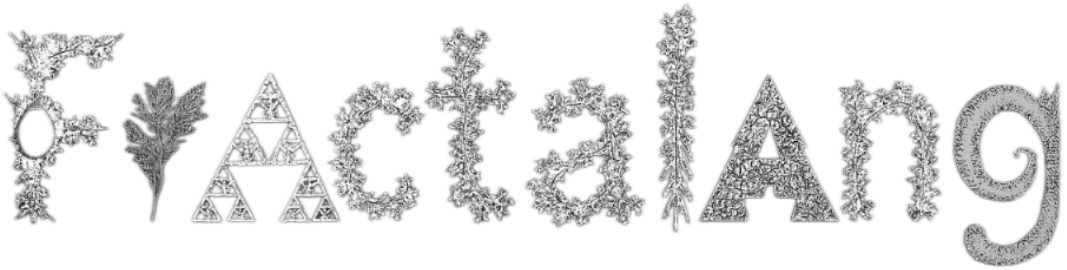



# Especificación - Fractalang

Diseño e Implementación de un Lenguaje

v1.0.3



## 1. Equipo



| Nombre | Apellido | Legajo | E-mail |
|---|---|---|---|
| Nicolás | Rivas | 64292 | nrivas@itba.edu.ar |
| Toribio | Vitón Sconza | 64275 | tvitonsconza@itba.edu.ar |
| Simon | Marabi | 61626 | smarabi@itba.edu.ar |
| Joaquín José | Huerta | 64252 | johuerta@itba.edu.ar |



## 2. Repositorio

La solución y su documentación serán versionadas en: https://github.com/nrivas-itba/TLA.

# 3. Dominio

Desarrollar un lenguaje que permita crear fractales en base a código, mediante reglas, iteraciones simples, y el dibujo de figuras.

La correcta implementación de este lenguaje permitirá crear múltiples fractales, de distintos tipos. Como por ejemplo: “Mandelbrot”, “Sierpinski”, y “Barnsley Fern”, entre otros.

A su vez, mediante la modificación de la vista, se podrá cambiar la perspectiva desde la que se ve el fractal, acercando o alejando la imagen, y moviéndola hacia los lados. A su vez, se podrán modificar las fórmulas para crear otros fractales conocidos, como Mandelbrot al cubo, entre otros nuevos nunca vistos por un humano.

El lenguaje compilará a formato .bmp, sin compresión, que permitirá generar fractales de máxima resolución, en un formato que puede ser fácilmente convertido en cualquier otro.

El lenguaje será no tipado, y case sensitive. Esto permitirá mayor flexibilidad a la hora de escribir el código. A su vez, las instrucciones terminarán en salto de línea, y los bloques de código serán definidos por el nivel de indentación.

# 4. Construcciones

## 4. Sintaxis General

Un archivo describe un fractal y sigue este orden:

```
[size <ancho> <alto>]
view [xmin,xmax] [ymin,ymax]
[color <hex1> <hex2>]

[rule <nombre_regla>:
    <instrucciones>]*

[start <nombre_regla>]*
```

**Elementos:**

- `size` → tamaño de salida en píxeles. (Opcional con default)
- `view` → ventana de coordenadas del plano a renderizar.
- `color` → dos colores hexadecimales; el motor genera un gradiente. (Opcional con default)
- `rule` → define un conjunto de instrucciones que el motor ejecuta.
- `start` → indica qué regla ejecutar como entrada principal.

## 4.2. Instrucciones Permitidas en `rule`

### 4.2.1 Escape-time fractals

*escape <var> = <expresión> until <condición> max <iteraciones>*
Itera la expresión asignando a `<var>` hasta que se cumpla la condición o se alcance el máximo.
Variables especiales:

- `z` → valor iterado.
- `c` → coordenada del punto actual.

### 4.2.2 Geometría recursiva

*draw polygon*
    *[point x1 y1]*
    *[point x2 y2]*
    *[point x3 y3]*
    *…*
Dibuja un polígono con coordenadas absolutas.

*call <nombre_regla> arg1 arg2 …*
Llama a otra regla (o a sí misma) con parámetros.

*if <condición> stop*
Detiene la ejecución de la regla actual si la condición es verdadera.

### 4.2.3 IFS (Iterated Function Systems)

*transform <prob>*
    *[Translate: x, y]*
    *[Scale: x, y]*
    *[Rotation: n°]*
    *[Shear: x°, y°]*

Aplica una transformación afín con probabilidad `<prob>` %:
De no aclararse una, se aplica a identidad. Todas las transformaciones podrán aparecer más de una vez en cualquier orden.

*points <n>*
Número de puntos a generar antes de detenerse.

De esta forma, se permitirán las siguientes funcionalidades

(I). Se podrán dibujar polígonos.
(II). Se podrán generar estructuras recursivas.
(III). Se podrán dibujar figuras píxel a píxel mediante ecuaciones recursivas.
(IV). Se podrán dibujar figuras no determinísticas en base a probabilidades.
(V). Se podrán definir reglas de dibujo.
(VI). Las reglas de dibujo podrán llamar a otras reglas de dibujo.
(VII). Las reglas de dibujado podrán ser recursivas.
(VIII). Se podrá definir el área de dibujo.
(IX). Se podrá definir el rango de color de trazado.
(X). Se podrá definir el tamaño en píxeles de la imágen de salida.

# 5. Casos de Prueba

Se proponen los siguientes casos iniciales de prueba de **`aceptación`**:

(I). Definir una vista.
(II). Definir un tamaño de imagen.
(III). Definir un rango de color.
(IV). Un programa que genera una Línea.
(V). Un programa que genera un triángulo.
(VI). Un programa que genera un cuadrado.
(VII). Un programa que genera el fractal de “Mandelbrot”.
(VIII). Un programa que genera el fractal de “Sierpinski”.
(IX). Un programa que genera el “Helecho de Barnsley”.
(X). Un programa que genera el fractal de “Julia”
(XI). Un programa que genera una vista de cerca del fractal de “Mandelbrot”.
(XII). Un programa que genera una vista de cerca del fractal de “Sierpinski”.
(XIII). Un programa que genera una vista de cerca del “Helecho de Barnsley”.

Además, los siguientes casos de prueba de **`rechazo`**:

(I). Una vista malformada.
(II). Un tamaño de imagen mal formado.
(III). Un rango de color mal formado.
(IV). Una transformación mal formada.
(V). Una rule mal formada.
(VI). Una fórmula de escape mal formada.
(VII). Un programa que inicie el dibujo de una regla no definida.
(VIII). Un programa que llame una función con una cantidad menor de argumentos a la definida por el lenguaje.
(IX). Un programa que no defina la view.
(X). Un programa que utilice símbolos no definidos.

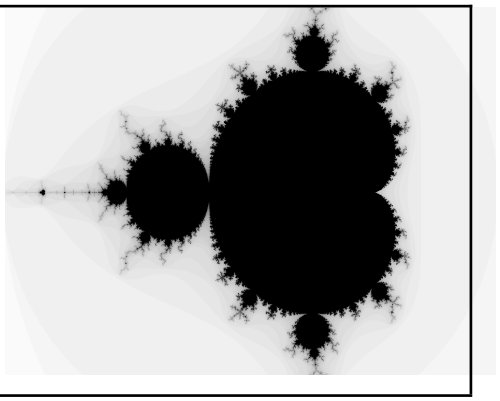
# 6. Ejemplos

Creación de un fractal de Mandelbrot.

```
// Código que genera un fractal de Mandelbrot
size 1920 1080
view [-2.5,1.0] [-1.25,1.25]
color #000000 #FFFFFF

rule mandelbrot:
    escape z = z*z + c until |z| > 2 max 1000

start mandelbrot
```



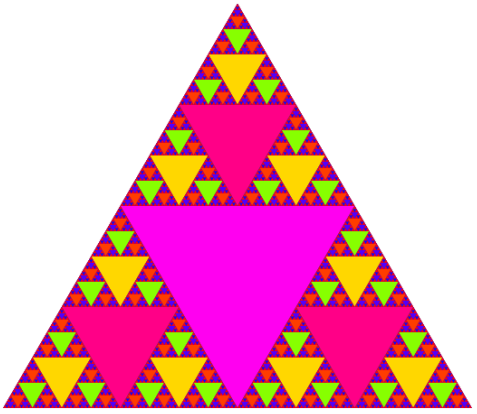
Creación de un fractal de Sierpinski

```
// Código que genera un fractal de Sierpinski
size 800 700
view [0,1] [0,0.9]
color #ffffff #000000

rule sierpinski:
    draw polygon
        point 0 0
        point 1 0
        point 0.5 0.866
    call sierpinski_part 0 0 0.5
    call sierpinski_part 0.5 0 0.5
    call sierpinski_part 0.25 0.433 0.5

rule sierpinski_part x y size:
    if size < 0.01 stop
    draw polygon
        point x y
        point x+size y
        point x+size/2 y+size*0.866
    call sierpinski_part x y size/2
    call sierpinski_part x+size/2 y size/2
    call sierpinski_part x+size/4 y+size*0.433 size/2

start sierpinski
```



Creación de un fractal de “Barnsley Fern”.

```
// Código que genera un “Helecho de Barnsley”
size 900 1400
view [-3,3] [0,10]
color #0b3d2e #8be08b

rule barnsley:
    transform 1
        scale: 0, 0.16
        translate: 0, 0

    transform 85
        scale: 0.85, 0.85
        shear: 0.04, -0.04
        translate: 0, 1.6

    transform 7
        scale: 0.2, 0.22
        shear: -0.26, 0.23
        translate: 0, 1.6

    transform 7
        scale: -0.15, 0.24
        shear: 0.28, 0.26
        translate: 0, 0.44

    points 200000

start barnsley
```

Fuentes:

- https://en.wikipedia.org/wiki/Transformation_matrix

Fuentes de imágenes:

- Mandelbrot: TP ARQUI
- Sierpinksi: https://i.sstatic.net/mb2C8.png
- Fern: https://upload.wikimedia.org/wikipedia/commons/thumb/8/83/Barnsley_fern_1024x1024.png/800px-Barnsley_fern_1024x1024.png