Processamento de Linguagens:
Relatório do Trabalho Prático

Logo

Ricardo Sampaio nº 18827
Cláudio Silva nº 18843
Curso: LESI

1 Introdução

Neste capítulo falamos sobre a contextualização do trabalho prático, a motivação e os objetivos e, para terminar, descrevemos a estrutura do documento.

1.1 Contextualização

O Logo é uma linguagem de programação educativa, com origens em 1967. A linguagem é usada para controlar um cursor, normalmente representado por uma tartaruga, com o propósito de ensinar a "tartaruga" novos procedimentos além dos que ele já conhece, a fim de criar desenhos ou programas.

Pretende-se implementar um interpretador básico, capaz de simular o maior número possível de comandos da linguagem original.

1.2 Objetivos do trabalho prático

O objetivo deste trabalho prático é a criação de um programa em Python + Ply que analise um programa Logo (ficheiro de texto com uma ou mais linhas), e deve implementar os seguintes comandos:

- **fd** ou **forward** -> move a tartaruga n pixels para a frente. Ex: **fd 10**
- **bk** ou **back** - > move a tartaruga n pixels para trás. Ex: **bk 30**
- **lt** ou **left** -> roda a tartaruga para a esquerda, n graus. Ex: **lt 180**
- **rt** ou **right** -> roda a tartaruga para a direita, n graus. Ex: **rt 360**
- **setpos** -> define as posições x e y para qual a tartaruga se deve mover. Ex: **setpos [50 10]**
- **setxy** -> define as posições x e y para qual a tartaruga se deve mover. Ex: **setxy 50 20**

- **setx** -> define a posição x para qual a tartaruga se deve mover, mantendo a posição y. Ex: **setx 80**
- **sety** -> define a posição y para qual a tartaruga se deve mover, mantendo a posição x. Ex: **sety 20**

- **home** -> move a tartaruga para a posição (0, 0) e roda-a para a orientação original. Ex: **home**

- **pendown ou pd, penup ou pu** -> permite alternar entre o modo de desenho e modo de movimentação livre.

- **setpencolor** -> permite alterar as cores das linhas seguintes. Ex: **setpencolor [255 0 0]**

- **make** -> permite definir o valor de uma variável. Ex: **make “nomevar 5**

- **if, ifelse** -> permite definir estruturas condicionais.

- **repeat** -> repete um conjunto de comandos.

- **while** -> permite definir ciclos.

- **to** -> permite criar funções.

Exemplo de ficheiro **logo**:

```
make " teste 0
make " teste2 100

while [ : teste < teste2 ] [
    make " teste teste + 2
    fd teste
    rt 90
]
```

1.3 Estrutura do documento

Este documento está estruturado em três partes.
Na primeira falamos sobre a contextualização do trabalho, os objetivos e a estrutura do documento.

Na segunda falamos sobre a estrutura e funções do trabalho.

2 Implementação

Primeiramente implementamos o lexer e o yacc.
O lexer, que se encontra no ficheiro lexer.py, reconhece sequências de caracteres lidas do ficheiro e, se alguma dessas sequências corresponder a qualquer comando implementado no yacc, que se encontra no ficheiro yacc.py, esse comando é guardado e executado uma vez todos os comandos reconhecidos.

Os comandos que o yacc executa estão guardados no ficheiro Command.py e podem ser acedidos através de uma dispatch table.

Estes comandos podem executar várias funções tais como por exemplo:

- **Desenhar uma linha** dada a distância e o ângulo:

```
def do_line(command, parser):


   if(command.args['ahead'] == False):
       distance = 0 - parser.eval(command.args['distance'])
   else:
       distance = parser.eval(command.args['distance'])


   x2, y2 = Utils.Utils.CalculatePos(None, parser.x1, parser.y1,
parser.angle, distance)

   if(parser.penstate == True):
       Utils.Utils.DrawLine(None, parser, x2, y2)

   parser.x1 = x2
   parser.y1 = y2
```

- **Criar uma variável**:

```
def do_make_var(command, parser):
    var_name = command.args['target']
   if (len(command.args) == 2):
       var_value = parser.eval(command.args['value'])
  else:
   var_value = parser.eval({'left': command.args['left'], 'right':
command.args['right'], 'op': command.args['op']})
     parser.vars[var_name] = var_value
```

- **Criar um ciclo while:**

```
def do_while(command, parser):
   var_name = command.args['var']
   value = parser.eval(command.args['value'])
   if(command.args['signal'] == '>'):
       while parser.vars[var_name] > value:
           parser.execute(command.args['commands'])
   elif(command.args['signal'] == '<'):
       while parser.vars[var_name] < value:
           parser.execute(command.args['commands'])
```

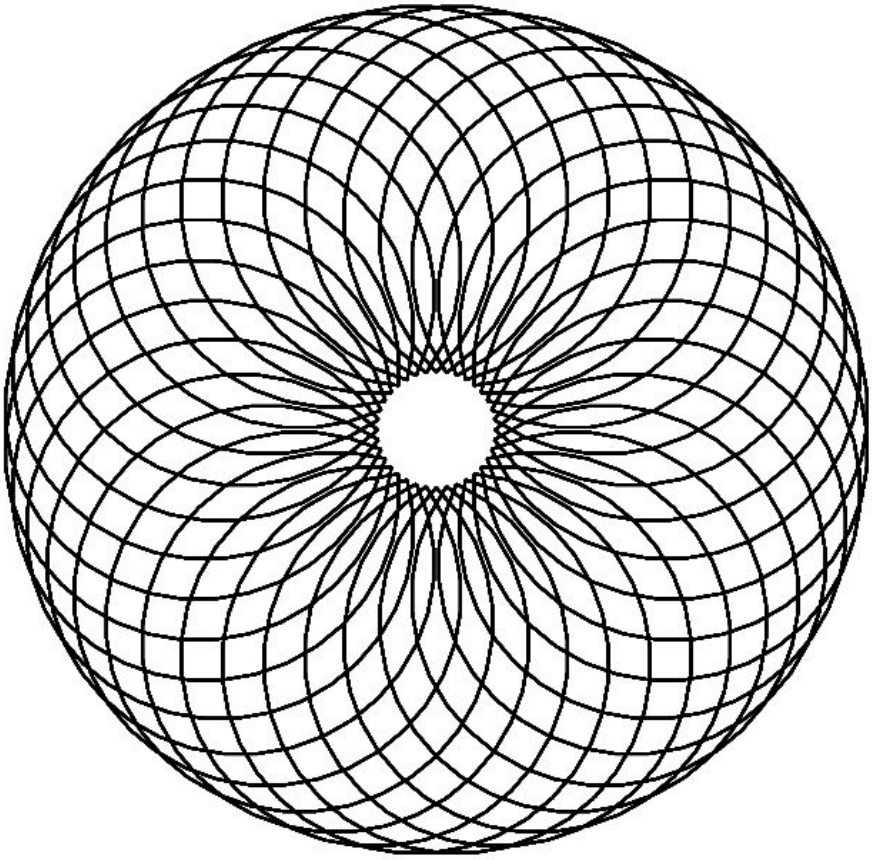
Entre outras...

O resultado deste programa são imagens .svg criadas a partir de um ficheiro .logo, tal como o exemplo seguinte:

```
to : rotatingcircle ( time1 time2 time3 ) [
    penup sety 170 setx 40 pendown
    repeat time1 [ repeat time2 [ fd time3 rt 10] rt 90]
]

rotatingcircle ( 400 34 12 )
```

O ficheiro acima cria uma função rotatingcircle e chama-a no fim dando origem à seguinte imagem:



Com a utilização de todos os comandos deste programa, é possível criar uma infinidade de imagens.

Conclusão

Com a realização deste trabalho conseguimos aprimorar os conhecimentos adquiridos durante a unidade curricular de Processamento de Linguagens, seja em termos da implementação de um interpretador lexer com o yacc, como na utilização da linguagem de programação Python.

Conseguimos implementar todos os comandos pedidos no enunciado no trabalho e implementar vários exemplos de ficheiros .logo.