INSTITUTO FEDERAL DO ESPÍRITO SANTO

DAVID PANTALEÃO VILAÇA
LUCAS GOMES FLEGLER
TADEU DA PENHA MORAES JUNIOR

PaintCode

Parse Linguagem

Serra
2019

DAVID PANTALEÃO VILAÇA
LUCAS GOMES FLEGLER
TADEU DA PENHA MORAES JUNIOR

PaintCode

Parse Linguagem

Trabalho apresentado na disciplina de Linguagens Formais e Autômatos no curso Bacharelado em Sistemas de Informação, do Instituto Federal do Espírito Santo - Campus Serra, orientado pelo docente Jefferson O. Andrade.

Sumário

1 - Introdução

O PaintCode é uma linguagem que foi desenvolvida no intuito de auxiliar na criação de desenhos, utilizando o módulo Turtle do Python. Para analisar a gramática livre de contexto do PaintCode, utilizamos o Lark, que tem como objetivo analisar qualquer gramática que for inserida.

2 - Estrutura do código Fonte

Organização dos arquivos

O código fonte encontra-se estruturado da seguinte maneira:

```
src
|_ main.py
|_ paint_code.py
|_ paint_code.sh
|_ grammar
   |_ grammar.lark
|_ testes
   |_ example-circle.pc
   |_ example-frac.pc
   |_ example-frac2.pc
   |_ example-if.pc
   |_ example-square.pc
   |_ example_star.pc
```

main.py

É o módulo principal do programa, que tem como objetivo lê a expressão digitada através de um REPL (Read–eval–print loop) ou um arquivo texto que identificamos com a extensão .pc e posteriormente envia para o módulo paint_code.py.

paint_code.py

É um módulo que contém uma classe única chamada paint_code, com a responsabilidade de manipular as instruções de desenho inseridas pelo programador.

paint_code.sh

Script de configuração de ambiente para realizar testes na linguagem.

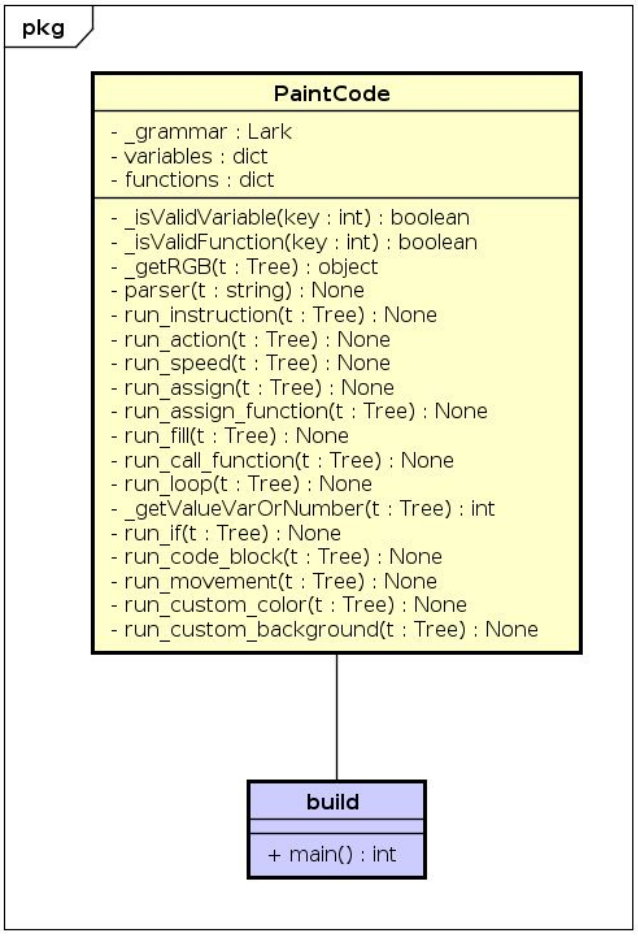
grammar.lark

Arquivo que contém toda a gramática do PaintCode, desenvolvida sobre a biblioteca do Lark.

Arquivos *.pc

Arquivos que contém exemplos prontos para testes da linguagem.

Diagrama de Classes



3 - Definição PaintCode DSL

Gramática no Lark (grammar.lark)

No arquivo grammar.lark contém a escrita no formato EBNF, que posteriormente é interpretada pela biblioteca Lark. Abaixo encontra-se a escrita da gramática contida no arquivo:

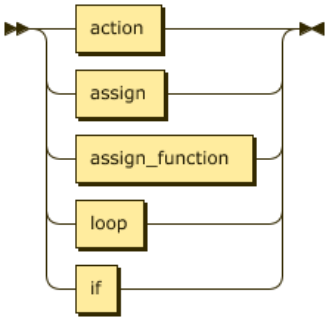
```
start: instruction

instruction: action                -> action
           |assign                 -> assign
           |assign_function        -> assign_function
           |loop                   -> loop
           |if                     -> if



action: movement                -> movement
       |custom_color            -> custom_color
       |custom_background       -> custom_background
       |clear                   -> clear
       |reset                   -> reset
       |fill                    -> fill
       |call_function           -> call_function
       |speed                   -> speed


speed: "speed" NUMBER
movement: "move" (DIRECTION (NUMBER | variable))+
fill: BEGINFILL | ENDFILL
custom_color: "color" (COLOR | rgb)
custom_background: "bg" (COLOR | rgb)
rgb: "rgb" (("0".."9")~3 | variable) " " (("0".."9")~3 | variable)
" " (("0".."9")~3 | variable)
clear: "clear"
reset: "reset"
assign: "var" NAME "=" NUMBER
assign_function: "def" NAME "{" instruction (";" instruction)* "}"
```

```
variable: NAME
call_function: NAME "(" ")"
loop: "repeat" (NUMBER | variable) code_block
code_block: "{" action (";" action)* "}"
if:   "if"   (NUMBER|variable)   CONDITION   (NUMBER|variable)   "{"
instruction (";" instruction)* "}"

COLOR: "red" | "green" | "blue" | "white" | "black"
DIRECTION: "f"|"b"|"l"|"r"
NUMBER: ("0".."9")+
NAME: ("a".."z")+
BEGINFILL: "begin-fill"
ENDFILL: "end-fill"
CONDITION: ">=" | "<=" | "!=" | "==" | ">" | "<"

%import common.WS_INLINE
%ignore WS_INLINE
```

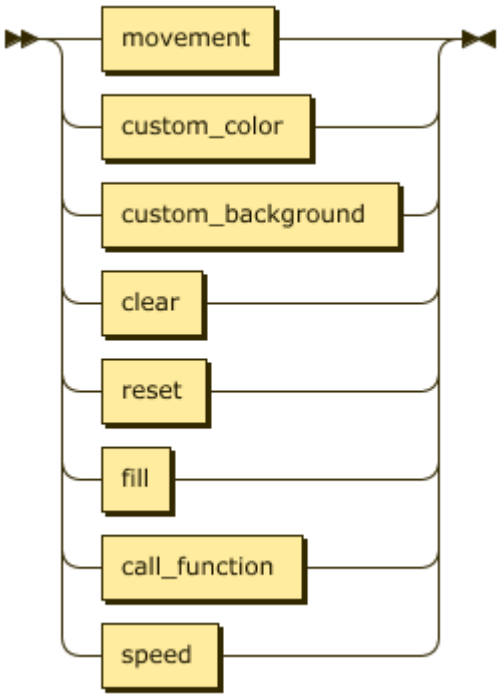
4 - Diagramas da Sintaxe

Start

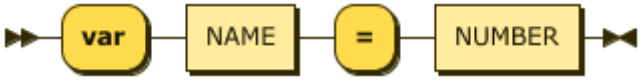
Instruction



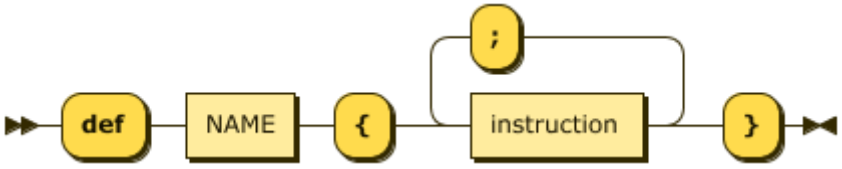
Action



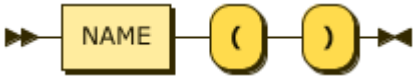
Atribuição de variável



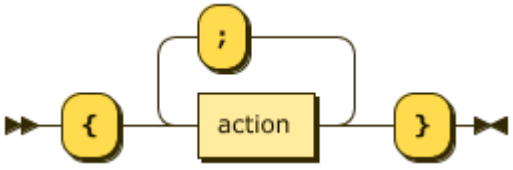
Declaração de função



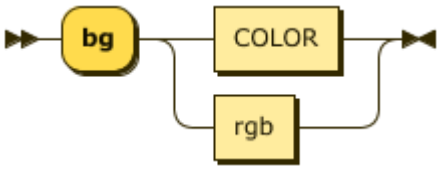
Chamada de função



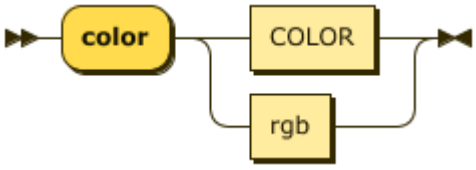
Bloco



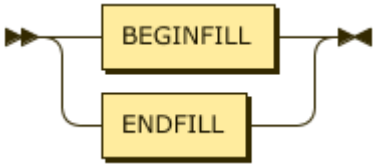
Cor de fundo



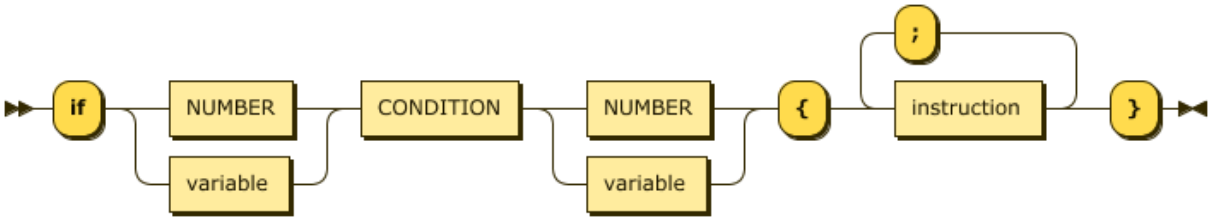
Cor do pincel



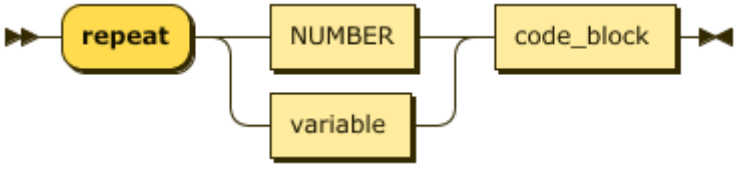
Preenchimento do desenho



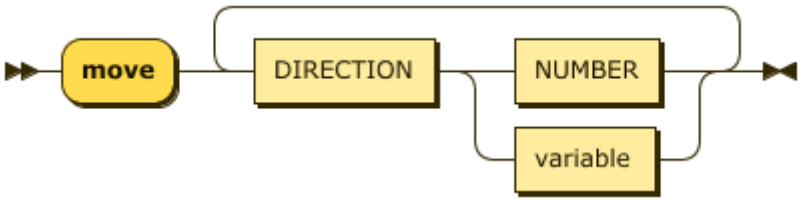
Estrutura de seleção



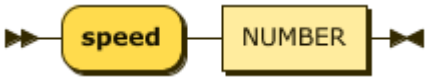
Estrutura de repetição



Movimento



Velocidade do pincel



Variável

5 - Executando o código

Configurando o ambiente

Para execução do projeto é recomendado utilizar uma máquina virtual do python, mas não é obrigatório para o funcionamento da linguagem.

Dentro do diretório “/src” existe um script básico para execução do programa. O comando abaixo mostra como executar.

```
source ./paint_code.sh
```

***Obs: Esse script instala algumas dependências via apt-get, tais como pip e tkinter. Portanto caso o script falhe tente atualizar a lista de repositórios locais com “sudo apt-get update” e logo após execute o script novamente.**

Caso o script acima falhe, os comandos a seguir mostram como configurar o ambiente do projeto:

- Insira o comando abaixo para instalar o gerenciador de biblioteca do python, o pip.

```
sudo apt-get -yqq install python3-pip python3-tk
```

- Insira o comando abaixo para criar uma máquina virtual do python

```
virtualenv venv --python=python3
```

- Para ativar a máquina insira o comando:

```
source venv/bin/activate
```

- Insira o comando abaixo para instalar a biblioteca do lark-parser

```
pip install lark-parser
```

Testando a linguagem

O código do PaintCode pode ser executado de duas maneiras diferentes, executando o arquivo main.py manualmente(esta opção o programador fica livre para escrever seu código) ou informando um arquivo com a extensão do PaintCode “arquivo.pc”.

Teste manual

Insira o comando abaixo dentro do diretório /src.

```
python main.py
```

- Após executar o comando acima a seguinte mensagem aparecerá no console:

```
insira a expressão -> |
insira a expressão -> | move f 100 l 5
```

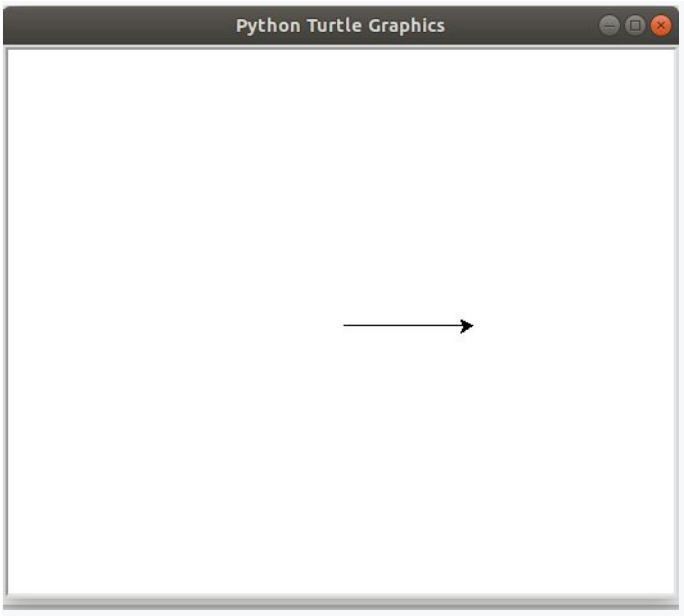


Figura 1 - Imagem gerada pelo código

Teste com arquivos

Todos os arquivos com extensão “.pc” se encontram no diretório “testes/”. Para executar estes arquivos, insira o comando abaixo:

```
python main.py --file testes/example-star.pc
```

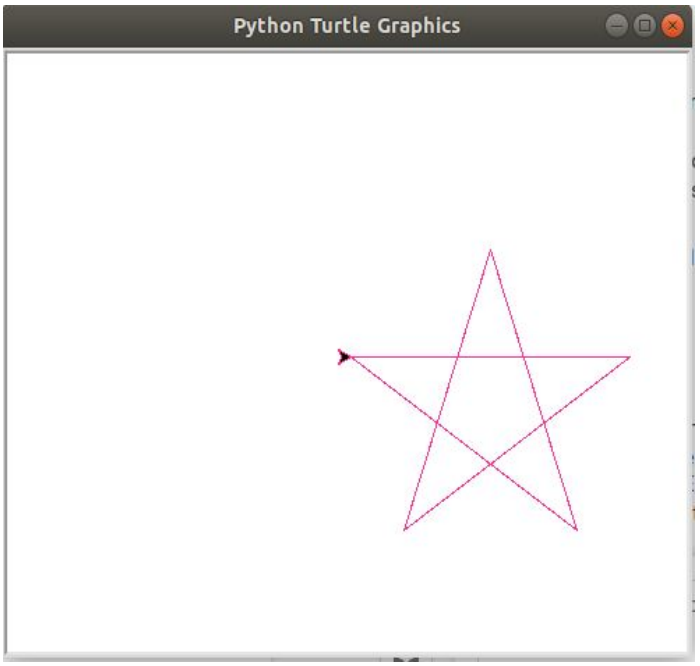


Figura 2 - Resultado do arquivo “example-star.pc”

6 - Comandos da linguagem

Velocidade de desenho

Para trocar a velocidade de movimento do pincel, utiliza-se o comando “speed” seguido de um número. Segue abaixo a descrição da velocidade, segundo a documentação do turtle.

- “fastest”: 0
- “fast”: 10
- “normal”: 6
- “slow”: 3
- “slowest”: 1

```
speed 0 // mais rapido
speed 3 // lento
speed 1 // mais lento
```

Movimento

No PaintCode, para executar a instrução mais simples de mover o pincel, utiliza-se o comando “move” acompanhado de "f, b, l, r" seguido de um número. Onde:

- f: move para frente;
- b: move para trás;
- l: move o ângulo da caneta para esquerda;
- r: move o ângulo da caneta para direita;

```
move f 100
```

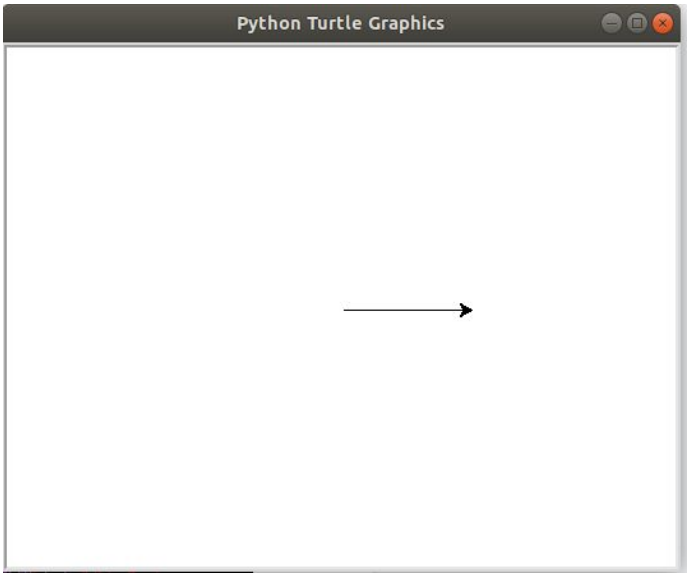


Figura 3 - movimento

Declaração de Variável

Para declaração de variáveis no código, é necessário que venha acompanhada da palavra “var”, o símbolo de igual “=” e um valor numérico. O Paint code aceita somente atribuições numérica em suas variáveis. Segue abaixo o código da figura 3 utilizando variável.

```
var passos = 100
var angulo = 0
move f passos l angulo
```

Para alterar o valor de uma variável já existente o processo é o mesmo de criação de uma variável nova.

Cor do pincel

Por definição a cor em que o pincel desenha é preto, mas caso o desenvolvedor queira mudar para outras cores, basta inserir o comando “color” seguindo por uma das cores padrões, que podem ser vermelho "red", verde "green", azul "blue", branco "white" ou preto "black".

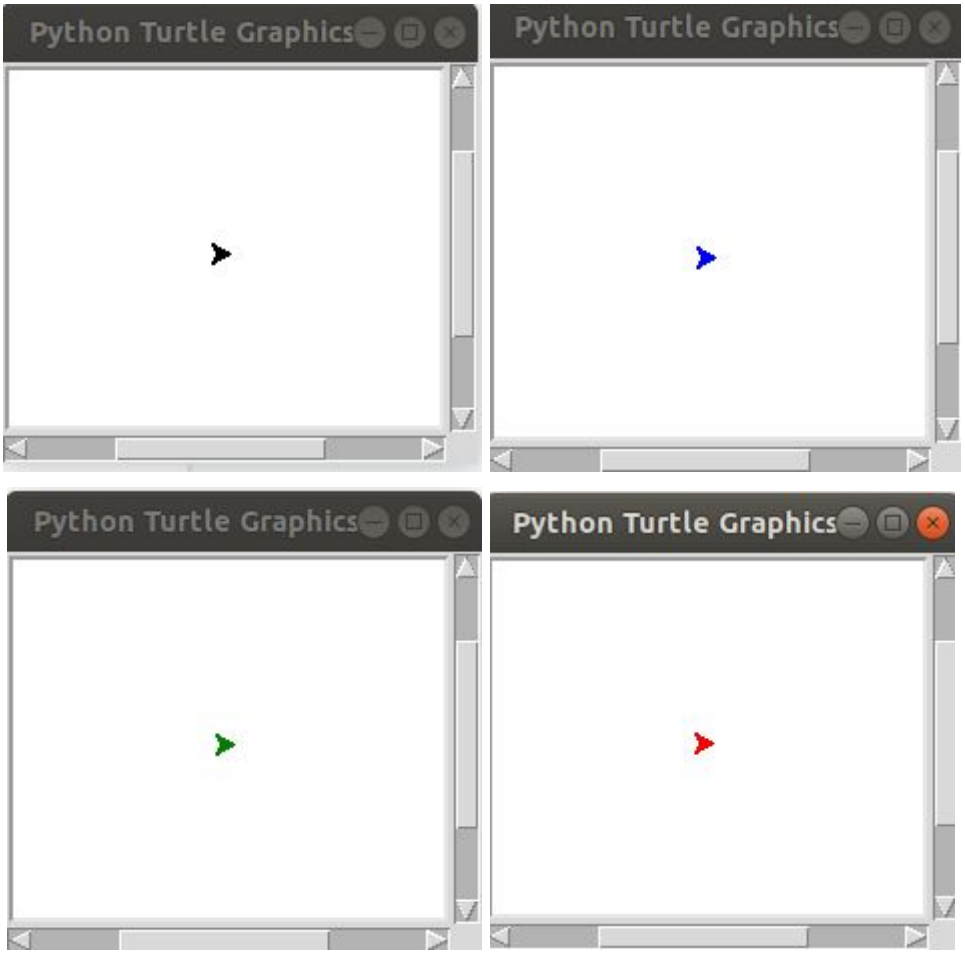


Figura 4 - cores do pincel

Caso queira uma cor personalizada o desenvolvedor pode usar a função rgb após o comando color, a sua estrutura deve ser da seguinte maneira: “color” “rgb” e **obrigatoriamente três número** de 000 a 255, veja o exemplo a seguir:

```
color rgb 255 000 255
```

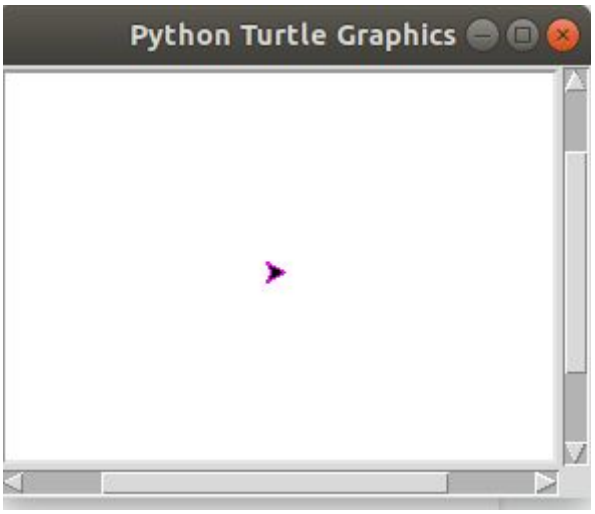


Figura 5 - pincel cor rgb rosa

Ao mudar a cor do pincel cada linha desenhada será de uma cor diferente, como pode ser visto no exemplo a seguir:

```
move f 100 r 90
color rgb 255 000 255
move f 100 r 90
color red
move f 100 r 90
color green
move f 100 r 90
color blue
move f0 r 45 f 150
```

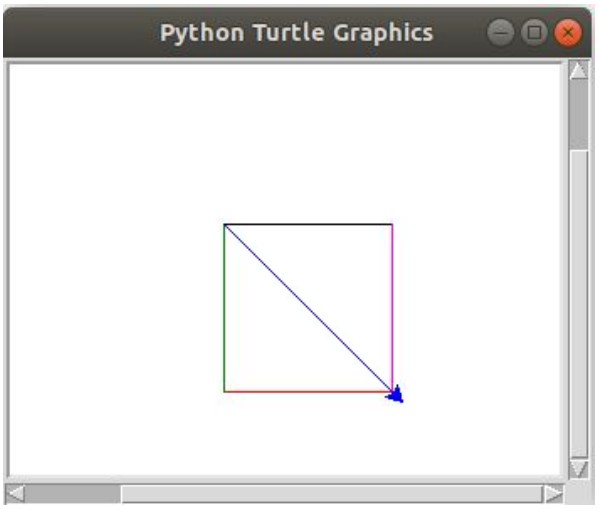


Figura 6 - quadrado com cores citadas

Preenchimento de desenho

No PaintCode, os desenhos também podem ser preenchidos pelas cores pré definidas ou com o rgb. O preenchimento funciona da seguinte maneira: antes de desenhar uma determinada figura é preciso colocar o comando begin-fill, que sinaliza o início do desenho. Após o desenvolvedor desenhar a(as) figura(as), deve ser informado o comando end-fill, que sinaliza o fim do preenchimento, que utilizará a última cor definida. Veja no exemplo a seguir:

```
1  bg black                          ←definição de background
2  color rgb 255 185 053             ← definição cor do pincel
3  begin-fill                        ← iniciando bloco do preenchimento
4  move f 100 r 90                   ← instruções para o desenho
5  move f 100 r 90
6  move f 100 r 90
7  move f 100 r 90
8  color red                         ← escolha da cor a ser preenchida
9  end-fill                          ← fim do bloco de preenchimento
```

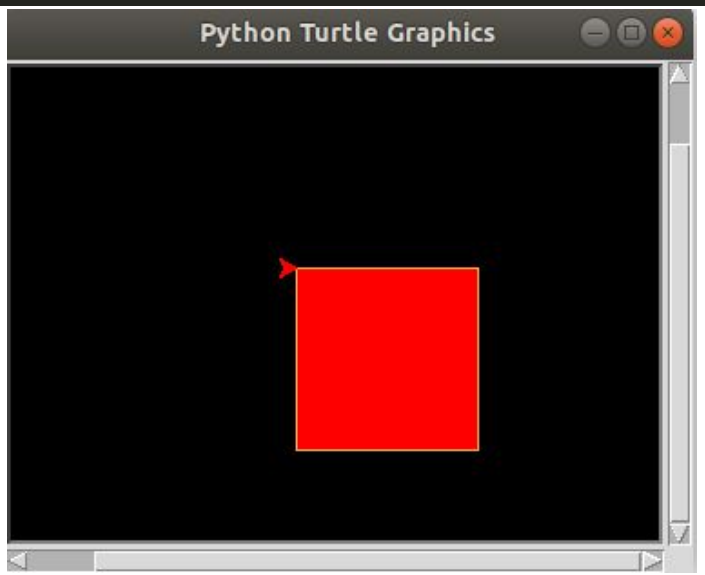


Figura 7 - Imagem de um desenho preenchido com uma cor

Declaração de Função

Para declarar uma função no código deve-se usar o seguinte formato: iniciar a declaração da função com a palavra “def”, inserir um nome, e escrever as instruções dentro do code block “{” ”}”. No código uma variável pode ter o mesmo nome de uma função, para diferenciar um do outro, deve-se utilizar os sinais de parêntese “(“ ”)” na chamada da função.

```
var tam = 100
var ang = 90
def quadrado { move f tam l ang f tam l ang f tam l ang f tam l ang }
bg red
begin-fill
quadrado()
color white
end-fill
move f 300
begin-fill
quadrado()
color blue
end-fill
```

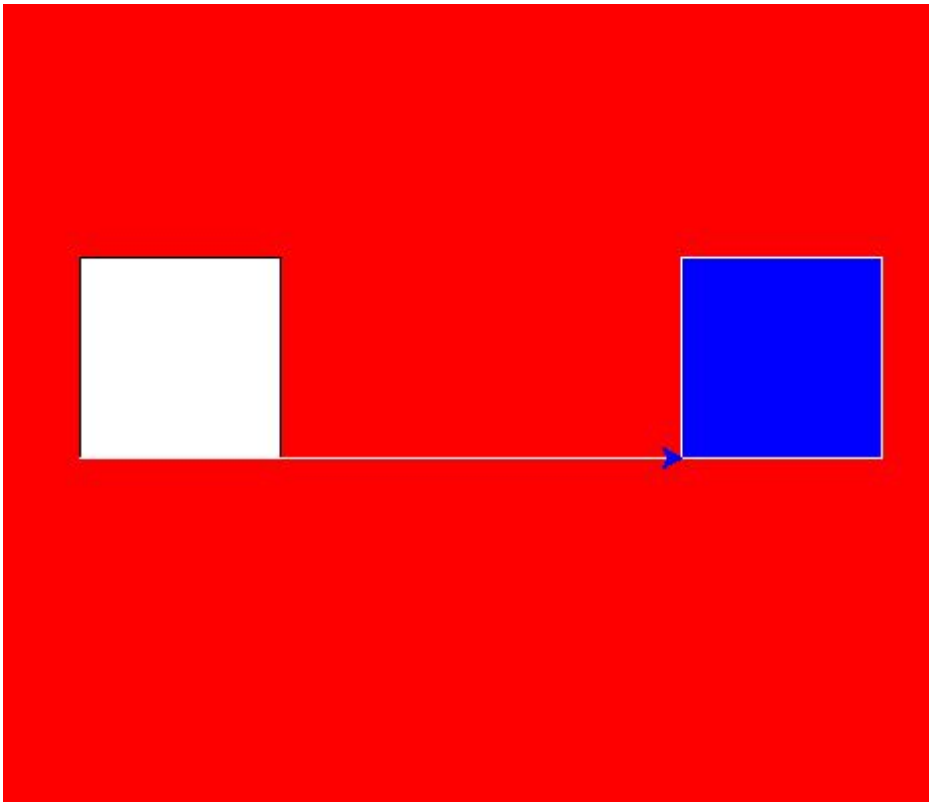


Figura 8 - criando função utilizando variáveis e preenchendo com cores os objeto

Estrutura de seleção

Para utilizar a estrutura de seleção no código deve-se usar o seguinte formato: iniciar a declaração da função com a palavra “if”, informar a condição lógica e escrever as instruções dentro do bloco “{” ”}”.

```
var x = 10
```

```
def square {color rgb 000 000 000; repeat 4 { move f 200 r 90 } }

if x < 20 { color blue }
if x < 5 { color black }
begin-fill
square()
end-fill

move f 200
color rgb 255 255 255
move f 200

if x == 10 { color red }
begin-fill
square()
end-fill

color rgb 000 000 000
move f 200 r 90 f 200 r 90 f 600 r 90 f 200 r 90 f 600
```

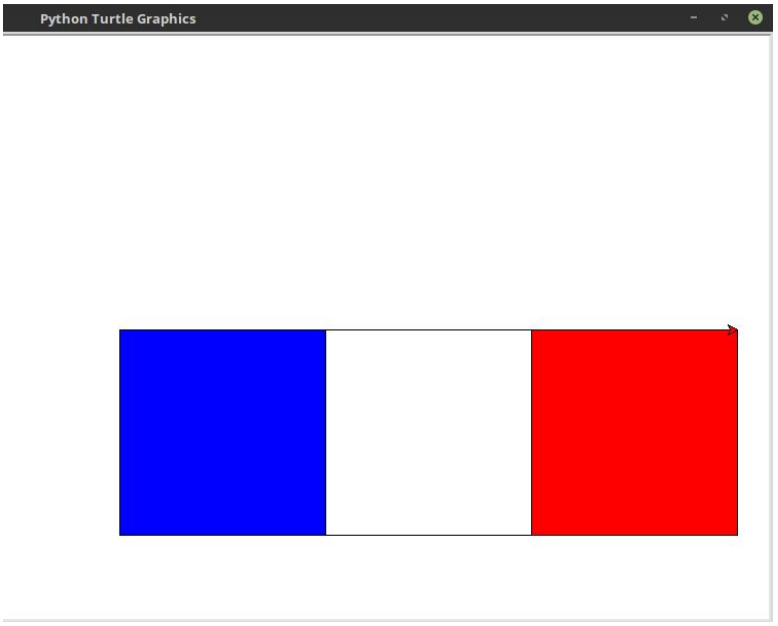


Figura 9 - Bandeira da França criada com if

Estrutura de repetição

Para utilizar a estrutura de repetição do PaintCode deve-se usar o comando *"repeat"* informando a quantidade de vezes que o código deve ser repetido, e escrever as instruções

dentro do bloco “{” ”}”. Tal comando pode ser visualizado no exemplo abaixo e na figura 10.

```
color rgb 255 048 142
begin-fill
repeat 5 {move f 200 r 144 }
end-fill
```

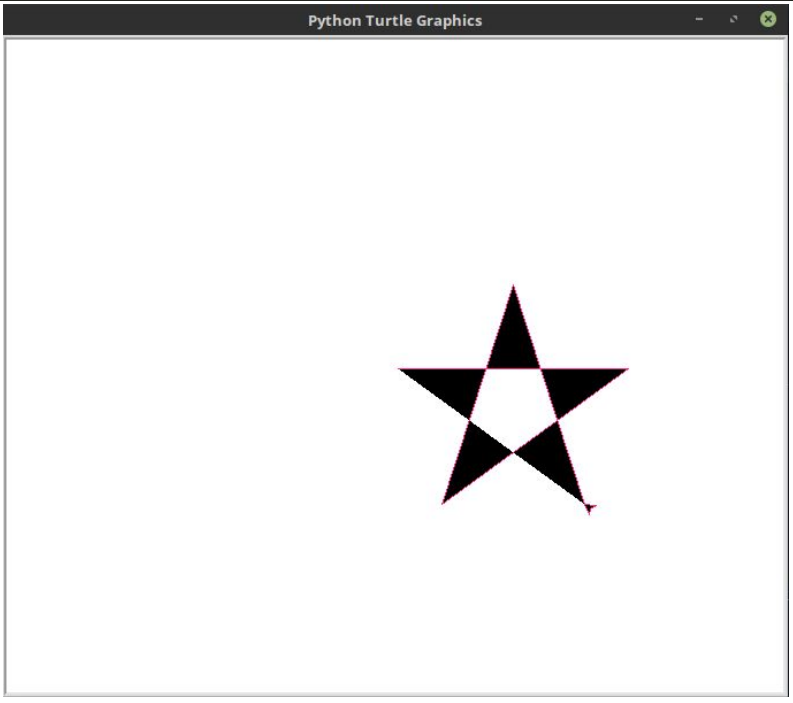


Figura 10- Estrela com estrutura de repetição

Utilizando todos os comandos do PaintCode

```
speed 0
var size = 200
var rptqtd = 36

def frac { move f size l 90 f size l 90 f size l 90 f size l 90 }
bg rgb 000 000 000

begin-fill
color red
if size > rptqtd { repeat rptqtd { frac(); move l 50 } }

var size = 150
color rgb 214 169 255
repeat rptqtd { frac(); move l 50 }

begin-fill
var size = 100
color rgb 211 119 039
repeat rptqtd { frac(); move l 50 }

var size = 50
color rgb 012 119 005
repeat rptqtd { frac(); move l 50 }

var size = 50
color rgb 255 000 255
repeat rptqtd { frac(); move l 50 }

var size = 25
color rgb 255 255 000
repeat rptqtd { frac(); move l 50 }

var size = 10
color rgb 00 255 000
repeat rptqtd { frac(); move l 50 }


color white
end-fill
```

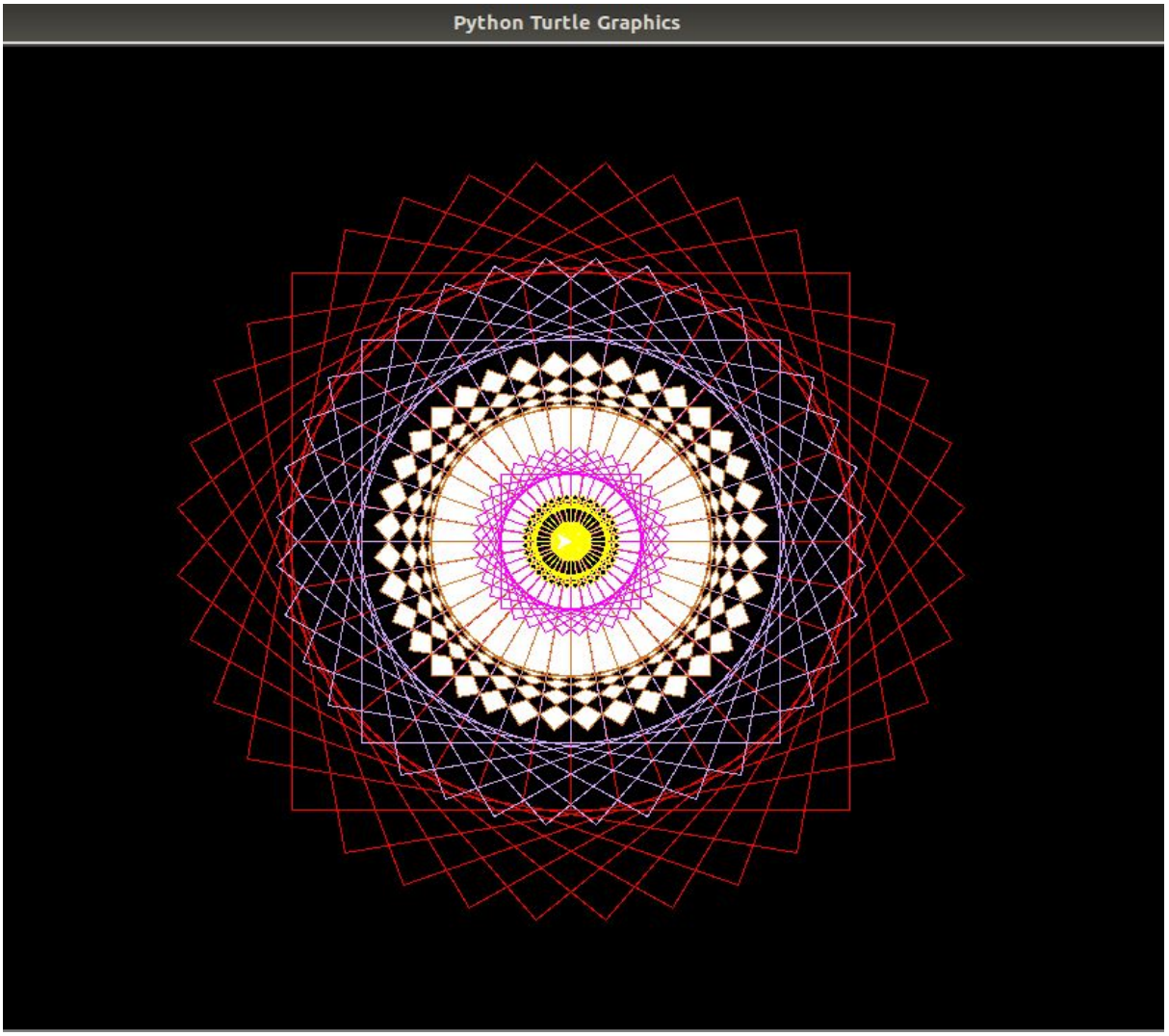


Figura 11 - fractal criado com todos os comando do PaintCode

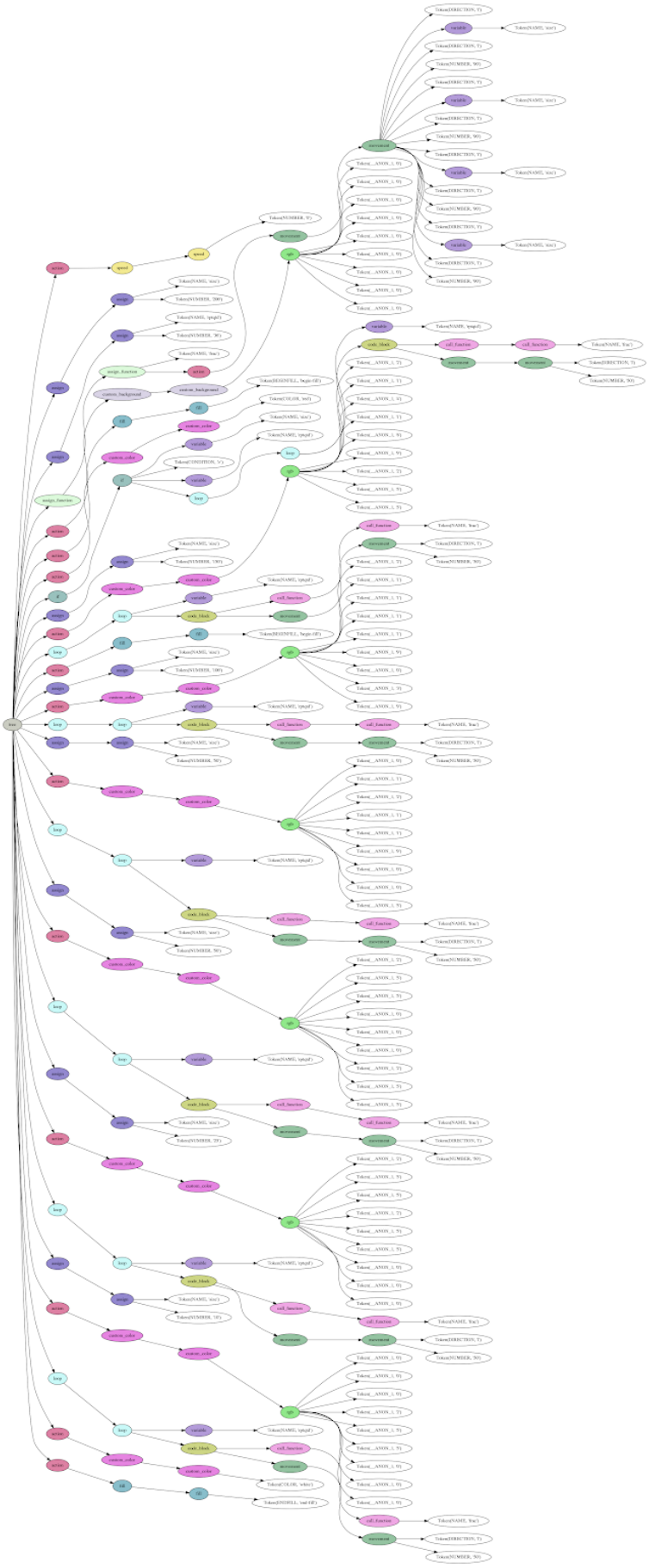


Figura 12 - Imagem AST gerada correspondente da execução do código da figura 11. (Link)

7 - Conclusão

Ao contrário das Linguagens de Propósito Geral, uma DSL possui expressividade bastante limitada, possibilitando ao usuário construir estruturas que modelam de forma clara e concisa funcionalidades específicas de um determinado domínio. A utilização de uma DSL possui algumas vantagens, sendo elas: Alto nível de abstração, eliminando detalhes desnecessários de implementação, Trechos de código escritos em uma DSL são claros, concisos e auto-documentados, dentre outras.

O PaintCode nos fez enxergar a importância desta disciplina para curso de Sistemas de Informação, onde desenvolvemos uma linguagem que facilita a programação voltada para desenho, fazendo com que o desenho fique simples e conciso. Da mesma forma esse raciocínio pode perfeitamente ser aplicado a outros contextos visando a resolução facilitada de um problema de domínio complexo.