Instituto Tecnológico y De Estudios Superiores De Monterrey

Campus Monterrey

Compiler Design

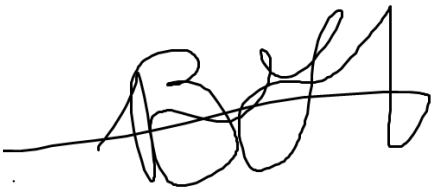
Group 02

Elda Quiroga

Hector Ceballos

Beexl Programming Language

Diana Cecilia Martinez Valverde
A00825049



José Bernardo Girón Arana
A00823490

Disclaimer:

Nos equivocamos de entregable e hicimos el tercero para el segundo :'^(
(cubo semántico)

Beexl Vision and Purpose

This compiler is a disruptive way of making pixel art. It's also to help people visualize how a vector or various geometrical forms would look after programming them in a fun and interactive way.

Main Objective and Category

Our main objective is to make pixel art with code through an imperative programming language.

Beexl Requirements

Basic elements

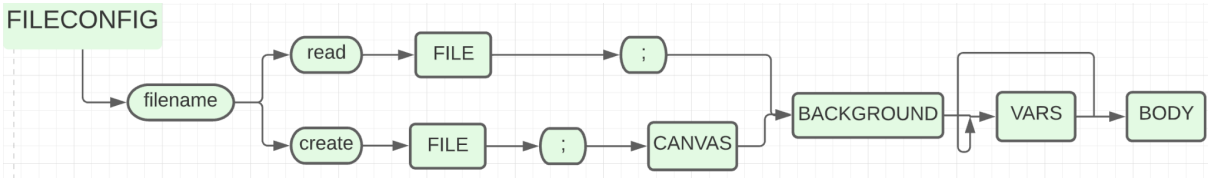
Token List

filename
read
create
;
canvas
,
rgba
{
}
vector
format
var
fun
void
main
return
#
(
)
=
[
]
:
.
x
y
red
green
blue
alpha
!

```
<
>
if
fun
from
to
do
ID
MAX_RED
MAX_BLUE
MAX_GREEN
MAX_ALPHA
```

Syntax Diagrams

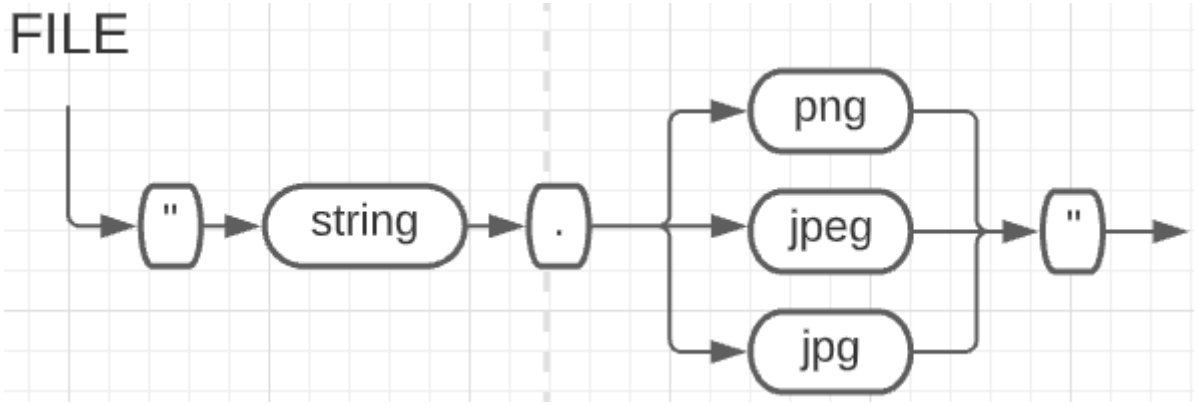
https://lucid.app/lucidchart/invitations/accept/inv_0a1f5bb4-0081-46ff-b22c-89d2b09cff47



FILENAME (se renombró a FILECONFIG)

The first thing declared, you can either create a new image or edit it, ex:

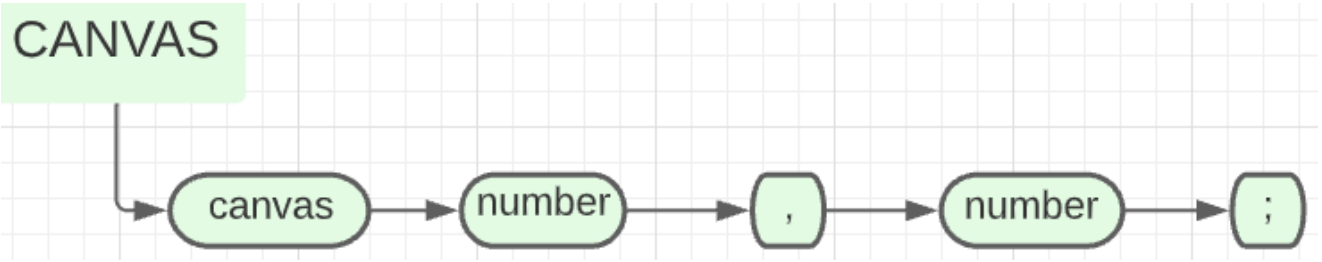
```
FILENAME CREATE "beexl.png";
CANVAS 150,1000;
FORMAT rgba;
```



FILE

At the moment of reading/creating in FILENAME we use it, example in use:

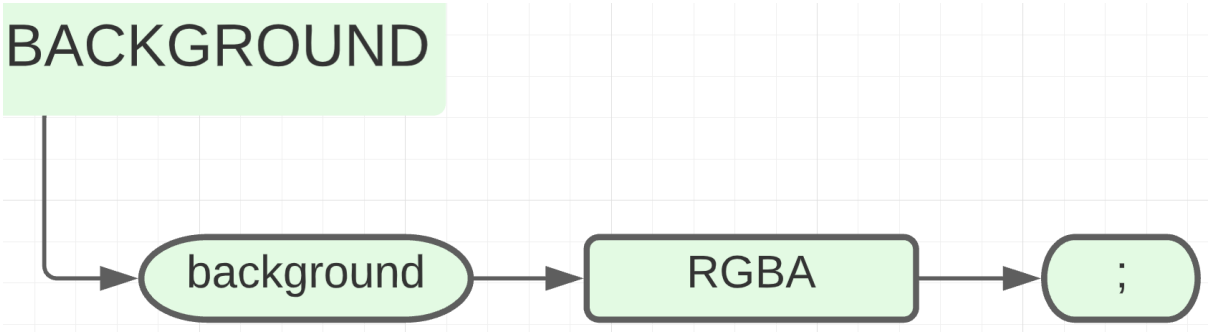
```
FILENAME CREATE "beexl.png";
```



CANVAS

To establish the size of the drawing canvas, example:

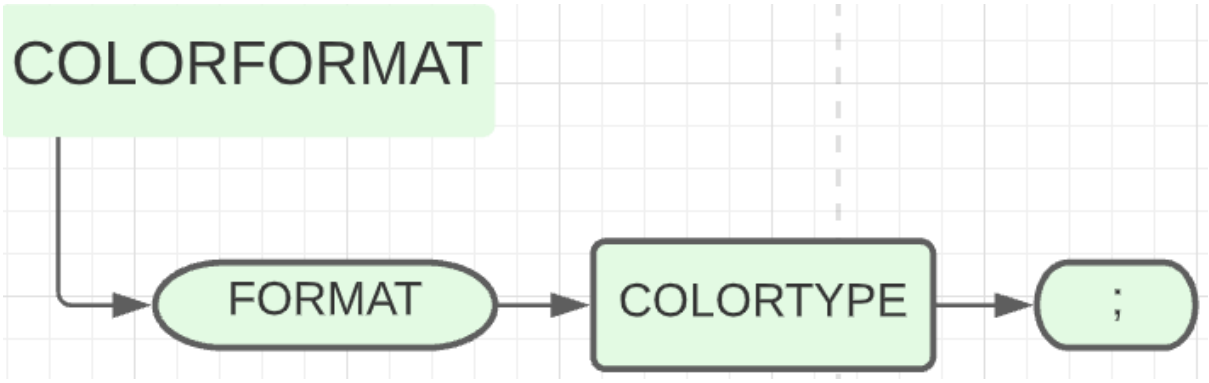
```
CANVAS 150,1000;
```



BACKGROUND

To declare the color of the background

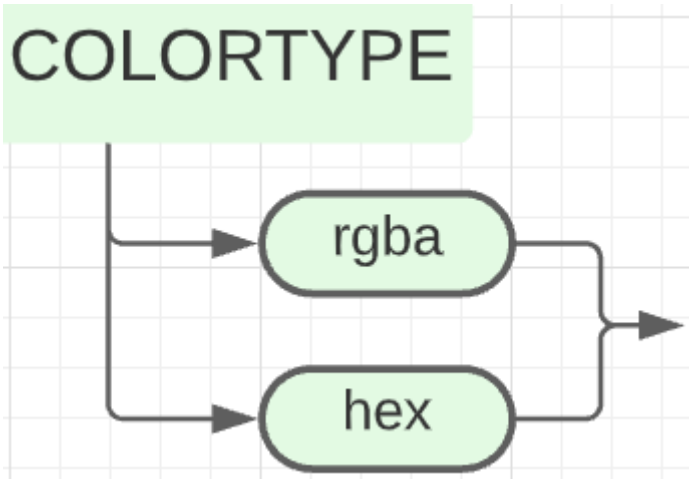
```
BACKGROUND   rgba(134,12,250,76);
```



COLORFORMAT

To declare the color format that will be used in the whole program, ex:

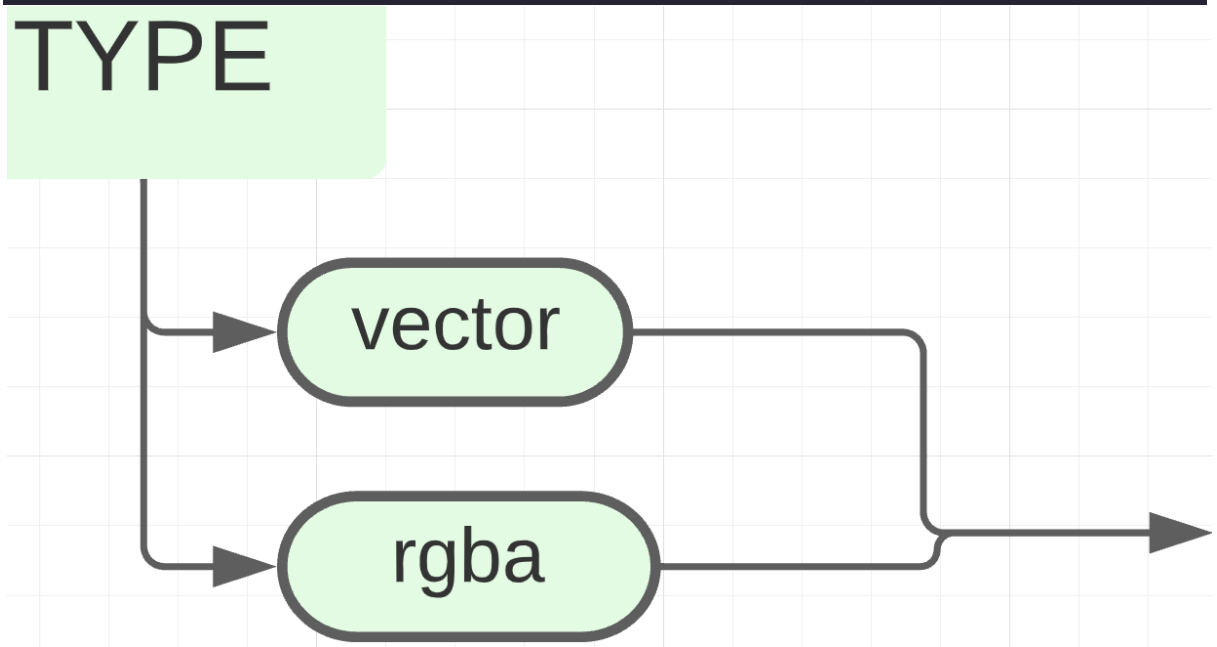
```
FORMAT rgba;
```



COLORTYPE

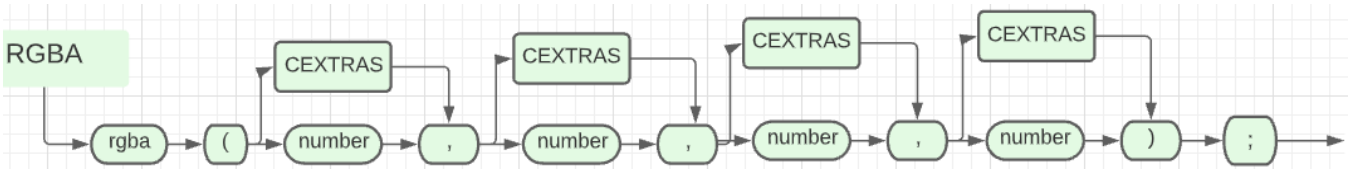
To know establish the color selection of the program, ex:

```
FORMAT rgba;
```



TYPE

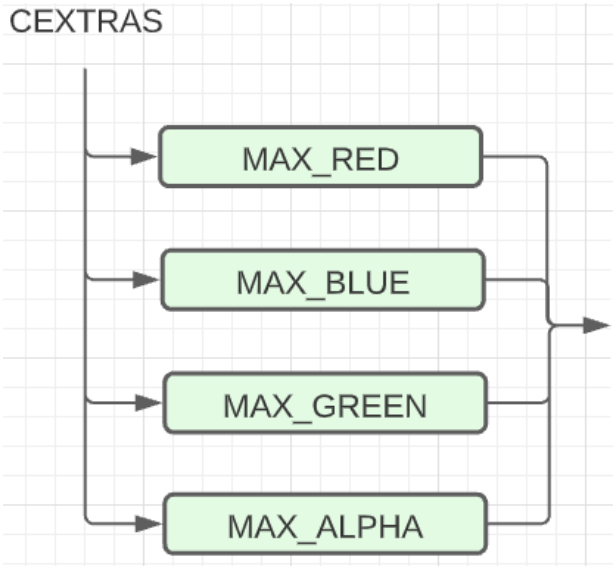
To know what type is, ex:



RGBA

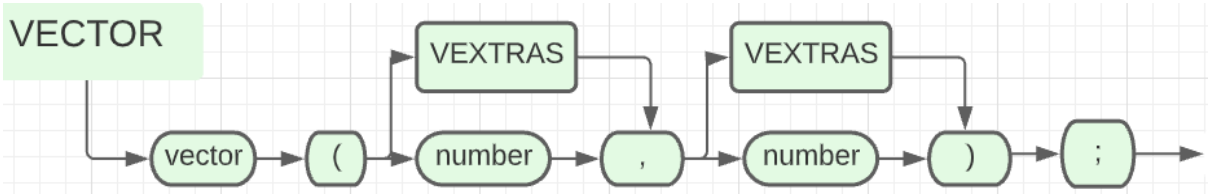
If the color format is established as RGBA, we use this to declare colors ex:

```
var color1:rgba = rgba(15,12,234,23);
```



CEXTRAS

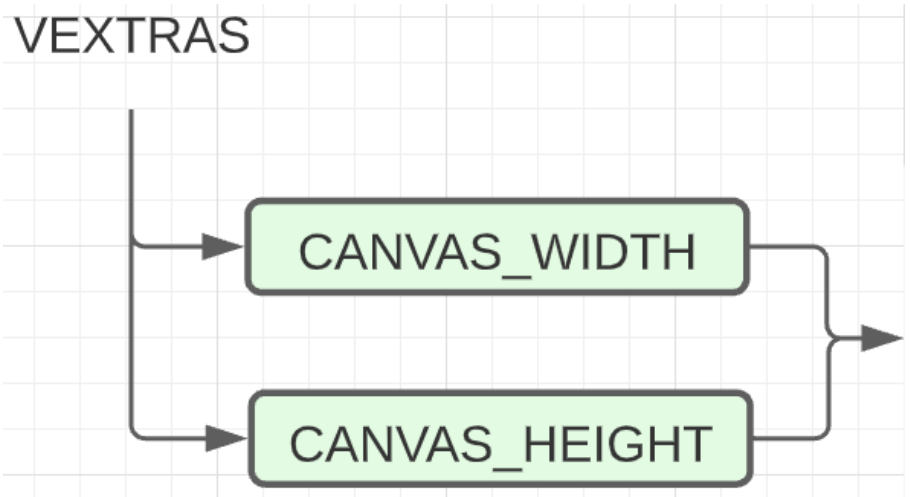
Maximum value of RGBA it cannot surpass 255



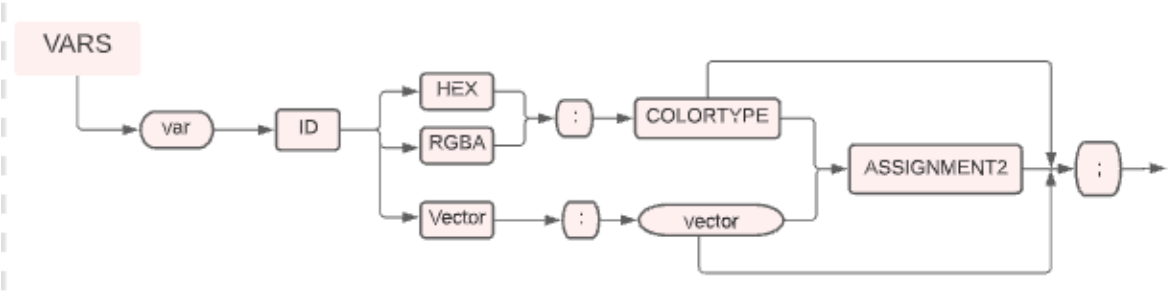
VECTOR

To establish the position of a vector, ex:

```
var vect1:vector = vector(15,32);
```

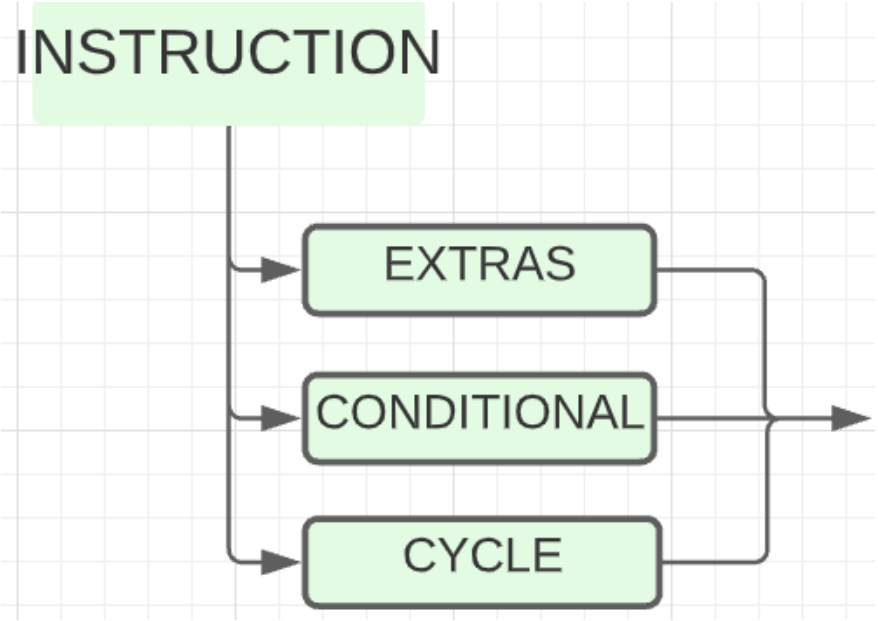


VEXTRAS



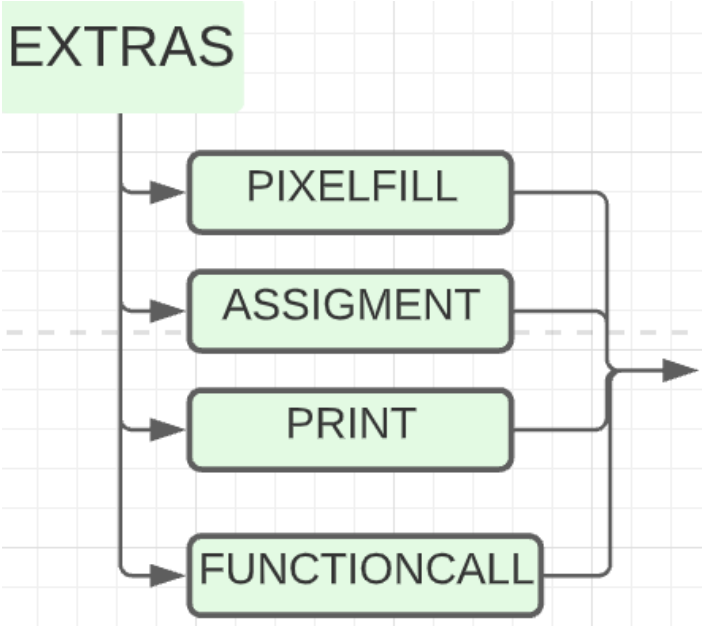
VARS

To declare variables



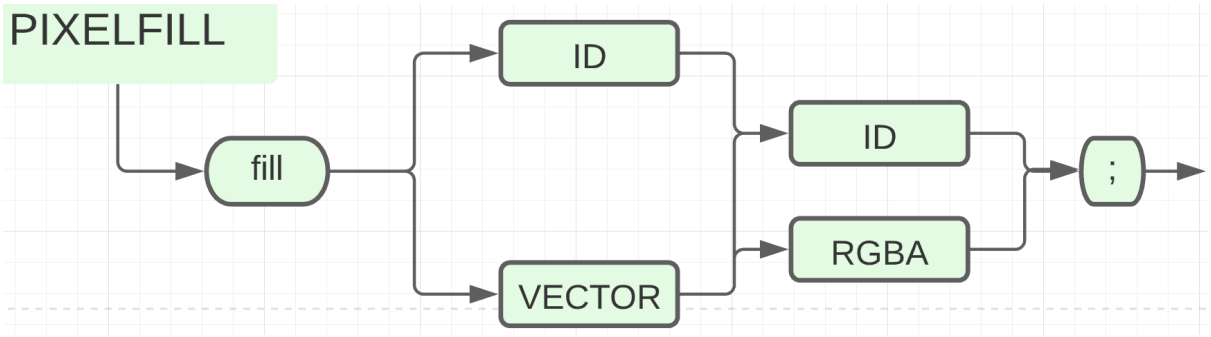
INSTRUCTION

It directs to EXTRAS/CONDITIONAL/CYCLE.



EXTRAS

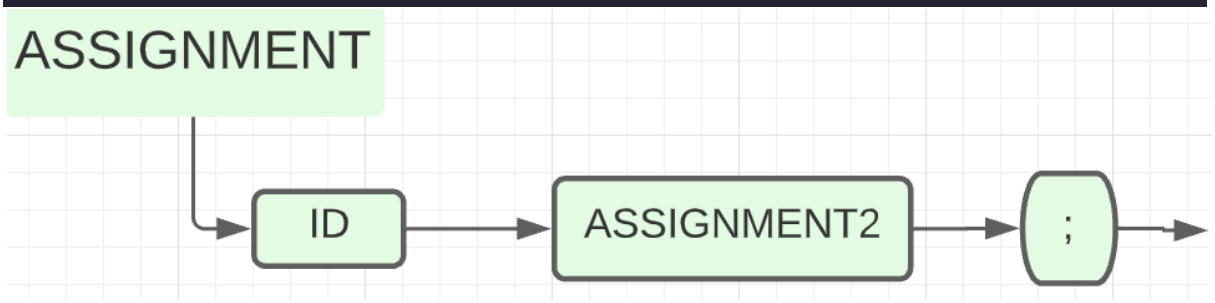
It directs to special operations
PIXELFILL/ASSIGNMENT/PRINT/FUNCTIONCALL.



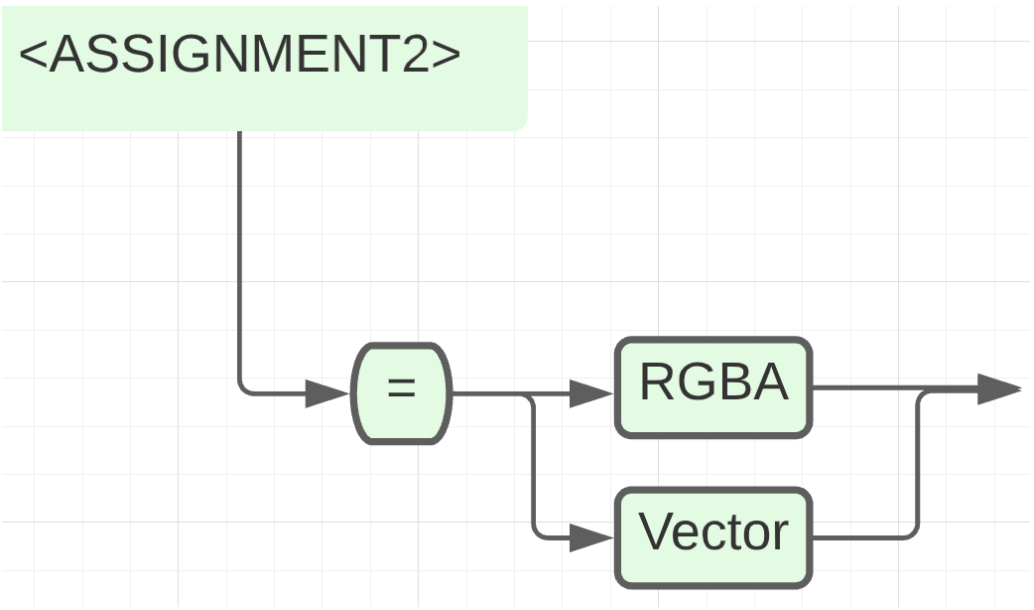
PIXELFILL

Special operation to fill pixels, ex:

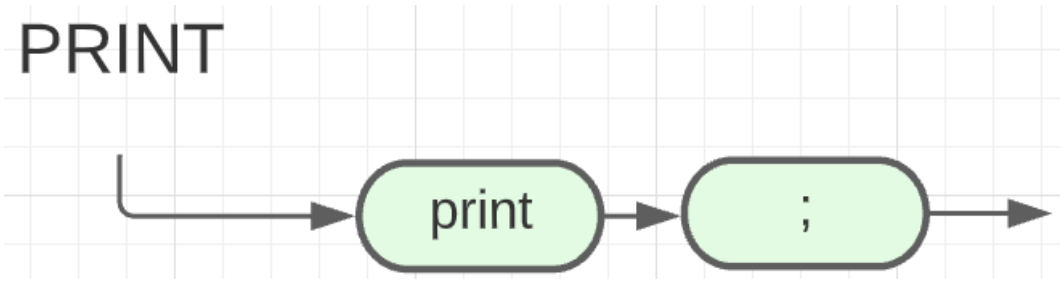
```
from startPos to endPos do
        {
            fill startPos myColor;
        }
```



ASSIGNMENT

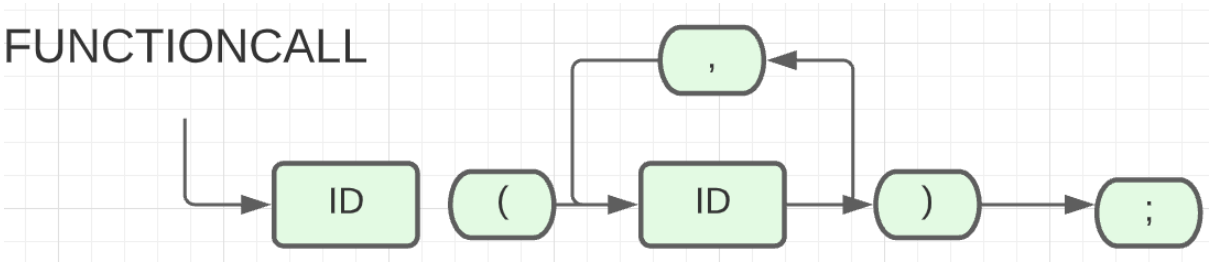


ASSIGNMENT2



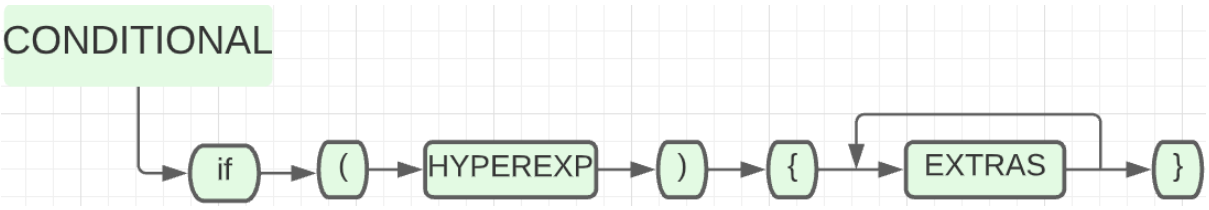
PRINT

It prints!



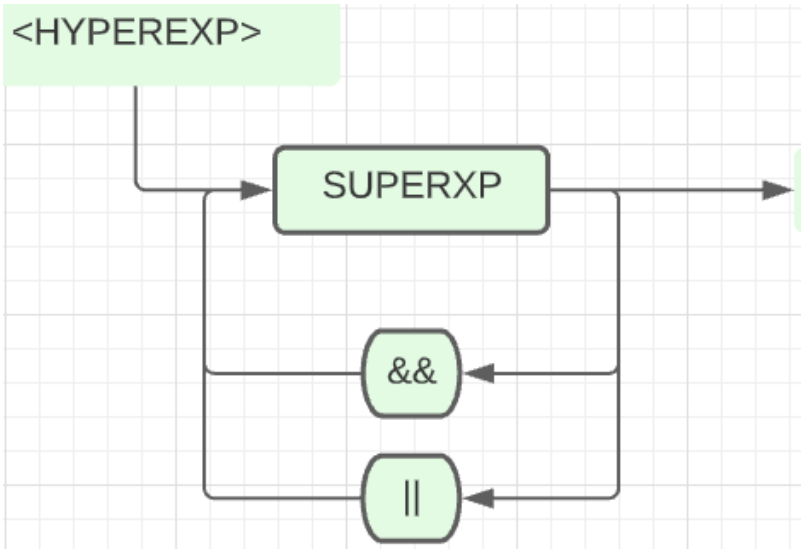
FUNCTIONCALL

To call a function



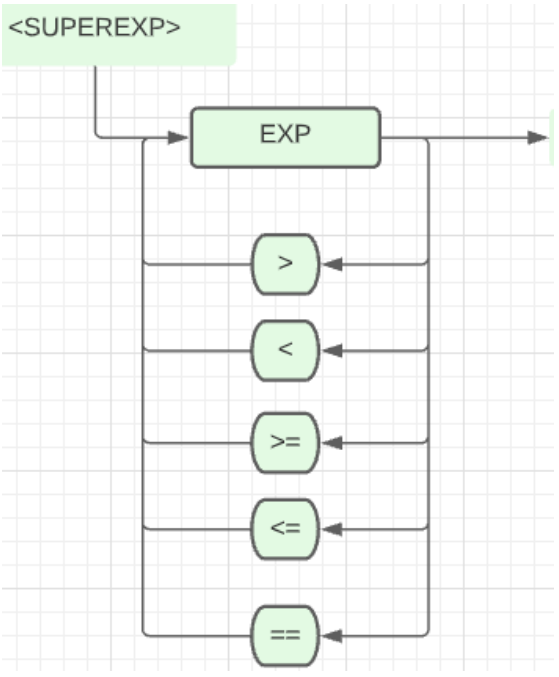
CONDITIONAL

Condition if



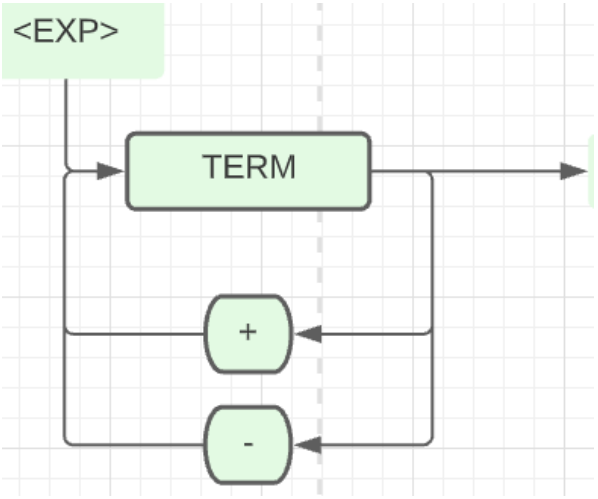
HYPEREXP

Boolean expressions (AND/OR)



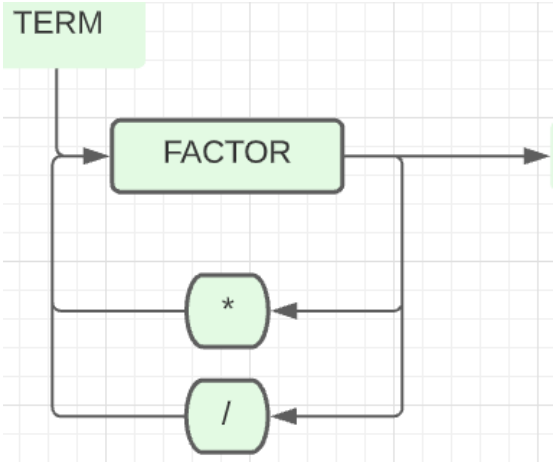
SUPEREXP

Relational expressions



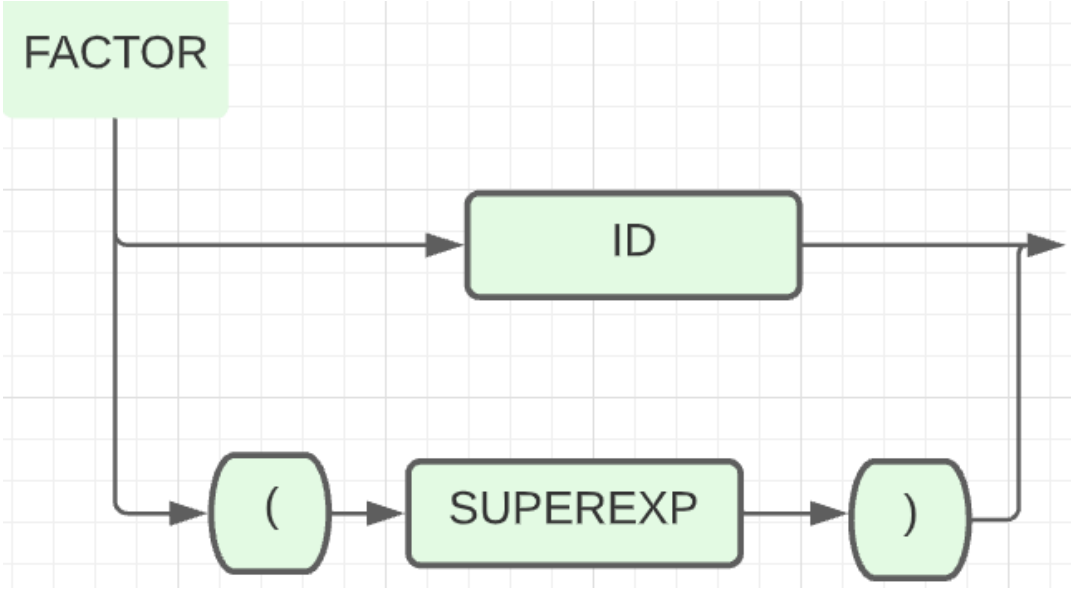
EXP

Math term for plus and minus.



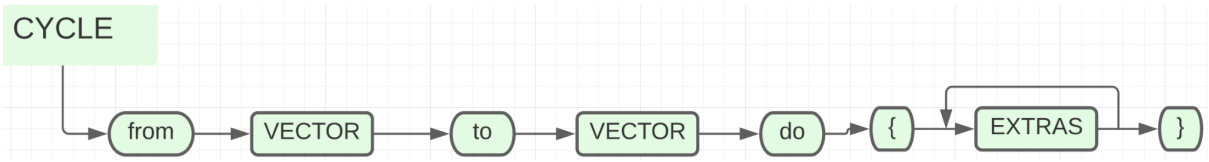
TERM

Math term for division and multiplication.



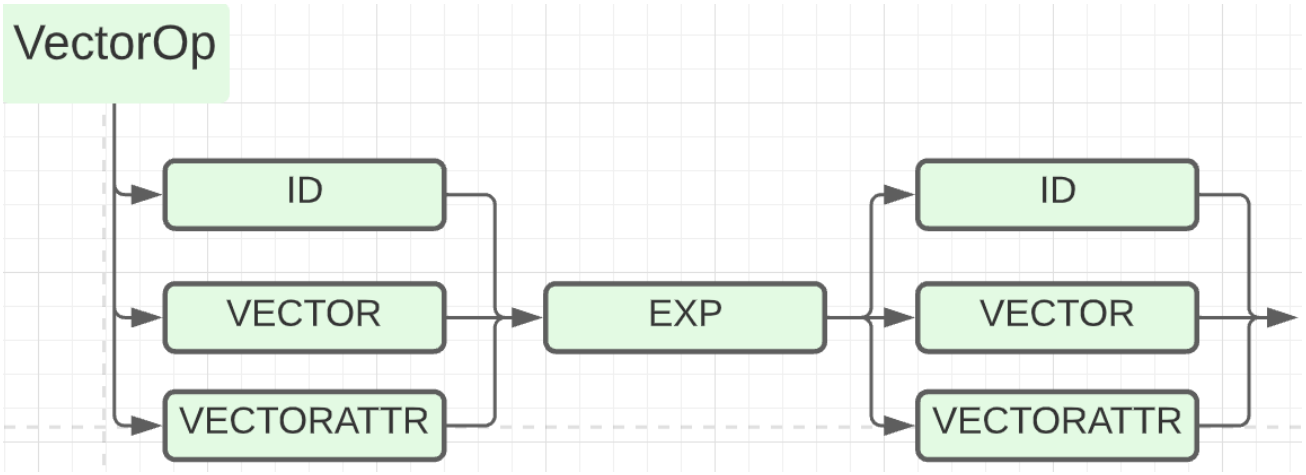
FACTOR

To prioritise a term/expression

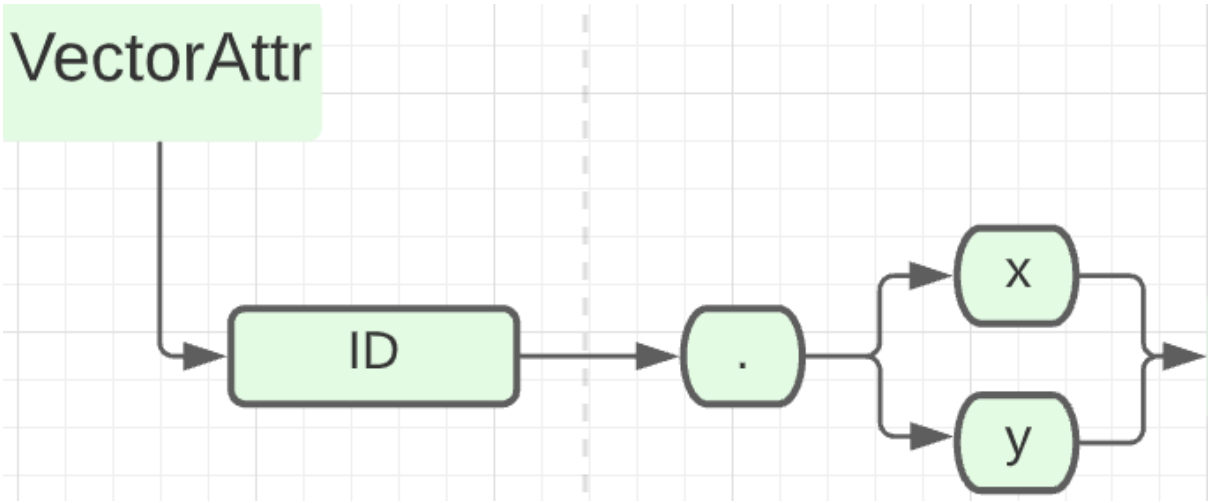


CYCLE

AS

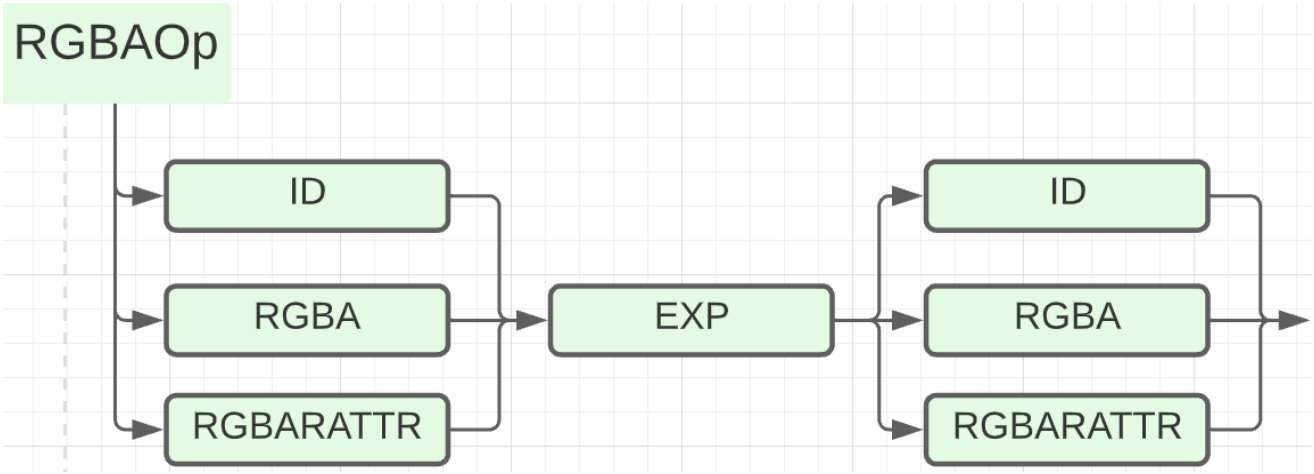


VectorOp

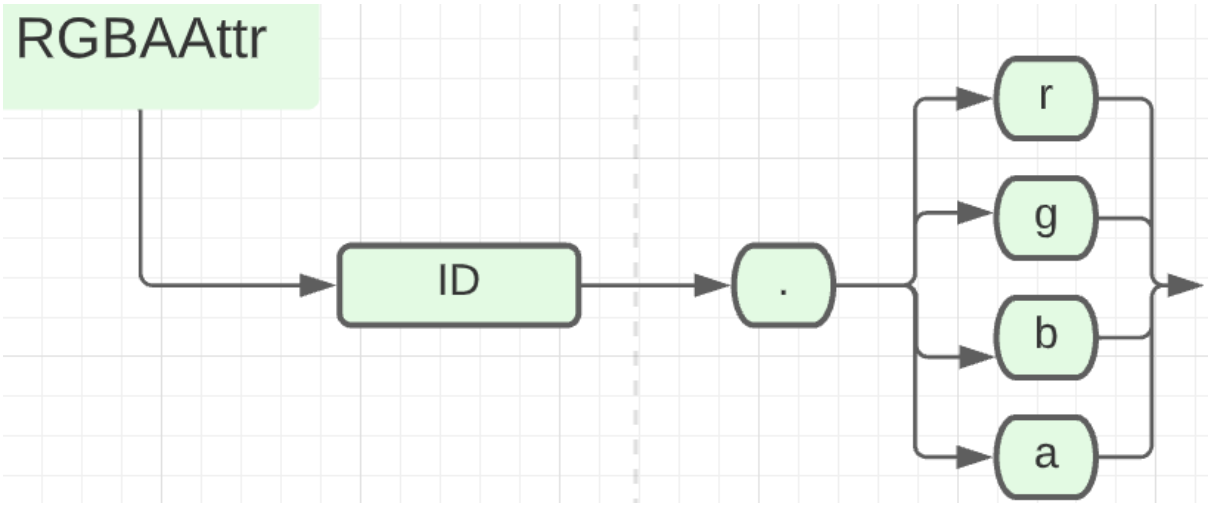


VectorAttr

To access a specific element in an Vector variable.

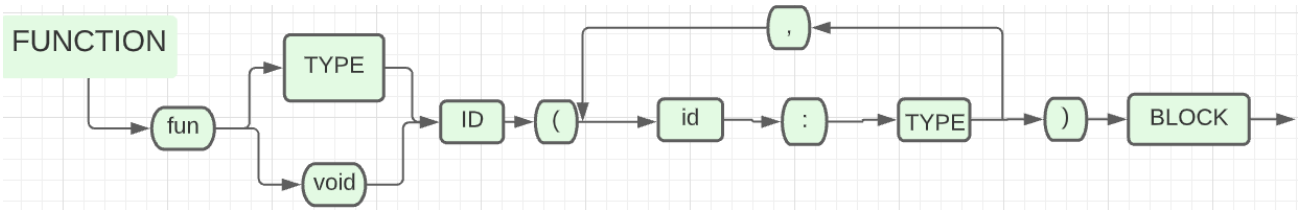


RGBAOp



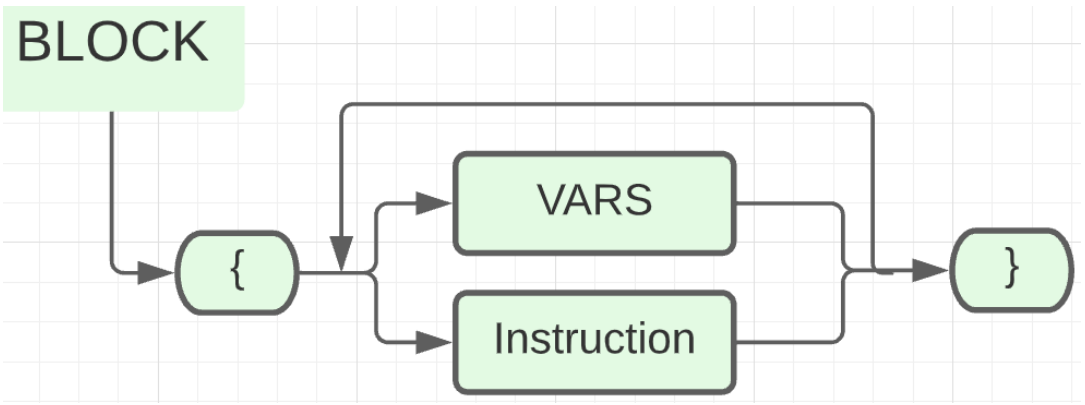
RGBAAttr

To access a specific element in an RGBA variable.

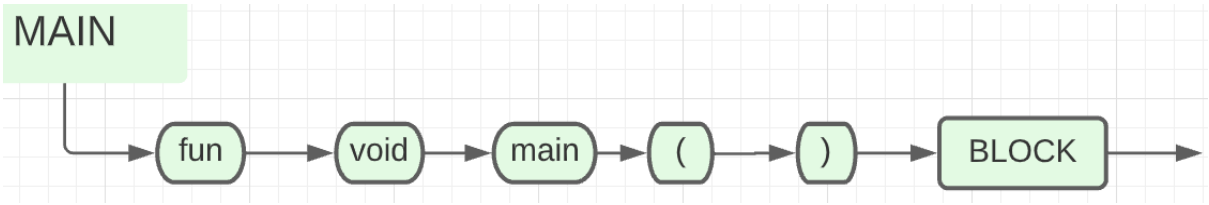


FUNCTION

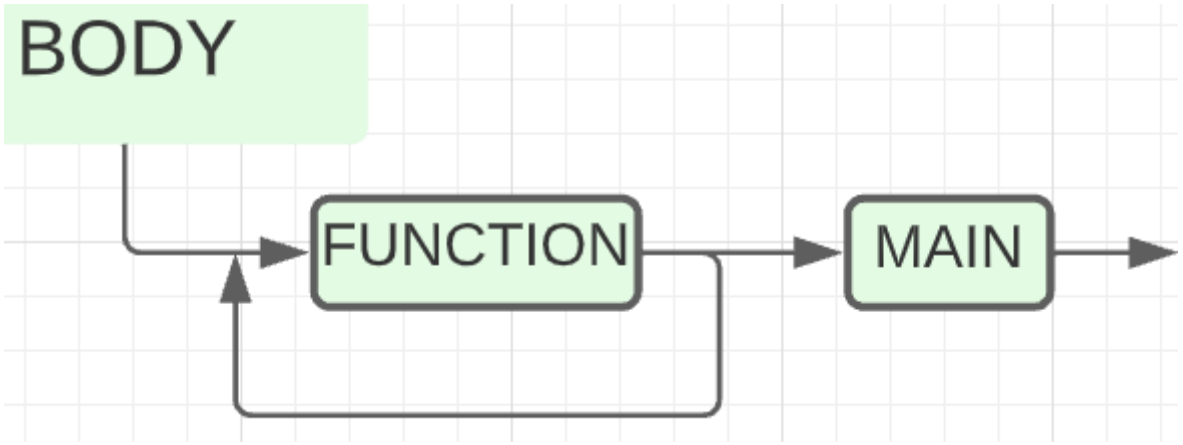
To create a function



BLOCK



MAIN



BODY

Examples

```
FILENAME CREATE "beexl.png";
CANVAS 150,1000;
FORMAT rgba;
BACKGROUND  rgba(134,12,250,76);
var vect1:vector = vector(15,32);
```

```
var vect2:vector = vector(0,123);
var vect3:vector = vector(vect1.x,vect2.y - 15);
var vect4:vector;
var color1:rgba = rgba(15,12,234,23);
var color2:rgba = rgba(0,0,0,10);
var color3:rgba = rgba(100,123,230,100);
var color4:rgba = rgba(
                                color1.r * 3,
                                color2.g / 2,
                                color3.a - 10,
                                color1.a +
color2.a
                            );

fun void createHorizontalLine(
        startPos: Vector,
        myColor: RGBA,
        length: number
    )
{

        var endPos = vector(

startPos.x,

startPos.y + length
                                        );

        from startPos to endPos do
        {
```

```
            fill startPos myColor;
        }
}

fun Vector obtainAverage(
    firstVector:vector,
    secondVector:vector
)
{
    var tempVector = vector(
        (firstVector.x + secondVectorx) / 2,
        (secondVector.y + secondVector.y) / 2
    )
    return tempVector;
}


fun void main()
{
    var firstAverage:vector =
obtainAverage(vect1,vect2);

    vect4 = obtainAverage(firstAverage,vector3);
    if((vect4.y > 200) && ((vector4.x < 10) ||
(vector4.x > 100)){

        createHorizontalLine(vect4,10,color1);
        createHorizontalLine(vect3,20,color2);
        createHorizontalLine(vect2,30,color3);
        createHorizontalLine(vect1,40,color4);
```

```
    }else if(vect4.y > 100){

        createHorizontalLine(vect4,20,color1);
        createHorizontalLine(vect3,30,color2);
        createHorizontalLine(vect2,40,color3);
        createHorizontalLine(vect1,50,color4);

    }else if(vect4.y > 50){

        createHorizontalLine(vect4,30,color1);
        createHorizontalLine(vect3,40,color2);
        createHorizontalLine(vect2,50,color3);
        createHorizontalLine(vect1,60,color4);

    }else{
        createHorizontalLine(vect4,60,color1);
        createHorizontalLine(vect3,40,color2);
        createHorizontalLine(vect2,10,color3);
        createHorizontalLine(vect1,100,color5);
    }

}
```

```
FILENAME READ "beexl.png";
```

```
FORMAT rgba;
var startPos: vector = vector(0,0);
var endPos: vector = vector(
            CANVAS_WIDTH,
            CANVAS_HEIGHT
        );


fun void darkenColors(
    rate:number,
    alphaRate:number
)
{
    var tempStart:vector=startPos;
    var tempColor:rgba;
    from tempStart to endPos do{
        tempColor = COLOR_AT(tempStart);
        tempColor.r = tempColor.r - number;
        tempColor.b = tempColor.b - number;
        tempColor.g = tempColor.g - number;
        tempColor.a = tempColor.a - alphaRate;
    }
}

fun void lightenColors(
    rate: number,
    alphaRate:number
)
{
    var tempStart:vector=startPos;
```

```
    var tempColor:rgba;
    from tempStart to endPos do{
        tempColor = COLOR_AT(tempStart);
        tempColor.r = tempColor.r + number;
        tempColor.b = tempColor.b + number;
        tempColor.g = tempColor.g + number;
        tempColor.a = tempColor.a + alphaRate;
    }
}



fun void main()
{
    var rate:number = 15;
    var alphaRate:number = 5;
    if(endPos.x > 100){
        darkenColors(rate,alphaRate);
    }else{
        lightenColors(rate,alphaRate);
    }
    print;
}
```

Main Semantic characteristics

Int definitions at rgba must be within 0-255 (inclusive).

The string that follows the filename must end with a valid image extension.

A var must be declared in order to be assigned a different value than the one it was originally assigned to, or to be used in a conditional, cycle, PixelFill or PixelShape.

Brief description of every special functions as well as rarely used instructions in your language

For

The structure of the loop in our programming language is as follows:
for x to y do { instructions }

This loop uses a vector or color to repeat a set of instructions while the initial vector or color values are different from the target one.

Examples:

```
from vector(10,15) to vector(15,15) do {
    PixelFill vector(10,10)
}
```

in this case the loop will check if the first vector'x value is different from the target's one. If they are different, it will first sum or subtract one value per iteration until both vectors are equal. The same happens to colors.

CANVAS_WIDTH → returns the width of the canvas
CANVAS_HEIGHT → returns the height of the canvas
MAX_RED → 255
MAX_BLUE → 255
MAX_GREEN → 255
MAX_ALPHA → 100

Data Types

Types the user can declare

rgba → RGBA representation of a color
vector → one-dimensional-two-element array that represents a coordinate within the canvas.

Types the user may use but won't be able to declare

number → non-decimal number within the range of 0 to 2^31-1
bool → true or false

holi

Glossary

data type	abbreviation
vector	v
vector x attribute	v.x
vector y attribute	v.y

number	n
rgba	r
bool	b
rgba red attribute	r.r
green attribute	r.g
blue attribute	r.b
alpha attribute	r.a
string	s
ERROR	e

Semantic Cube

Vector

Operator 1	Operator 2	operand	result
v	v	+	v
v	v.x	+	v
v	v.y	+	v
v	n	+	e
v	r	+	e
v	h	+	e
v	b	+	e
v	r.r	+	e
v	r.g	+	e
v	r.b	+	e
v	r.a	+	e
v	h.a	+	e
v	h.g	+	e
v	h.b	+	e
v	s	+	e
v	v	-	v
v	v.x	-	v
v	v.y	-	v
v	n	-	e
v	r	-	e
v	h	-	e
v	b	-	e

v	r.r	-	e
v	r.g	-	e
v	r.b	-	e
v	r.a	-	e
v	h.a	-	e
v	h.g	-	e
v	h.b	-	e
v	s	-	e
v	v	*	v
v	v.x	*	v
v	v.y	*	v
v	n	*	e
v	r	*	e
v	h	*	e
v	b	*	e
v	r.r	*	e
v	r.g	*	e
v	r.b	*	e
v	r.a	*	e
v	h.a	*	e
v	h.g	*	e
v	h.b	*	e
v	s	*	e
v	v	/	v
v	v.x	/	v
v	v.y	/	v
v	n	/	e
v	r	/	e
v	h	/	e
v	b	/	e
v	r.r	/	e
v	r.g	/	e
v	r.b	/	e
v	r.a	/	e
v	h.a	/	e
v	h.g	/	e
v	h.b	/	e

v	s	/	e
v	v	> < <= >= ==	b
v	v.x	> < <= >= ==	b
v	v.y	> < <= >= ==	b
v	n	> < <= >= ==	e
v	r	> < <= >= ==	e
v	h	> < <= >= ==	e
v	b	> < <= >= ==	e
v	r.r	> < <= >= ==	e
v	r.g	> < <= >= ==	e
v	r.b	> < <= >= ==	e
v	r.a	> < <= >= ==	e
v	h.a	> < <= >= ==	e
v	h.g	> < <= >= ==	e
v	h.b	> < <= >= ==	e

Vector X Attribute

Operator 1	Operator 2	operand	result
v.x	v	+	v
v.x	v.x	+	v
v.x	v.y	+	v
v.x	n	+	v
v.x	r	+	e
v.x	h	+	e
v.x	b	+	e
v.x	r.r	+	v
v.x	r.g	+	v
v.x	r.b	+	v
v.x	r.a	+	v
v.x	h.a	+	v
v.x	h.g	+	v
v.x	h.b	+	v
v.x	s	+	e
v.x	v	-	v
v.x	v.x	-	v
v.x	v.y	-	v

v.x	n	-	v
v.x	r	-	e
v.x	h	-	e
v.x	b	-	e
v.x	r.r	-	v
v.x	r.g	-	v
v.x	r.b	-	v
v.x	r.a	-	v
v.x	h.a	-	v
v.x	h.g	-	v
v.x	h.b	-	v
v.x	s	-	e
v.x	v	*	v
v.x	v.x	*	v
v.x	v.y	*	v
v.x	n	*	v
v.x	r	*	e
v.x	h	*	e
v.x	b	*	e
v.x	r.r	*	v
v.x	r.g	*	v
v.x	r.b	*	v
v.x	r.a	*	v
v.x	h.a	*	v
v.x	h.g	*	v
v.x	h.b	*	v
v.x	s	*	e
v.x	v	/	v
v.x	v.x	/	v
v.x	v.y	/	v
v.x	n	/	v
v.x	r	/	e
v.x	h	/	e
v.x	b	/	e
v.x	r.r	/	v
v.x	r.g	/	v
v.x	r.b	/	v

v.x	r.a	/	v
v.x	h.a	/	v
v.x	h.g	/	v
v.x	h.b	/	v
v.x	s	/	e
v.x	v	> < <= >= ==	b
v.x	v.x	> < <= >= ==	b
v.x	v.y	> < <= >= ==	b
v.x	n	> < <= >= ==	e
v.x	r	> < <= >= ==	e
v.x	h	> < <= >= ==	e
v.x	b	> < <= >= ==	e
v.x	r.r	> < <= >= ==	e
v.x	r.g	> < <= >= ==	e
v.x	r.b	> < <= >= ==	e
v.x	r.a	> < <= >= ==	e
v.x	h.a	> < <= >= ==	e
v.x	h.g	> < <= >= ==	e
v.x	h.b	> < <= >= ==	e

Vector Y Attribute

Operator 1	Operator 2	operand	result
v.y	v	+	v
v.y	v.x	+	v
v.y	v.y	+	v
v.y	n	+	v
v.y	r	+	e
v.y	h	+	e
v.y	b	+	e
v.y	r.r	+	v
v.y	r.g	+	v
v.y	r.b	+	v
v.y	r.a	+	v
v.y	h.a	+	v
v.y	h.g	+	v
v.y	h.b	+	v

v.y	s	+	e
v.y	v	-	v
v.y	v.x	-	v
v.y	v.y	-	v
v.y	n	-	v
v.y	r	-	e
v.y	h	-	e
v.y	b	-	e
v.y	r.r	-	v
v.y	r.g	-	v
v.y	r.b	-	v
v.y	r.a	-	v
v.y	h.a	-	v
v.y	h.g	-	v
v.y	h.b	-	v
v.y	s	-	e
v.y	v	*	v
v.y	v.x	*	v
v.y	v.y	*	v
v.y	n	*	v
v.y	r	*	e
v.y	h	*	e
v.y	b	*	e
v.y	r.r	*	v
v.y	r.g	*	v
v.y	r.b	*	v
v.y	r.a	*	v
v.y	h.a	*	v
v.y	h.g	*	v
v.y	h.b	*	v
v.y	s	*	e
v.y	v	/	v
v.y	v.x	/	v
v.y	v.y	/	v
v.y	n	/	v
v.y	r	/	e
v.y	h	/	e

v.y	b	/	e
v.y	r.r	/	v
v.y	r.g	/	v
v.y	r.b	/	v
v.y	r.a	/	v
v.y	h.a	/	v
v.y	h.g	/	v
v.y	h.b	/	v
v.y	s	/	e
v.y	v	> < <= >= ==	b
v.y	v.x	> < <= >= ==	b
v.y	v.y	> < <= >= ==	b
v.y	n	> < <= >= ==	e
v.y	r	> < <= >= ==	e
v.y	h	> < <= >= ==	e
v.y	b	> < <= >= ==	e
v.y	r.r	> < <= >= ==	e
v.y	r.g	> < <= >= ==	e
v.y	r.b	> < <= >= ==	e
v.y	r.a	> < <= >= ==	e
v.y	h.a	> < <= >= ==	e
v.y	h.g	> < <= >= ==	e
v.y	h.b	> < <= >= ==	e

Number

Operator 1	Operator 2	operand	result
n	v	+	e
n	v.x	+	n
n	v.y	+	n
n	n	+	n
n	r	+	e
n	h	+	e
n	b	+	e
n	r.r	+	n
n	r.g	+	n
n	r.b	+	n

n	r.a	+	n
n	h.a	+	n
n	h.g	+	n
n	h.b	+	n
n	s	+	e
n	v	-	e
n	v.x	-	n
n	v.y	-	n
n	n	-	n
n	r	-	e
n	h	-	e
n	b	-	e
n	r.r	-	n
n	r.g	-	n
n	r.b	-	n
n	r.a	-	n
n	h.a	-	n
n	h.g	-	n
n	h.b	-	n
n	s	-	e
n	v	*	e
n	v.x	*	n
n	v.y	*	n
n	n	*	n
n	r	*	e
n	h	*	e
n	b	*	e
n	r.r	*	n
n	r.g	*	n
n	r.b	*	n
n	r.a	*	n
n	h.a	*	n
n	h.g	*	n
n	h.b	*	n
n	s	*	e
n	v	/	e
n	v.x	/	n

n	v.y	/	n
n	n	/	n
n	r	/	e
n	h	/	e
n	b	/	e
n	r.r	/	n
n	r.g	/	n
n	r.b	/	n
n	r.a	/	n
n	h.a	/	n
n	h.g	/	n
n	h.b	/	n
n	s	/	e
n	v	> < <= >= ==	e
n	v.x	> < <= >= ==	b
n	v.y	> < <= >= ==	b
n	n	> < <= >= ==	b
n	r	> < <= >= ==	e
n	h	> < <= >= ==	e
n	b	> < <= >= ==	e
n	r.r	> < <= >= ==	b
n	r.g	> < <= >= ==	b
n	r.b	> < <= >= ==	b
n	r.a	> < <= >= ==	b
n	h.a	> < <= >= ==	b
n	h.g	> < <= >= ==	b
n	h.b	> < <= >= ==	b
n	s	> < <= >= ==	e

RBGA

Operator 1	Operator 2	operand	result
r	v	+	e
r	v.x	+	e
r	v.y	+	e
r	n	+	e
r	r	+	r

r	h	+	e
r	b	+	e
r	r.r	+	e
r	r.g	+	r
r	r.b	+	r
r	r.a	+	r
r	h.a	+	e
r	h.g	+	e
r	h.b	+	e
r	s	+	e
r	v	-	e
r	v.x	-	e
r	v.y	-	e
r	n	-	e
r	r	-	r
r	h	-	e
r	b	-	e
r	r.r	-	e
r	r.g	-	r
r	r.b	-	r
r	r.a	-	r
r	h.a	-	e
r	h.g	-	e
r	h.b	-	e
r	s	-	e
r	v	*	e
r	v.x	*	e
r	v.y	*	e
r	n	*	e
r	r	*	r
r	h	*	e
r	b	*	e
r	r.r	*	e
r	r.g	*	r
r	r.b	*	r
r	r.a	*	r
r	h.a	*	e

r	h.g	*	e
r	h.b	*	e
r	s	*	e
r	v	/	e
r	v.x	/	e
r	v.y	/	e
r	n	/	e
r	r	/	r
r	h	/	e
r	b	/	e
r	r.r	/	e
r	r.g	/	r
r	r.b	/	r
r	r.a	/	r
r	h.a	/	e
r	h.g	/	e
r	h.b	/	e
r	s	/	e
r	v	> < <= >= ==	e
r	v.x	> < <= >= ==	e
r	v.y	> < <= >= ==	e
r	n	> < <= >= ==	e
r	r	> < <= >= ==	b
r	h	> < <= >= ==	e
r	b	> < <= >= ==	e
r	r.r	> < <= >= ==	b
r	r.g	> < <= >= ==	b
r	r.b	> < <= >= ==	b
r	r.a	> < <= >= ==	b
r	h.a	> < <= >= ==	e
r	h.g	> < <= >= ==	e
r	h.b	> < <= >= ==	e
r	s	> < <= >= ==	e

RGBA Red Attribute

Operator 1	Operator 2	operand	result
r.r	v	+	e
r.r	v.x	+	e
r.r	v.y	+	e
r.r	n	+	e
r.r	r	+	r
r.r	h	+	e
r.r	b	+	e
r.r	r.r	+	r
r.r	r.g	+	r
r.r	r.b	+	r
r.r	r.a	+	r
r.r	h.a	+	e
r.r	h.g	+	e
r.r	h.b	+	e
r.r	s	+	e
r.r	v	-	e
r.r	v.x	-	e
r.r	v.y	-	e
r.r	n	-	e
r.r	r	-	r
r.r	h	-	e
r.r	b	-	e
r.r	r.r	-	r
r.r	r.g	-	r
r.r	r.b	-	r
r.r	r.a	-	r
r.r	h.a	-	e
r.r	h.g	-	e
r.r	h.b	-	e
r.r	s	-	e
r.r	v	*	e
r.r	v.x	*	e
r.r	v.y	*	e
r.r	n	*	e
r.r	r	*	r
r.r	h	*	e

r.r	b	*	e
r.r	r.r	*	r
r.r	r.g	*	r
r.r	r.b	*	r
r.r	r.a	*	r
r.r	h.a	*	e
r.r	h.g	*	e
r.r	h.b	*	e
r.r	s	*	e
r.r	v	/	e
r.r	v.x	/	e
r.r	v.y	/	e
r.r	n	/	e
r.r	r	/	r
r.r	h	/	e
r.r	b	/	e
r.r	r.r	/	r
r.r	r.g	/	r
r.r	r.b	/	r
r.r	r.a	/	r
r.r	h.a	/	e
r.r	h.g	/	e
r.r	h.b	/	e
r.r	s	/	e
r.r	v	> < <= >= ==	e
r.r	v.x	> < <= >= ==	e
r.r	v.y	> < <= >= ==	e
r.r	n	> < <= >= ==	b
r.r	r	> < <= >= ==	b
r.r	h	> < <= >= ==	e
r.r	r.r	> < <= >= ==	b
r.r	r.g	> < <= >= ==	b
r.r	r.b	> < <= >= ==	b
r.r	r.a	> < <= >= ==	b
r.r	h.a	> < <= >= ==	e
r.r	h.g	> < <= >= ==	e
r.r	h.b	> < <= >= ==	e

RGBA Green Attribute

Operator 1	Operator 2	operand	result
r.g	v	+	e
r.g	v.x	+	e
r.g	v.y	+	e
r.g	n	+	e
r.g	r	+	r
r.g	h	+	e
r.g	b	+	e
r.g	r.r	+	r
r.g	r.g	+	r
r.g	r.b	+	r
r.g	r.a	+	r
r.g	h.a	+	e
r.g	h.g	+	e
r.g	h.b	+	e
r.g	s	+	e
r.g	v	-	e
r.g	v.x	-	e
r.g	v.y	-	e
r.g	n	-	e
r.g	r	-	r
r.g	h	-	e
r.g	b	-	e
r.g	r.r	-	r
r.g	r.g	-	r
r.g	r.b	-	r
r.g	r.a	-	r
r.g	h.a	-	e
r.g	h.g	-	e
r.g	h.b	-	e
r.g	s	-	e
r.g	v	*	e
r.g	v.x	*	e

r.g	v.y	*	e
r.g	n	*	e
r.g	r	*	r
r.g	h	*	e
r.g	b	*	e
r.g	r.r	*	r
r.g	r.g	*	r
r.g	r.b	*	r
r.g	r.a	*	r
r.g	h.a	*	e
r.g	h.g	*	e
r.g	h.b	*	e
r.g	s	*	e
r.g	v	/	e
r.g	v.x	/	e
r.g	v.y	/	e
r.g	n	/	e
r.g	r	/	r
r.g	h	/	e
r.g	b	/	e
r.g	r.r	/	r
r.g	r.g	/	r
r.g	r.b	/	r
r.g	r.a	/	r
r.g	h.a	/	e
r.g	h.g	/	e
r.g	h.b	/	e
r.g	s	/	e
r.g	v	> < <= >= ==	e
r.g	v.x	> < <= >= ==	e
r.g	v.y	> < <= >= ==	e
r.g	n	> < <= >= ==	b
r.g	r	> < <= >= ==	b
r.g	h	> < <= >= ==	e
r.g	r.r	> < <= >= ==	b
r.g	r.g	> < <= >= ==	b
r.g	r.b	> < <= >= ==	b

r.g	r.a	> < <= >= ==	b
r.g	h.a	> < <= >= ==	e
r.g	h.g	> < <= >= ==	e
r.g	h.b	> < <= >= ==	e

RGBA Blue Attribute

Operator 1	Operator 2	operand	result
r.b	v	+	e
r.b	v.x	+	e
r.b	v.y	+	e
r.b	n	+	e
r.b	r	+	r
r.b	h	+	e
r.b	b	+	e
r.b	r.r	+	r
r.b	r.g	+	r
r.b	r.b	+	r
r.b	r.a	+	r
r.b	h.a	+	e
r.b	h.g	+	e
r.b	h.b	+	e
r.b	s	+	e
r.b	v	-	e
r.b	v.x	-	e
r.b	v.y	-	e
r.b	n	-	e
r.b	r	-	r
r.b	h	-	e
r.b	b	-	e
r.b	r.r	-	r
r.b	r.g	-	r
r.b	r.b	-	r
r.b	r.a	-	r
r.b	h.a	-	e
r.b	h.g	-	e

r.b	h.b	-	e
r.b	s	-	e
r.b	v	*	e
r.b	v.x	*	e
r.b	v.y	*	e
r.b	n	*	e
r.b	r	*	r
r.b	h	*	e
r.b	b	*	e
r.b	r.r	*	r
r.b	r.g	*	r
r.b	r.b	*	r
r.b	r.a	*	r
r.b	h.a	*	e
r.b	h.g	*	e
r.b	h.b	*	e
r.b	s	*	e
r.b	v	/	e
r.b	v.x	/	e
r.b	v.y	/	e
r.b	n	/	e
r.b	r	/	r
r.b	h	/	e
r.b	b	/	e
r.b	r.r	/	r
r.b	r.g	/	r
r.b	r.b	/	r
r.b	r.a	/	r
r.b	h.a	/	e
r.b	h.g	/	e
r.b	h.b	/	e
r.b	s	/	e
r.b	v	> < <= >= ==	e
r.b	v.x	> < <= >= ==	e
r.b	v.y	> < <= >= ==	e
r.b	n	> < <= >= ==	b
r.b	r	> < <= >= ==	b

r.b	h	> < <= >= ==	e
r.b	r.r	> < <= >= ==	b
r.b	r.g	> < <= >= ==	b
r.b	r.b	> < <= >= ==	b
r.b	r.a	> < <= >= ==	b
r.b	h.a	> < <= >= ==	e
r.b	h.g	> < <= >= ==	e
r.b	h.b	> < <= >= ==	e

RGBA Alpha Attribute

Operator 1	Operator 2	operand	result
r.a	v	+	e
r.a	v.x	+	e
r.a	v.y	+	e
r.a	n	+	e
r.a	r	+	r
r.a	h	+	e
r.a	b	+	e
r.a	r.r	+	r
r.a	r.g	+	r
r.a	r.b	+	r
r.a	r.a	+	r
r.a	h.a	+	e
r.a	h.g	+	e
r.a	h.b	+	e
r.a	s	+	e
r.a	v	-	e
r.a	v.x	-	e
r.a	v.y	-	e
r.a	n	-	e
r.a	r	-	r
r.a	h	-	e
r.a	b	-	e
r.a	r.r	-	r
r.a	r.g	-	r

r.a	r.b	-	r
r.a	r.a	-	r
r.a	h.a	-	e
r.a	h.g	-	e
r.a	h.b	-	e
r.a	s	-	e
r.a	v	*	e
r.a	v.x	*	e
r.a	v.y	*	e
r.a	n	*	e
r.a	r	*	r
r.a	h	*	e
r.a	b	*	e
r.a	r.r	*	r
r.a	r.g	*	r
r.a	r.b	*	r
r.a	r.a	*	r
r.a	h.a	*	e
r.a	h.g	*	e
r.a	h.b	*	e
r.a	s	*	e
r.a	v	/	e
r.a	v.x	/	e
r.a	v.y	/	e
r.a	n	/	e
r.a	r	/	r
r.a	h	/	e
r.a	b	/	e
r.a	r.r	/	r
r.a	r.g	/	r
r.a	r.b	/	r
r.a	r.a	/	r
r.a	h.a	/	e
r.a	h.g	/	e
r.a	h.b	/	e
r.a	s	/	e
r.a	v	> < <= >= ==	e

r.a	v.x	> < <= >= ==	e
r.a	v.y	> < <= >= ==	e
r.a	n	> < <= >= ==	b
r.a	r	> < <= >= ==	b
r.a	h	> < <= >= ==	e
r.a	r.r	> < <= >= ==	b
r.a	r.g	> < <= >= ==	b
r.a	r.b	> < <= >= ==	b
r.a	r.a	> < <= >= ==	b
r.a	h.a	> < <= >= ==	e
r.a	h.g	> < <= >= ==	e
r.a	h.b	> < <= >= ==	e

Development Language and OS

Development language: Python
OS: Windows

Bibliography