# Informe Laboratorio 2
# Paradigma lógico
# Programación en Lenguaje Prolog

Nombre: Iván Alejandro Guajardo Arias.
Curso: Paradigmas de programación.
Sección: A-1.
Profesor: Gonzalo Martinez.

03 de Noviembre, 2022.

# Tabla de contenidos:

# 2. Introducción

En el informe que se presenta a continuación, se expone el primer laboratorio del ramo "Paradigmas de programación"; el cual, se basa en el paradigma lógico, por medio del lenguaje semi-interpretado, prolog. El proyecto consiste en desarrollar un programa que facilite la manipulación de imágenes con un enfoque simplificado, utilizando el, ya mencionado, lenguaje prolog, usando el sitio swish.swi-prolog.org para trabajarlo durante su desarrollo.
Para lograr el objetivo general del laboratorio, se necesita entender el enunciado, esta vez, desde la perspectiva lógica (a diferencia de la anterior ocasión) y, desde el mismo paradigma, observar los requerimientos para llevarlos a reglas, predicados y hechos. Dentro de los pasos consecuentes, se pueden encontrar los siguientes:

1. Entender el propósito del proyecto y abstraer el problema de manera lógica, pues prolog se encargará de interpretar el TDA propuesto en este caso.
2. Proponer un sistema de reglas y predicados que permitan utilizar los hechos de manera eficiente, tal que se logre cumplir con los requerimientos funcionales de este laboratorio por medio de la interpretación de prolog.
3. Utilizar correctamente las recursiones vistas en clases, con el fin de poder implementar correctamente las reglas ya mencionadas sin desviarnos de nuestro paradigma propuesto y que se logre diferenciar correctamente del ya visto, el funcional.

En lo que viene de informe, se verá la descripción del paradigma, se hará un marco teórico que facilite la comprensión del problema y, facilite así, el análisis de este. De esta manera, se podrá comprender además cómo el paradigma lógico puede facilitar la resolución de problemas de manera efectiva desde una perspectiva distinta, abstrayendo todo como una base de conocimientos y consultando adecuadamente.

# 3. Descripción del paradigma

Empezaremos por definir el paradigma lógico. ¿De qué se trataría este caso?

Según lo aprendido en clases, se deja claro que este paradigma consiste en una base de conocimiento basada en **hechos** (conocimiento sabido) y **reglas** (maneras de interpretar el conocimiento, por medio de relaciones lógicas). A partir de ello, es que se pueden realizar consultas y, de manera lógica, se obtendrán todas las posibles respuestas a dichas consultas.

“No existe el concepto de asignación de variables, sino el de unificación. No hay un “estado” de las variables que se vaya modificando por sucesivas asignaciones, generalmente asociadas a posiciones de memoria, sino que las variables asumen valores al unificarse o “ligarse” con valores particulares temporalmente y se van sustituyendo durante la ejecución del programa.  “ (Lucas Spigariol, 2007, “Paradigma Lógico (material complementario)”, Universidad Tecnológica Nacional de Buenos Aires).

Así pues, todo en este paradigma dependerá de dichas relaciones lógicas que se ligan temporalmente, proponiendo todas las combinaciones posibles para que la relación se cumpla de manera correcta, siendo sus ejes principales.

## 3.1 Marco teórico

Lo más básico para poder resolver el problema, sería empezar por el proceso de abstracción. Dicho proceso se tiende a considerar como el traducir un problema a la generalidad, llevándolo a distintos niveles y subprocesos que faciliten el desarrollo de este. Sin embargo, a diferencia del laboratorio pasado, en este no considera realmente un algoritmo que permita responder a los requerimientos solicitados, sino que se plantean relaciones lógicas que permitan al computador resolver por sí mismo las consultas solicitadas.

Ahora, sería prudente si también se menciona lo que serían los subprocesos. En el caso del paradigma lógico, corresponde plantear distintos hechos, predicados y reglas que permitan la resolución de los problemas. Así pues, se toman en cuenta metas secundarias que permitan el planteamiento de las metas principales, siendo estas últimas los requerimientos solicitados.

Por otro lado, también es importante describir el trabajo en que se basa el lenguaje Prolog. Existe un concepto de “solver” que plantea nociones similares, en donde se consulta al programa, frecuentemente con el fin de optimizar procesos y otros. Desde el libro “Seven languages in seven weeks” de Bruce Tate, se menciona que Prolog es como un buen repostero, pues le describes lo que te gusta y luego él elige los ingredientes y así hornear el pastel para ti. Así pues, este paradigma nos deja una noción bastante clara, y es que acá nos enfocamos en qué queremos, no en el cómo.

# 4. Problemática

## 4.1 Descripción del problema

El problema que el enunciado propone, consiste en el desarrollo de un software que permita la manipulación de imágenes, así como photoshop o GIMP. Sin embargo, este tiene un enfoque simplificado de lo que serían dichos programas, pero permitiría igualmente el uso de funciones tales como la rotación de imagen, voltearla, transformarla, editarla, etc. Otra diferencia destacable entre los softwares mencionados y el propuesto por el laboratorio, es que este último tendría un enfoque principalmente en imágenes RGB/RGB-D y representaciones sencillas de las imágenes, como listas de píxeles u otras.

## 4.2 Análisis del problema

A grandes rasgos, el problema consiste en presentar un software que permita la edición de imágenes, como los ya señalados. Entonces, lo primero sería considerar que las imágenes consisten en arreglos bidimensionales de píxeles, por lo que sería importante disponer de funciones que permitan la manipulación de los píxeles, sus valores, sus posiciones, etc. Considerando la perspectiva lógica que se le da al laboratorio, es que se propone el uso de listas nuevamente para poder contener la información de cada pixel y, posteriormente, cada imagen. Por otro lado, también se consideran los requerimientos solicitados para el laboratorio, los cuales, se pueden observar en la tabla de autoevaluación, anexada al final del informe (ver tabla 3).

A partir de dicha tabla, podemos considerar que los constructores irán principalmente de reglas que creen listas con la información propuesta; la pertenencia irá por un lado de tomar dichos constructores y corroborar la información mediante la unificación; y, el resto de requerimientos, consistirán en recibir información que será interpretada bajo ciertas relaciones que se adecúen mejor en cada caso.

## 4.3 Otras consideraciones

Se pueden ingresar **N** píxeles a cada imagen, considerando su ancho Ancho y su alto Largo; esto implica entonces la posibilidad de una imagen infinitamente grande, o una diminuta. A partir de ello, es que se podrían considerar los valores Ancho y Largo para poder implementar funciones de volteo, o la posibilidad de un TDA para cada píxel, de modo que se pueda abordar cada caso particular (bit, rgb y hex) de manera adecuada en su respectiva imagen. Siguiendo con la diferenciación, es que podemos distinguir 2 casos principales para cada regla: bit/hex o rgb, puesto que sólo existen 2 largos distintos para los pixeles y se pueden usar relaciones similares entre los bits y los hex, en la mayoría de los casos. Así pues, se distinguen algunos predicados que permiten trabajar casos similares para las distintas reglas que se proponen posteriormente.

De esta manera, se lograría finalmente crear imágenes en base a listas de píxeles, modificarlas en base a modificar los valores de los píxeles, convertirlas, preguntar de qué tipo son, etc.

# 5. Diseño de solución

## 5.1 Descripción de la solución

La solución propuesta, plantea que en un inicio se trabaje desde lo más básico hasta lo más complejo, empezando por píxeles de cada tipo, sus funciones generales y, desde ahí, retomar con lo que corresponde a imágenes como tal, que es lo que aparece en el enunciado del laboratorio. De esta manera, se definen los TDAs pixrgb, pixbit, pixhex y el de imagen.
Por su parte, se podría considerar cada TDA como una lista con sus respectivos valores, selectores, modificadores, etc. Finalmente, esto permitiría implementar funciones que faciliten la perspectiva del qué se quiere obtener, proyectando un resultado que cumpla con lo solicitado en cada aspecto de la evaluación.
En cada píxel, se considera posición x, posición y, y sus respectivos valores que le dan un significado como píxel, más allá de su ubicación en la imagen y el tipo de píxel; de esta manera, pixrgb quedaría (pos_x pos_y r g b d), por ejemplo. Posteriormente, se consideran también los predicados que caracterizan a cada tipo de dato abstracto, como su pertenencia y sus modificadores, que luego son de utilidad para la resolución de problemas propuestos por el enunciado.

No se considera como la solución más refinada, puesto que tiene sus limitaciones en cuanto a la pulcritud del código, sin embargo, cumple perfectamente con los requisitos propuestos en un inicio. Por su parte, cada función ha tenido su utilidad en su respectivo sector o, incluso en otras funciones de mayor grado.

## 5.2 TDAs

Cada píxel consiste en una lista que considera sus posiciones x e y, su(s) valor(es) y la profundidad de este. De esta forma, se consideran los ya mencionados como los siguientes ejemplos:
(ver tabla 1 en Anexos)

Por otro lado, es importante considerar que todos estos elementos comparten alguna característica y esto implica el uso de funciones similares, por lo que se consideraron predicados que facilitan el seleccionar y modificar dichos valores en común sin la necesidad de llenar de código irrelevante.

Finalmente, el constructor para el TDA image consistiría en algo como lo que se presenta a continuación:
(ver tabla 2 en Anexos)

Siendo p, una cantidad N de píxeles a entregar, Width el ancho y Height el alto, que servirían para poder implementar algunas funciones de posicionamiento. (Queda anexado un esquema de los TDA).

<u>TDAs implementados:</u>

a. image
b. pixrgb
c. pixhex
d. pixbit

# 6. Aspectos de implementación

La estructura a seguir por el proyecto, contempla el uso de metas secundarias que faciliten la implementación de los más complejos. De esta manera, se contempla un archivo main (tuve problemas para separar los TDA en distintos archivos) que contiene los 4 TDA: pixbit, pixhex, pixrgb e image. El orden que sigue, consiste en separar con comentarios cada meta principal, creando una sección que permita leer las metas secundarias utilizadas en la mayoría de casos, cercano a la principal.

Otro aspecto relevante, sería la prohibición de cualquier uso de programación bajo otro paradigma, como lo son por ejemplo las variables. Además de mencionar que la versión utilizada fue la 8.5.20 de prolog desde swish.swi-prolog.org para toda la realización de este laboratorio.

# 7. Instrucciones de uso

Lo más relevante al momento de utilizar el programa, consiste en ingresar correctamente los valores. Cada regla tiene definido su dominio, también tiene comentada una descripción que facilita su comprensión para no cometer errores. Lo importante en esta experiencia es principalmente ir al archivo main y plantear consultas como las que se ejemplifican en los ejemplos anexados.

Finalmente, basta con abrir el código desde swish.swi-prolog.org y escribir las consultas en su respectiva ventana, utilizando los scripts comentados al final del código para poder probar cada regla de manera oportuna.

# 8. Resultados

## 8.1 Ideales

Idealmente, se espera un software de uso simple que permita trabajar píxeles representados como listas. Se logra de manera exitosa el realizar las siguientes acciones en píxeles e imágenes modificar, convertir, manipular, etc. Además, se hace correcto uso del proceso de abstracción para proponer TDAs que permitan implementar cada regla que logre relacionar los hechos propuestos dentro del enunciado mediante listas y recursiones principalmente (es esencial para el paradigma lógico, a diferencia de las variables, que derechamente no se deben usar).

### 8.2 Reales

Se lograron implementar las primeras 12 funciones del enunciado, hasta rotate90. Esto demuestra un buen aprendizaje a nivel estudiante, ya que se usó provechosamente el tiempo y se dio un correcto uso a los TDA, recursiones, listas y otros aspectos vistos en clase. Satisfactoriamente, funciona cada función como debería.

## 8.3 Posibles fallos en el estudiante y análisis de lo no logrado

Principalmente, se destaca la falta de orden y claridad al momento de empezar con el proceso de abstracción, desde ahí, se derivan otras falencias como el desorden en funciones de menor relevancia y la abundancia de líneas innecesarias. Es posible que exista una mejor manera de plantear los TDA, aunque de todos modos resultó efectivo lo propuesto en el código.

De las reglas no implementadas, están: compress, edit, invertColorBit, invertColorRGB, adjustChannel, image to string, depthLayers y decompress. Lamentablemente, no se alcanzaron a realizar, sin embargo hay algunas que no son tan complicadas de resolver si se considera por ejemplo, que para invertir el color de un bit basta con cambiar de 1 a 0 y viceversa. Gracias al paradigma lógico, se puede concluir que se trabajan de manera similar a las ya vistas, mas sería necesario investigar el procedimiento detrás de cada una para poder implementarlas como reglas.

### 8.4 Autoevaluación

En aspectos generales, se considera un buen desempeño en cuanto al estudiante. A pesar de ello, vale la pena mencionar que existe un desorden en sus TDAs y funciones, por lo que podría considerarse un éxito. Ver tabla 3, adjunta en Anexos.

# 9. Conclusión

Dentro de todo, se considera como un éxito a la segunda experiencia realizada en el laboratorio de paradigmas, puesto que se cumplió con el mínimo de requerimientos solicitados satisfactoriamente. Logra responder a cada consulta esperada como corresponde. La mayor limitación en este caso, fue nuevamente, la falta de claridad que tuvo el estudiante desde un inicio pues, en esta ocasión, existían algunos conceptos que le costó asimilar en el nuevo paradigma, principalmente. Después de algunos días de demora, el estudiante logra establecer un avance significativo y esto le permite concluir lo suficiente; de todos modos, es importante mejorar las buenas prácticas que el estudiante posee para las siguientes entregas, de modo que se optimice el trabajo y se puedan cumplir mejor los plazos, a la vez que se obtenga mayor legibilidad para el código.

Sobre el paradigma, el estudiante concluye que es una propuesta bastante interesante que permite abstraer procesos de otra manera distinta a las ya vistas. Recuerda un poco a la noción de solver y entrega una alternativa altamente eficiente para poder responder problemas de planteamiento en muy poco tiempo de desarrollo, como lo sería por ejemplo un solucionador de sudokus, grafos u otros. Lleva a pensar además, el cómo podría influir dicho paradigma en el área de la inteligencia artificial y cómo se podría llegar a relacionar con la gente, aunque también pueda tener su relevancia en criptominado y otros. Sobre las restricciones, se puede mencionar que el paradigma en muchos casos podría presentar fallas en cuanto al uso de memoria o si acaso existe un algoritmo ideal, puesto que lo lógico nos entrega el qué sin especificar cómo, llevando posiblemente a ideas concretas sin una consideración real de lo que sería el tiempo, la memoria u otros recursos, sin embargo, facilita principalmente el trabajo al programador en sí.

# 10. Anexos

***Tabla 1***

| TDA | Plantilla | Ejemplo |
|---|---|---|
| pixbit-d | ( pos_x pos_y bit depth) | ( 0 1 (0\|1) 4) |
| pixrgb-d | ( pos_x pos_y r g b depth) | (0 0 255 16 176 74) |
| pixhex-d | ( pos_x pos_y r g b depth) | (0 1 “#FF00AC” 89) |

***Tabla 2***

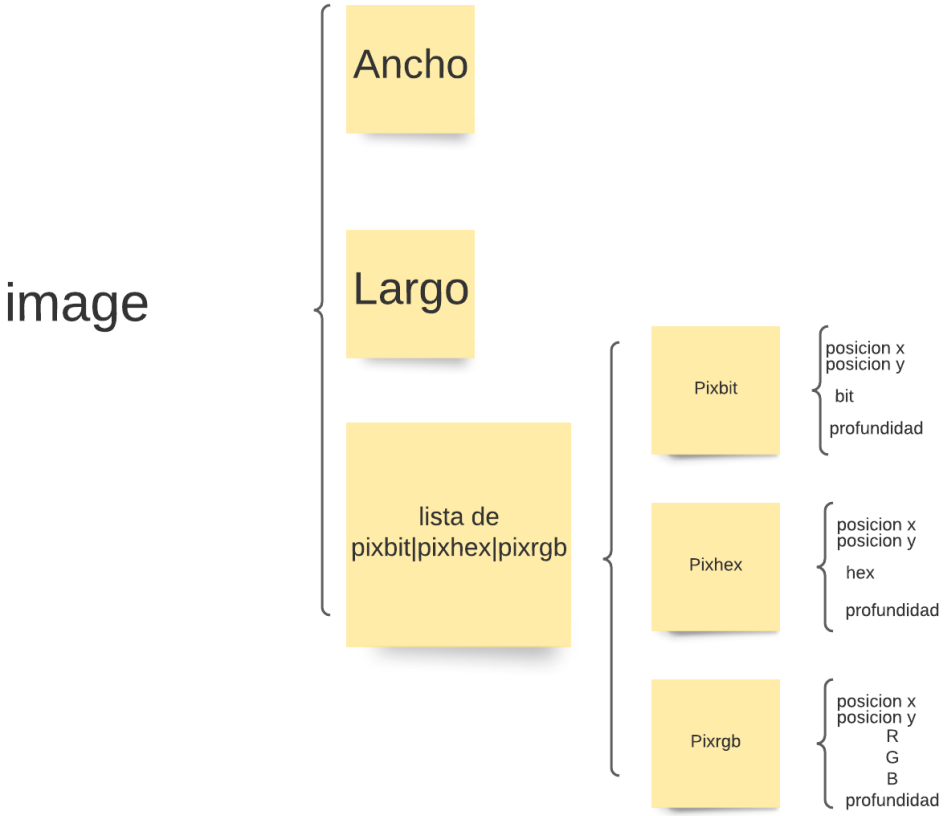
| | | |
|---|---|---|
| image | image(Ancho, Largo, Pixlist, [Ancho, Largo, Pixlist]). | (2 2 p1 p2 p3 p4) |

***Tabla 3***

| TDA | Valor binario (logrado \| no logrado) |
|---|---|
| **TDA image - constructor** | **logrado 1** |
| **TDA image - bitmap?** | **logrado 1** |
| **TDA image - pixmap?** | **logrado 1** |
| **TDA image - hexmap?** | **logrado 1** |
| **TDA image - compressed?** | **logrado 1** |
| **TDA image - flipH** | **logrado 1** |
| **TDA image - flipV** | **logrado 1** |
| **TDA image - crop** | **logrado 1** |
| **TDA image - imgRGB->imgHex** | **logrado 1** |
| **TDA image - histogram** | **logrado 1** |
| **TDA image - rotate90** | **logrado 1** |
| **TDA image - compress** | **no logrado 0** |
| **TDA image - edit** | **no logrado 0** |

| | |
|---|---|
| **TDA image - invertColorBit** | **no logrado 0** |
| **TDA image - invertColorRGB** | **no logrado 0** |
| **TDA image - adjustChannel** | **no logrado 0** |
| **TDA image - image->string** | **no logrado 0** |
| **TDA image - depthLayers** | **no logrado 0** |
| **TDA image - decompress** | **no logrado 0** |

***Esquema 1***




**%----------**

**% DESDE AQUÍ EMPIEZAN MIS SCRIPTS DE EJEMPLOS**

**/* Algunos predicados**

**pixbit( 0, 0, 1, 10, PA),**
**pixbit( 0, 1, 0, 20, PB),**
**pixbit( 1, 0, 0, 30, PC),**
**pixbit( 1, 1, 1, 4, PD),**
**image( 2, 2, [PA, PB, PC, PD], I),**

**imageTobitmap(I),**
**imageIsCompressed(I).**
**no comprimida, BORRAR 1 pixel de la lista para que dé true**

**/* flipH y flipV en los 3 pixeles**
**pixbit( 0, 0, 1, 10, PBA),**
**pixbit( 0, 1, 0, 20, PBB),**
**pixbit( 1, 0, 0, 30, PBC),**
**pixbit( 1, 1, 1, 4, PBD),**
**image( 2, 2, [PBA, PBB, PBC, PBD], IBit1),**
**flipH(IBit1, IBit1FH),**
**flipV(IBit1, IBit1FV),**
**pixhex( 0, 0, "#E3A589", 10, PHA),**
**pixhex( 0, 1, "#E3ABD9", 20, PHB),**
**pixhex( 1, 0, "#FFA234", 30, PHC),**
**pixhex( 1, 1, "#AB2345", 4, PHD),**
**image(2,2,[PHA, PHB, PHC, PHD], IH1),**
**flipH(IH1, IHFH),**
**flipV(IH1, IHFV),**
**pixrgb( 0, 0, 244, 255, 2, 10, PRA),**
**pixrgb( 0, 1, 244, 255, 2, 20, PRB),**
**pixrgb( 1, 0, 160, 1, 155, 30, PRC),**
**pixrgb( 1, 1, 1, 0, 2, 4, PRD),**
**image(2, 2, [PRA, PRB, PRC, PRD], IR1),**
**flipH(IR1, IR1FH),**
**flipV(IR1, IR1FV).**

**/*para 3x3**
**pixbit( 0, 0, 1, 10, PBA),**
**pixbit( 0, 1, 0, 20, PBB),**
**pixbit( 0, 2, 0, 30, PBC),**
**pixbit( 1, 0, 1, 4, PBD),**
**pixbit( 1, 1, 1, 4, PBE),**
**pixbit( 1, 2, 1, 4, PBF),**
**pixbit( 2, 0, 1, 4, PBG),**
**pixbit( 2, 1, 1, 4, PBH),**
**pixbit( 2, 2, 1, 4, PBI),**

```
image( 3, 3, [PBA, PBB, PBC, PBD, PBE, PBF, PBG, PBH, PBI], IBit1),
flipH(IBit1, IBit1FH),
flipV(IBit1, IBit1FV).


/* Ejemplo crop
pixbit( 0, 0, 1, 10, PBA),
pixbit( 0, 1, 0, 20, PBB),
pixbit( 0, 2, 0, 30, PBC),
pixbit( 1, 0, 1, 4, PBD),
pixbit( 1, 1, 1, 4, PBE),
pixbit( 1, 2, 1, 4, PBF),
pixbit( 2, 0, 1, 4, PBG),
pixbit( 2, 1, 1, 4, PBH),
pixbit( 2, 2, 1, 4, PBI),
image( 3, 3, [PBA, PBB, PBC, PBD, PBE, PBF, PBG, PBH, PBI], IBit1),
crop(1,1,2,2, IBit1, Icrop).

/* Ejemplo crop rgb
pixrgb( 0, 0, 244, 255, 2, 10, PRA),
pixrgb( 0, 1, 244, 255, 2, 20, PRB),
pixrgb( 1, 0, 160, 1, 155, 30, PRC),
pixrgb( 1, 1, 1, 0, 2, 4, PRD),
image(2, 2, [PRA, PRB, PRC, PRD], IR1),
crop(0,0,0,0, IR1, Icrop).*/

/* Ejemplo de imagen rgb a hex
pixrgb( 0, 0, 244, 255, 2, 10, PRA),
pixrgb( 0, 1, 244, 255, 2, 20, PRB),
pixrgb( 1, 0, 160, 1, 155, 30, PRC),
pixrgb( 1, 1, 1, 0, 2, 4, PRD),
image(2, 2, [PRA, PRB, PRC, PRD], IR1),
imgRGBToHex(IR1, IH1).


/* Histograma Bit
pixbit( 0, 0, 1, 10, PBA),
pixbit( 0, 1, 0, 20, PBB),
pixbit( 1, 0, 0, 30, PBC),
```

```
pixbit( 1, 1, 1, 4, PBD),
image(2, 2, [PBA, PBB, PBC, PBD], IR1),
histogram(IR1, IH1).

/* Histograma RGB
pixrgb( 0, 0, 244, 255, 2, 10, PRA),
pixrgb( 0, 1, 244, 255, 2, 20, PRB),
pixrgb( 1, 0, 160, 1, 155, 30, PRC),
pixrgb( 1, 1, 1, 0, 2, 4, PRD),
image(2, 2, [PRA, PRB, PRC, PRD], IR1),
histogram(IR1, IH1).



/* Histograma Hex
pixhex( 0, 0, "#E3A589", 10, PHA),
pixhex( 0, 1, "#E3ABD9", 20, PHB),
pixhex( 1, 0, "#FFA234", 30, PHC),
pixhex( 1, 1, "#AB2345", 4, PHD),
image(2, 2, [PHA, PHB, PHC, PHD], IR1),
histogram(IR1, IH1).


/* Rotate90 con bit
pixbit( 0, 0, 1, 10, PBA),
pixbit( 0, 1, 0, 20, PBB),
pixbit( 1, 0, 0, 30, PBC),
pixbit( 1, 1, 1, 4, PBD),
image(2, 2, [PBA, PBB, PBC, PBD], IR1),
rotate90(IR1, IH1).

/* Rotate90 con rgb
pixrgb( 0, 0, 244, 255, 2, 10, PRA),
pixrgb( 0, 1, 244, 255, 2, 20, PRB),
pixrgb( 1, 0, 160, 1, 155, 30, PRC),
pixrgb( 1, 1, 1, 0, 2, 4, PRD),
image(2, 2, [PRA, PRB, PRC, PRD], IR1),
rotate90(IR1, IH1).

/* Rotate90 con hex
```

**pixhex( 0, 0, "#E3A589", 10, PHA),**
**pixhex( 0, 1, "#E3ABD9", 20, PHB),**
**pixhex( 1, 0, "#FFA234", 30, PHC),**
**pixhex( 1, 1, "#AB2345", 4, PHD),**
**image(2, 2, [PHA, PHB, PHC, PHD], IR1),**
**rotate90(IR1, IH1).**

# 11. Bibliografía


a. https://www.youtube.com/watch?v=uudsNL2G7nE&list=PLGfF3KgbxaiwLDxZaSuec2zxNZC7zmQkI
b. Seven languages in seven days , Bruce Tate
c. https://stackoverflow.com
d. https://www.cosy.sbg.ac.at/~pmeerw/Watermarking/lena.html
e. https://uvirtual.usach.cl/moodle/mod/page/view.php?id=156475
f. https://kevinldp.wordpress.com/4-paradigma-logico/
g. Profesor Gonzalo Martinez.
h. geeksforgeeks.
i. Lucas Spigariol, 2007, “Paradigma Lógico (material complementario)”, Universidad Tecnológica Nacional de Buenos Aires.