uva.es/gycga/apuntes/NotaCristal/NotaCristal.html

[21] http://pythonhosted.org

[22] http://python-para-impacientes.blogspot.com.es

[23] http://pillow.readthedocs.org