**Licenciatura em Engenharia de Sistemas Informáticos  
Processamento de Linguagem   
  
Autores**

João Azevedo Nº18845

João Rodrigues Nº19431

Carlos Santos Nº19432

Professor: Alberto Simões

Data: 19/01/2021

# Resumo

O Logo é uma linguagem de programação educativa, com origem em 1967. Pretende-se implementar um interpretador um interpretador básico, capaz de simular o maior número possível de comandos da linguagem original. Nesta linguagem o programador controla uma tartaruga, que se vai mexendo através do espaço, e desenhando linhas por onde passa. O programa será um ficheiro de texto com uma ou mais linhas, e deve implementar, pelo menos, os seguintes comandos:

* O comando **fd** ou **foward** – move a tartaruga n pixies em frente: **fd** 10;
* O comando **bk** ou **back** – move a tartaruga, para trás: **back** 20;
* O comando **lt** ou **left** roda a tartaruga, para a esquerda, n graus: **left** 90;
* O comando **rt** ou **right** roda a tartaruga, para a direita, n graus: **right** 180;
* O comando **setpos**, **setxy**, **setx**, **sety** permite definir uma posição para a qual a tartaruga se deve movimentar: **setpos** [100 100], ou **setxy** 100 100 ou um eixo apenas, mantendo o outro eixo: **setx** -100;
* O comando gome move a tartaruga para o ponto inicial (0,0), e roda-a para a orientação original: **home**;
* Os comandos **pendown** e **pwnup**, respetivamente abreviados por **pd** e **pu**, permitem alternar entre o modo de desenho e o modo de movimentação livre;
* O comando **setpencolor** permite alterar a cor das linhas para os comandos que se seguem: **setpencolor** [99 0 0];
* O comando **make** permite definir o valor de uma variável: **make“varname**20;
* O comando **if** e **ifelse** permitem definir estruturas condicionais;
* O comando **repeat** repete um conjunto de comandos;
* O comando **while** permite definir ciclos;
* O comando **to** permite criar funções.

# Main

O ficheiro main.py recebe da consola o nome de um ficheiro, verifica se ele é válido e usando as funções do ficheiro **svg** lê-lo, executa o **parser**, o **lexer** e por fim realiza a função **drawAll**.

# Svg

Uma imagem com texto

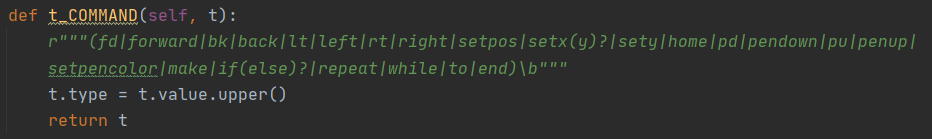
Descrição gerada automaticamenteO ficheiro **svg.py** é o *canvas* do programa. Neste ficheiro constam as funções que leem, abrem e escrevem em ficheiros e assim produzem o resultado final.

A função **drawLine** recebe um ponto origem, um ponto destino e uma cor e irá guardar a nova linha em HTML guardada em memória na variável **text**. Esta verifica também se a nova linha se encontra dentro dos limites de desenho, caso não se verifique emite uma mensagem de erro e para o programa.

A função **drawAll** irá escrever no ficheiro de resultado todo o texto HTML que se encontra guardado em memória e finalizar o programa.

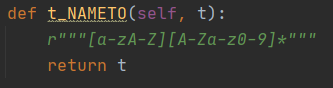
# Expressões Regulares

Esta foi a expressão regular usada de forma a ler apenas os comandos corretos:



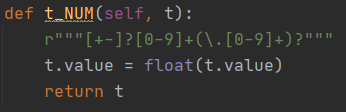
* Nesta expressão são lidos apenas os comandos pretendidos e suas abreviaturas, como o caso do **foward** ou **fd**.

Esta foi a expressão regular usada de forma a ler apenas os nomes das funções:



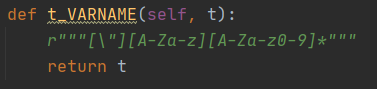
* Nesta expressão são lidos apenas os nomes das funções.

Esta foi a expressão regular usada de forma a ler apenas os números:



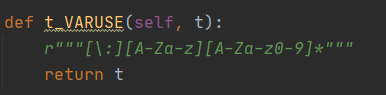
* Nesta expressão regular é lido qualquer número positivo ou negativo e com ou sem vírgulas, isto é -3 ou -3,14.

Esta foi a expressão regular usada de forma a ler apenas os nomes e os respetivos valores das varáveis:



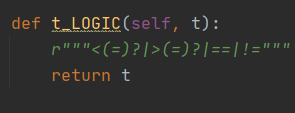
* Nesta expressão regular são lidas as variáveis que contenham o sinal **“**antes do seu nome e o respetivo valor associado, como por exemplo “numero10.

Esta foi a expressão regular usada de forma a ler apenas os nomes das varáveis que já possuem valores associados:



* Nesta expressão regular são lidas as variáveis que já possuam valores ou seja variáveis com “**:”** antes do nome da variável, um exemplo prático seria se já existir uma variável (“numero 10) e no código possuir “:numero” o valor associado a essa variável (10) irá ser usado.

Esta foi a expressão regular usada de forma a ler operadores lógicos:

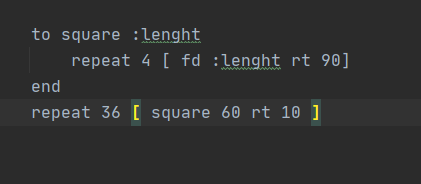


* Nesta expressão regular são lidos operadores lógicos que possam ser usados como comparadores (==, >, <, <=, >=, !=).

# Testes

À medida que o projeto estava a ser trabalhado era necessário fazer testes de forma a perceber se o **Lexer** estava a reconhecer as palavras no seu devido sítio assim sendo foi feito um teste a título de exemplo.

O Input foi o seguinte:



Com base no Input o Output foi:

Uma imagem com texto, placa

Descrição gerada automaticamente

# Gramática

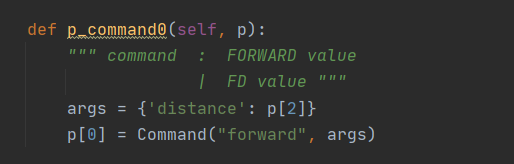
Uma imagem com texto

Descrição gerada automaticamenteNa classe Parser são mantidas em memória as variáveis, funções e outras definições escolhidas pelo utilizador que são usadas para obter o resultado final.

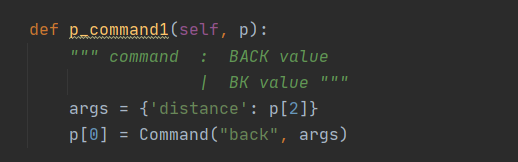
Uma imagem com texto

Descrição gerada automaticamenteA função Parse que é executada no main recebe todo o conteúdo inserido pelo utilizador e também todos os **tokens** reconhecidos pelo **Lexer**, depois verifica os comandos e por fim executa-os com a função **exec**.

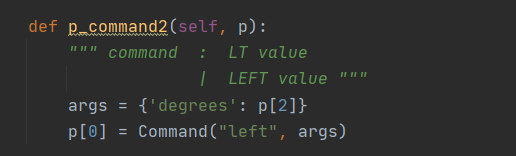
No Parser.py tem um conjunto de analisadores sintáticos de cada comando.



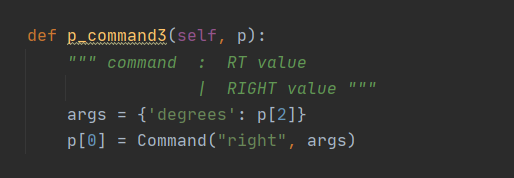
* Na função p\_command0, o comando **forward** ou **fd** é precedido de um valor, esse valor corresponde à distância que irá andar para a frente.



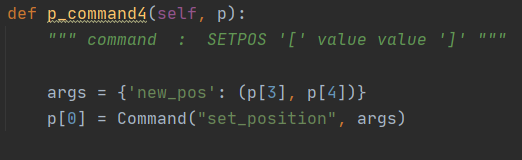
* Na função p\_command1, comando **back** ou **bk** é precedido de um valor, esse valor corresponde à distância que irá andar para trás.

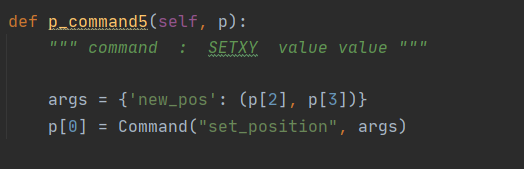


* Na função p\_command2, comando **left** ou **lt** é precedido de um valor, esse valor corresponde aos graus que pretende virar no sentido anti-horário.

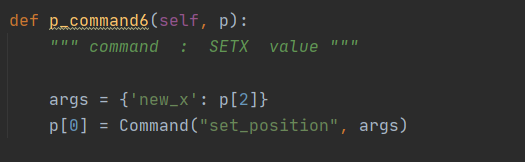


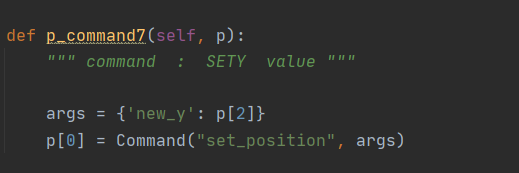
* Na função p\_command3, o comando **right** ou **rt** é precedido de um valor, esse valor corresponde aos graus que pretende virar no sentido horário.



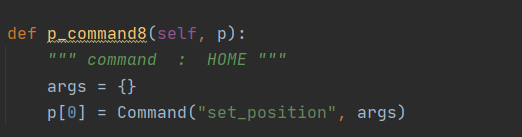


* Na função p\_command4 e p\_command5, o comando **setpos** e **setxy** são precedidos de dois valores entre parênteses ([10 10]) e sem eles (10 10) respetivamente. Estes valores têm como função colocar o objeto nos pontos que se pretende.

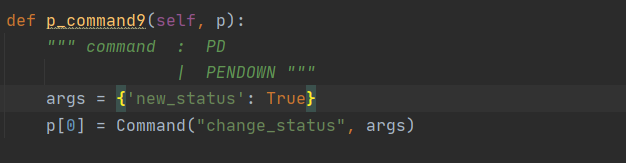


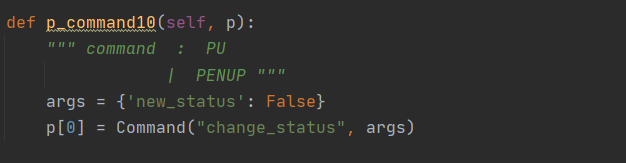


* Na função p\_command6 e p\_command7, os comandos **setx** ou **sety** são precedidos de um valor, esse valor corresponde a posição x que se pretende alterar no caso do comando **setx** e a posição y no caso do comando **sety**.

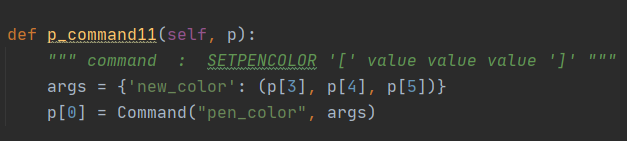


* Na função p\_command8, o comando **home** tem como função voltar a posição inicial do desenho.

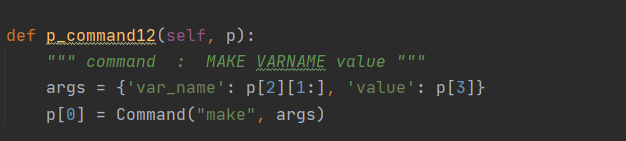




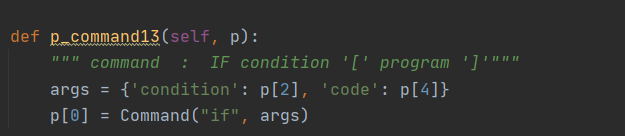
* Na função p\_command9, o comando **pendown** ou **pd** tem a função de desenhar as instruções seguintes de forma a que se consiga ver o desenho feito.
* No caso da p\_command10, o comando **penup** ou **pu** tem a função de desenhar as instruções seguintes de forma a que não se consiga ver o desenho feito.



* Na função p\_command11, o comando **setpencolor** é precedido de três valores entre parênteses [100 0 100], esses valores correspondem a uma cor que se pretenda ver no desenho, assim sendo até caso contrário as instruções seguintes serão desenhadas com essa mesma cor escolhida.

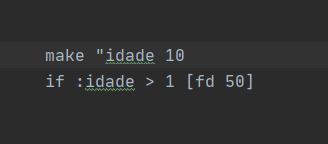


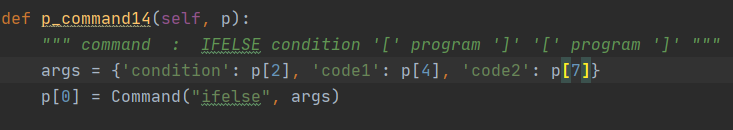
* Na função p\_command12, o comando **make** é precedido de uma variável (varname) e de um valor, esse valor corresponde ao valor da variável que poderá ser usada em outras estruturas com por exemplo as estruturas condicionais, entre outras.



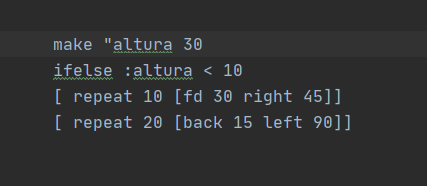
* Na função p\_command13, o comando **if** é precedido de uma condição, e entre parênteses o código que se pretenda ser executado.

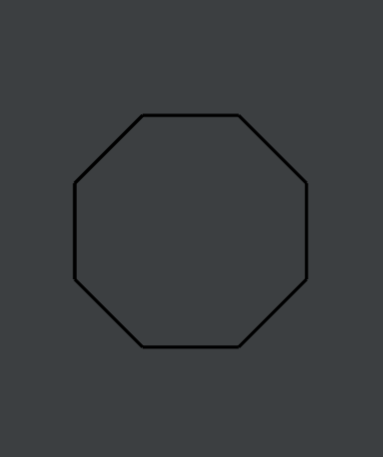


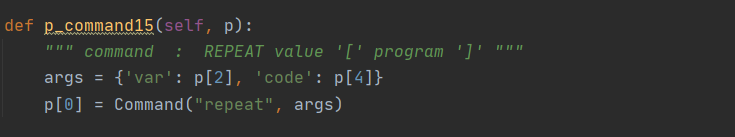




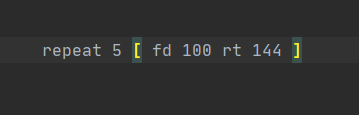
* Na função p\_command14, o comando **ifelse** é precedido de uma condição, e entre os primeiros parênteses o código que se pretenda ser executado no caso de essa condição seja verdadeira e os segundos parênteses o código a executar caso a condição seja falsa.



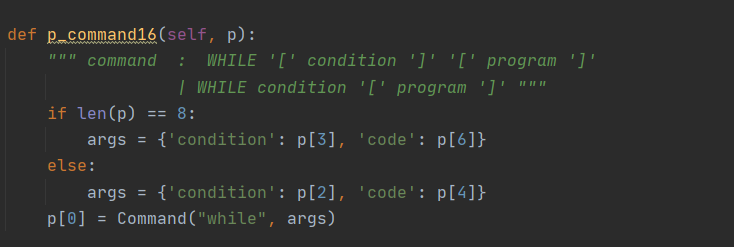




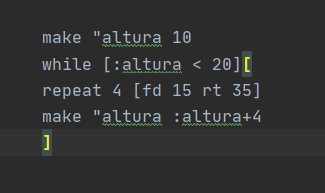
* Na função p\_command15, o comando **repeat** é precedido de um valor, que representa o número de repetições que devem ser feitas ao código que esta entre parênteses.



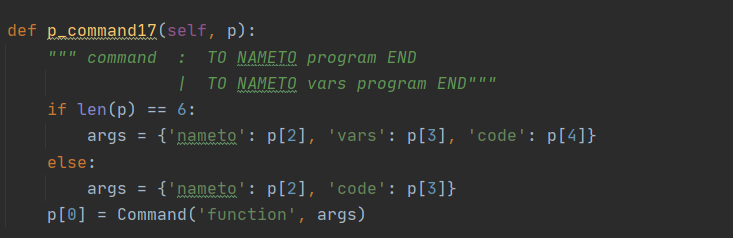




* Na função p\_command16, o comando **while** tem duas variantes, no caso da primeira a condição deve estar entre parênteses assim como o código a executar. Na segunda variante a condição não necessita de estar entre parênteses, no entanto o código a executar sim. Em ambas as situações o código só executa caso a condição seja verdadeira.

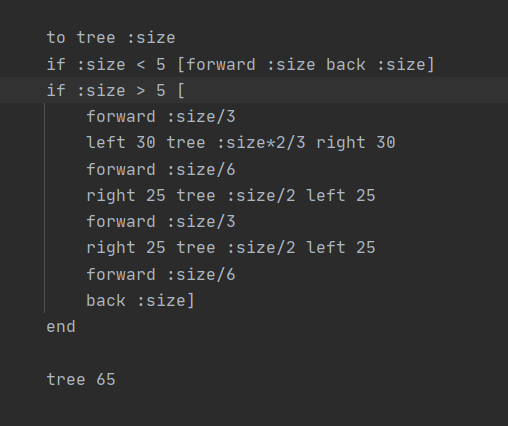






* Na função p\_command17, o comando **to** tem duas variantes, no caso da primeira possui o nome da função (nameto), o conjunto de código a ser executado (program) e o terminador de uma função (end). Na segunda possui o nome da função (nameto), os argumentos (vars) que a função necessita para ser executada, o conjunto de código a ser executado (program) e o terminador de uma função (end).

´



# Comandos

Foi criada uma classe Command.py, a função desta classe é armazenar os comandos e os seus respetivos argumentos (ex: **forward** 10; command (“**forward**”, 10)).

Foi criado um dicionário que relaciona o comando a uma função, como é o caso do “ **forward**: do\_forward ”, isto é o sempre que for chamado o comando **forward** ele irá fazer um conjunto de instruções da função “do\_forward” e de forma igual com os restantes comandos.



Na class Command consta uma tabela de nome dispatch\_table que procura pela função que deve ser executada consoante o nome recebido da classe Parser.

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

Nos exemplos acima estão duas funções, na primeira função o comando **repeat** é executado. Na segunda calcula a nova posição do desenho, caso esteja em modo desenho irá guardar o desenho na classe **svg** e no final atualiza a posição atual na classe **Parser**.

# Conclusão

O projeto foi bastante interessante e sobretudo exigente, o seu desenvolvimento potenciou conhecimento que nos será útil em projetos futuros. Houve algumas dificuldades na sua estruturação, no entanto foram anuladas com o decorrer do tempo.

Em suma, abordamos todos os tópicos pretendidos e conseguimos cumprir todos os objetivos propostos.