Universidade do Minho

Departamento de Informática

Processamento de Linguagens (3º ano de LEI)

Trabalho Prático 2

Tradutor PLY-simple para PLY

Relatório de Desenvolvimento

Inês Vicente (A93269) | Jorge Melo (A93308) | Miguel Martins (A93280)

22 de maio de 2022

Resumo

O seguinte relatório serve como um guia do processo de conceção da ferramenta **PLY Simple**. Nele podemos encontrar, não só a contextualização do problema que o grupo teve em mãos e a solução que o mesmo concebeu, mas também uma reflexão sobre esta última, dando especial ênfase à forma como os problemas encontrados foram resolvidos, à arquitetura da solução e à forma como o grupo decidiu introduzir alguma criatividade num projeto que, mesmo assim, não poderia deixar de ser maioritariamente técnico. Além disso, também serão exemplificadas diferentes formas de utilizar a ferramenta.

Conteúdo

Capítulo 1

Introdução

Para este segundo trabalho prático da disciplina, o grupo optou por conceber uma solução para o segundo problema apresentado no enunciado: criação de um *Tradutor PLY-simple para PLY*. Em suma, foi desenvolvido uma versão mais simples de PLY. Sendo assim, o programa recebe como *input* código *ply-simple* (definido por nós), passa-o para *ply* e compila como se fosse *ply*.

Capítulo 2

Metodologia e Documentação

Tal como referido acima, o objetivo deste projeto é criar uma versão mais simplificada de *Ply*. Para isso decidimos analisar vários aspetos do *Ply* e ver de que maneira era possível tornar cada uma delas mais intuitiva. O nosso objetivo principal era tornar a escrita de gramáticas o mais próxima da escrita usual possível, evitando a criação de várias funções para a mesma regra e tentando usar uma sintaxe e identação próximas do que se usa normalmente.
Abaixo encontra-se a documentação da nossa linguagem *ply-simple* de maneira a demonstrar as diferentes características da nossa linguagem.

Capítulo 3

States

3.1 text

Há partes do *ply-simple* em que o código é exatamente igual a como seria o código em *ply*. Sendo assim, só é necessário interpretar essa parte do código como texto, sem as regras que haviam nas outras partes. Para isso, é criado o estado *text*. O único aspeto desta parte do código a que é necessário ter em atenção são os *tabs*. Sendo assim, *TAB* é um *token* deste estado.

3.2 prec

Ao tratar as precedências, é necessário haver um *token* especial, *LR*. Por esse motivo, foi criado este estado.

3.3 gram

Ao interpretar a gramática, também é preciso criar um estado, devido à existência de muitos *tokens* específicos deste contexto, que não interessa serem detetados noutros estados.

3.4 gramT

Dentro da gramática, é necessário haver outro estado, para interpretar o texto da gramática, que tem algumas especificações diferentes.

Capítulo 4

Variáveis no código

4.1 Variáveis globais

- **halits -** indica se há *literals*. É necessário para, aquando da construção do *lex* em *ply*, sabermos se é necessário dar *import* dos *literals*.
- **buf -** armazena a secção da gramática que será colocadas numa certa função do código *ply*.

4.2 Variáveis do parser

- **ltok -** lista de *tokens*. É necessário contruí-la porque ela não é declarada em *ply-simple*, é induzida através dos *tokens* declarados individualmente.
- **isCode -** indica se estamos na secção *code*. Se estivermos, é necessário retirar *tabs* do fim, para se detetar o fim do cógido propriamente.
- **hasReserved -** indica se há *reserved words*. Isso é útil porque, se houver, é necessário junta-las à lista de *tokens*.
- **inYacc -** indica se estamos no *yacc*. É necessário porque o *p_error* é definido quase da mesma forma, sendo apenas o argumento diferente. Esta variável torna possível construir ambos os *p_error* na mesma função, denotando apenas a tal pequena diferença.

Capítulo 5

Website (EXTRA)

Uma vez que o grupo considerava que o projeto *per se* era bastante simples, pareceu-nos uma boa ideia implementar um extra de forma a aumentar a riqueza do produto final desenvolvido. Para que tal fosse conseguido, decidimos conceber um *website* simples, no qual seria possível:

- Executar o PLY-Simple;
- Consultar a Documentação da ferramenta.

Em suma, o *website* concebido foi tal que fizesse lembrar facilmente o tipo de *websites* comumente desenvolvidos para uma linguagem de progamação. Tal como os *websites* de *Python*, *Golang* ou *Rust*, para citar alguns, também o nosso possui como cerne a documentação e o compilador, como fica explicíto nas figuras seguintes e como pode ser testado no próprio *website*: https://ply-simple.herokuapp.com.

5.1 Funcionamento do *Website*

Como foi dito, o *website* é bastante simples. Nele é possível consultar a documentação da aplicação, bem como usar o compilador *online*.

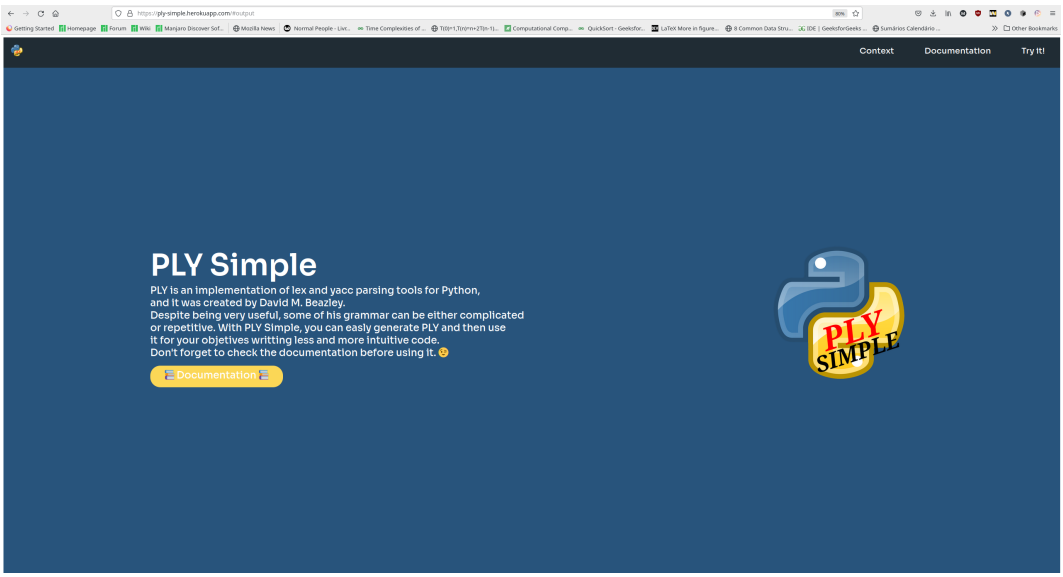

Figura 5.1: Página Inicial

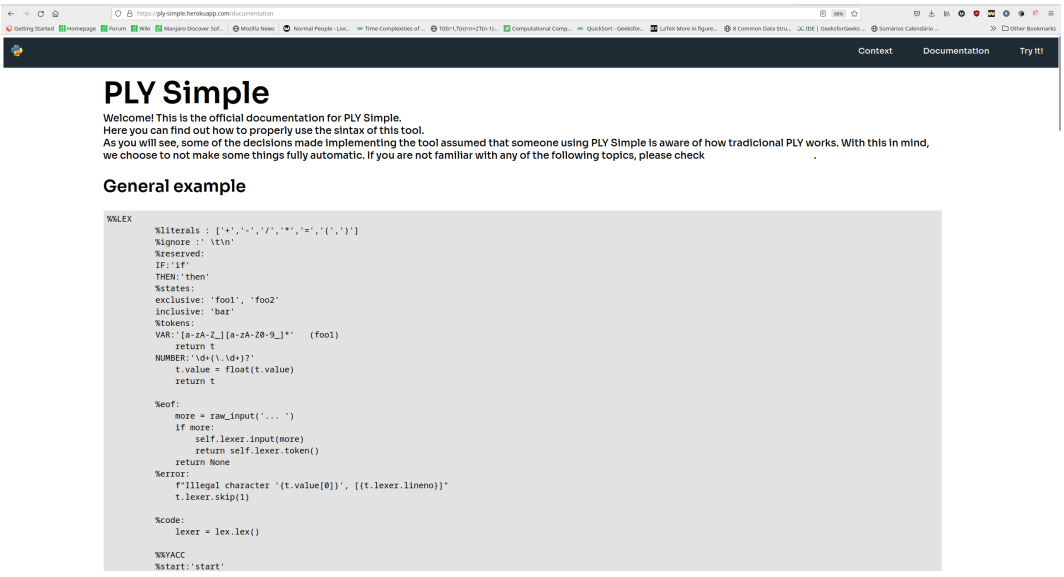

Figura 5.2: Página da Documentação

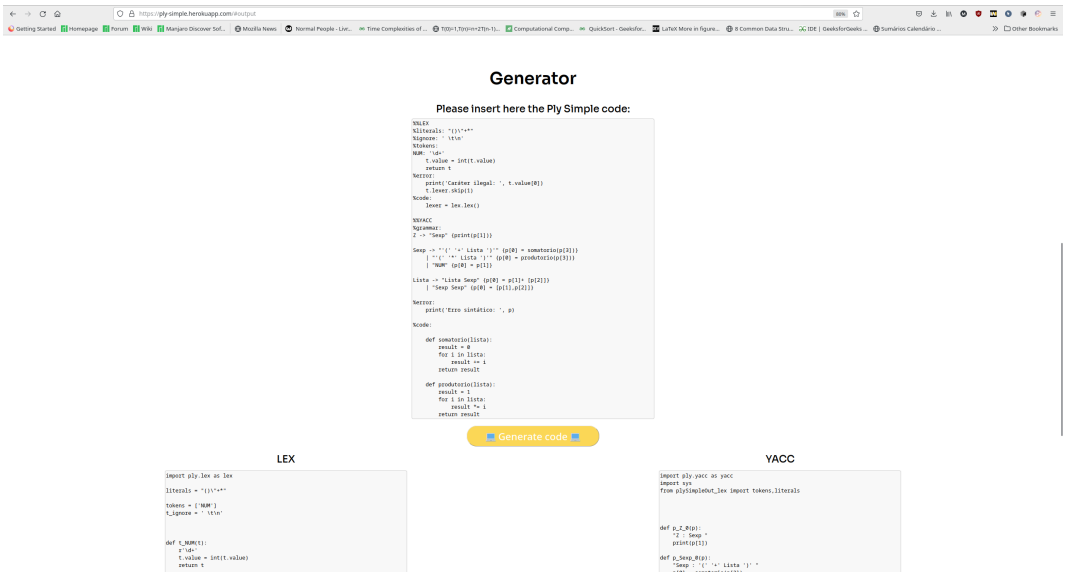

Figura 5.3: Utilização do Compilador

5.2 Arquitetura do *Website*

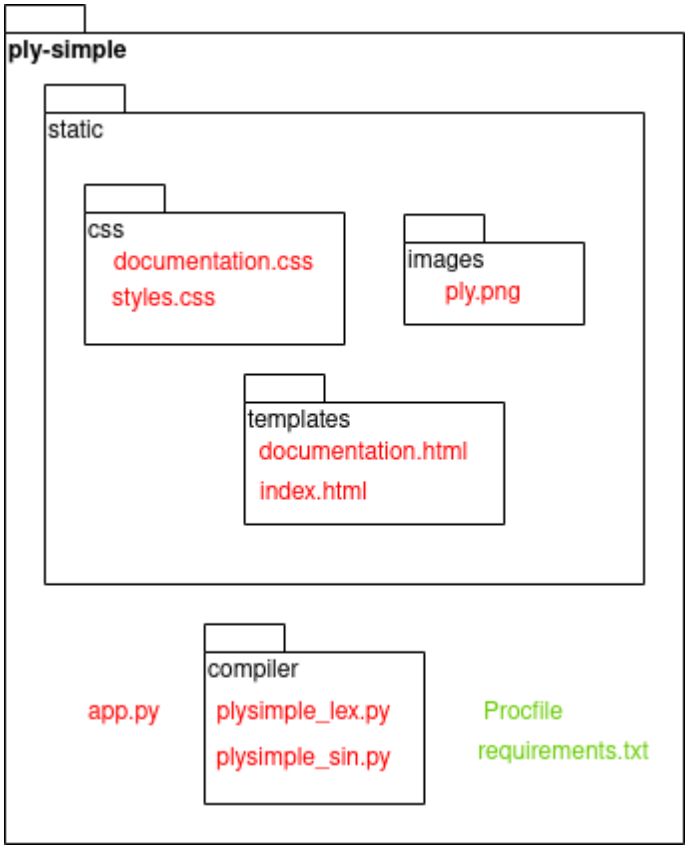

Figura 5.4: Arquitetura do *Website*

O *Website* teria de ser capaz de executar facilmente código *Python*, que foi a linguagem utilizada para conceber a ferramenta descrita nos capítulos anteriores, pelo que o grupo optou por utilizar uma *framework web* escrita em *Python*. As 2 opções mais conhecidas eram *Flask* e *Django* e a primeira acabou por ser a escolhida, por ser mais simples e os requisitos do *website* não exigirem nenhuma funcionalidade que a mesma não fosse capaz de fornecer.

Deste modo, a arquitetura acabou por seguir o que é tipicamente exigido numa aplicação *Flask*:

- Uma diretoria *static* temos os elementos correspondentes ao *Frontend*, sendo que é obrigatório que os ficheiros *HTML* estejam na diretoria *templates*. (O grupo não achou necessário usar uma *framework* como *React.js* ou *Vue.js* para desenvolver interface de utilizador, tendo em conta a simplicidade da ferramenta);
- Um ficheiro *Python app.py* onde está declarada toda a logistica *Backend* da aplicação, tirando partido das funcionalidades do *Flask*.

Além disso, também seriam necessário:

- Uma diretoria *compiler* onde estavam os ficheiros *Python* necessários para executar a ferramenta PLY Simple *per se*;
- 2 ficheiros essenciais para correr a aplicação na nuvem, através da ferramenta *Heroku*. São esses ficheiros:
  - *Procfile*, para especificar que comandos a aplicação executa na fase de arranque;
  - *requirements.txt*, para especificar as bibliotecas de *Python* que a aplicação obrigatoriamente necessita para executar corretamente.

Capítulo 6

Testes

Para melhor demonstrar o nosso projeto, decidimos mostrar um exemplo prático da nossa linguagem e o seu respetivo *output*.

```
%%LEX
%literals : "+-/*=()"
%ignore   :   ' \t\n'
IF  :   'if'
%tokens:

VAR   :  '[a-zA-Z_][a-zA-Z0-9_]*'
    return t

NUMBER  :  '\d+(\.\d+)?'
    t.value = float(t.value)
    return t

%error:
    f"Illegal character '{t.value[0]}', [{t.lexer.lineno}]"
    t.lexer.skip(1)

%code:
    lexer = lex.lex()
%%YACC

%prec:
'+','-' : left
'*','/' : left
'UMINUS' : right

%grammar:
Stat -> "VAR '=' Exp" {ts[p[1]] = p[3]}
    | "Exp" {print(p[1])}

Exp -> "Exp '+' Exp" {p[0] = p[1] + p[3]}
    | "Exp '-' Exp" {p[0] = p[1] - p[3]}
    | "Exp '*' Exp" {p[0] = p[1] * p[3]}
    | "Exp '/' Exp" {p[0] = p[1] / p[3]}
    | "'-' Exp %prec UMINUS" {p[0] = -p[2]}
    | "'(' Exp ')'" {p[0] = p[2]}
    | "NUMBER" {p[0] = p[1]}
    | "VAR" {p[0] = getval(p[1])}

%error:
    (print(f"Syntax error at '{p.value}', [{p.lexer.lineno}]"))

%code:
    ts = {}
    def getval(n):
        if n not in ts: print(f"Undefined name ‘{n}’")
        return ts.get(n,0)

    y=yacc.yacc()
    y.parse("3+4*7")
```

Figura 6.1: Programa em Ply Simple

Em cima está apresentado um programa em *ply simple*. Este programa é o mesmo que estava no enunciado (calculadora), mas aplicado à nossa linguagem.

```
import ply.lex as lex

literals = "+-/*=()"

tokens = ['NUMBER', 'VAR']
t_ignore = ' \t\n'
t_ignore_IF='if'



def t_VAR(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    return t

def t_NUMBER(t):
    r'\d+(\.\d+)?'
    t.value = float(t.value)
    return t







def t_error(t):
    f"Illegal character '{t.value[0]}', [{t.lexer.lineno}]"
    t.lexer.skip(1)

lexer = lex.lex()

```

Figura 6.2: Output do lexer do programa

```python
from plySimpleOut_lex import tokens,literals

precedence = (
    ('left', '+', '-'),
    ('left', '*', '/'),
    ('right', 'UMINUS'),
)

def p_Stat_0(p):
    "Stat : VAR '=' Exp "
    ts[p[1]] = p[3]
def p_Stat_1(p):
    "Stat : Exp "
    print(p[1])

def p_Exp_0(p):
    "Exp : Exp '+' Exp "
    p[0] = p[1] + p[3]
def p_Exp_1(p):
    "Exp : Exp '-' Exp "
    p[0] = p[1] - p[3]
def p_Exp_2(p):
    "Exp : Exp '*' Exp "
    p[0] = p[1] * p[3]
def p_Exp_3(p):
    "Exp : Exp '/' Exp "
    p[0] = p[1] / p[3]
def p_Exp_4(p):
    "Exp : '-' Exp %prec UMINUS "
    p[0] = -p[2]
def p_Exp_5(p):
    "Exp : '(' Exp ')' "
    p[0] = p[2]
def p_Exp_6(p):
    "Exp : NUMBER "
    p[0] = p[1]
def p_Exp_7(p):
    "Exp : VAR "
    p[0] = getval(p[1])


def p_error(p):
    (print(f"Syntax error at '{p.value}', [{p.lexer.lineno}]"))

ts = {}
def getval(n):
    if n not in ts: print(f"Undefined name '{n}'")
    return ts.get(n,0)
y=yacc.yacc()
y.parse("3+4*7")
```

Figura 6.3: Output do parser do programa

Podemos observar então o output do programa dividido em dois ficheiros: o lexer e o parser. Este programa pode agora ser executado para gerar o seguinte output:

```
[jorge@localhost TP2_PL]$ python -u "/home/jorge/Uni/3ano/2sem/PL/TP/TP2_PL/TP2_PL/plySimpleOut_sin.py"
Generating LALR tables
31.0
```

Figura 6.4: Output do programa Ply

Capítulo 7

Conclusão

Finalizado o trabalho, achamos que conseguimos concretizar os objetivos que nos foram propostos. Este projeto ajudou-nos a consolidar todo o processo de criação de gramáticas. Achamos que no final conseguimos obter uma linguagem que era consideravelmente mais simples e intuitiva que o *Ply*, diminuindo o número de funções que eram necessárias criar e tornando a linguagem mais natural. Contudo, consideramos que o nosso trabalho apresenta alguns problemas como a falta de identificação de erros, algumas restrições que são introduzidas que podem dificultar um pouco a utilização desta linguagem e o facto de não ser possível colocar comentários.