Processamento de Linguagens (3 ano de LEI)

Trabalho Prático 2.5

Relatório de Desenvolvimento

Grupo 2

Cláudio Bessa (a97063)

José Martins (a97903)

28 de maio de 2023

Resumo

Este relatório aborda o desenvolvimento de um conversor usando módulos de gramáticas tradutoras de *Python*, no contexto do trabalho prático da unidade curricular de Processamento de Linguagens.

Conteúdo

1 Introdução **2**
1.1 Conversor de Pug para HTML 2

2 Análise e Especificação **4**
2.1 Descrição informal do problema 4
2.2 Especificação do Requisitos 4
2.2.1 Dados 4
2.2.2 Pedidos 4

3 Concepção/Desenho da Resolução **5**
3.1 Estruturas de Dados 5
3.2 Algoritmos 6
3.2.1 *Tokens* 6
3.2.2 Linguagem da Gramática 6
3.2.3 *Parser Bottom-Up* 7

4 Codificação **8**
4.1 Alternativas, Decisões e Problemas de Implementação 8
4.2 Testes realizados e Resultados 9

5 Conclusão **10**

A Código do Programa **11**
A.1 Main 11
A.2 Lex 11
A.3 Yacc 14
A.4 Parser_r 15
A.5 Exemplos de Execução 16
A.5.1 Exemplo 1 16
A.5.2 Exemplo 2 16
A.5.3 Exemplo 3 17
A.5.4 Exemplo 4 18
A.5.5 Exemplo 5 19

Capítulo 1

Introdução

De forma a proporcionar uma experiência correta e positiva sobre a criação ou desenvolvimento de novos projetos os compiladores, interpretador e conversores tomam um papel crucial através da sua capacidade de realizar a análise sintática de texto. Isto decompondo um conjunto de dados de entrada, e.g. código escrito, em unidades estruturais, a partir de uma gramática tradutora. Este método, conhecido por *parsing*, é um método que existe *a posteriori* de uma análise léxica, na qual uma sequência de caracteres é convertida numa sequência de *tokens*.

1.1 Conversor de Pug para HTML

Área: Processamento de Linguagens

Enquadramento Neste projeto iremos, com o auxílio da linguagem de programação *python* e do módulo *ply*, desenvolver um programa capaz de fazer a análise léxica e sintática de uma linguagem de programação titulada de *Pug*. Mais especificamente, o nosso programa irá converter o código fornecedio pelo utilizador em *Pug* para um código correspondente em *HTML*.

Contexto A necessidade de um conversor de *Pug* para *HTML* surgiu devido à crescente popularidade do *Pug* como uma alternativa elegante e amigável para escrever código *HTML*. Embora o mesmo seja amplamente utilizado por desenvolvedores, a conversão manual de *Pug* para *HTML* pode ser demorada e propensa a erros. Portanto, este projeto procura resolver esses problemas, pelo menos facilitar na resolução dos mesmos, oferecendo uma ferramenta que simplifica a conversão e aumenta a produtividade dos desenvolvedores.

Problema O problema que procuramos resolver é a conversão manual e trabalhosa dos respetivos arquivos. Nosso objetivo é desenvolver um conversor automatizado que processe arquivos *Pug* e gere o código *HTML* correspondente, garantindo a precisão e eficiência do processo. O projeto visa simplificar a vida dos desenvolvedores, economizando tempo e esforço ao realizar a conversão de forma rápida e confiável.

Objetivo Este relatório tem como objetivo documentar o processo de desenvolvimento do conversor, apresentando as etapas chave, as decisões de projeto, as tecnologias utilizadas e os desafios enfrentados. Além disso, destacaremos os resultados alcançados e os benefícios do conversor, fornecendo uma visão geral abrangente do projeto.

Resultados ou Contributos Os principais pontos a serem evidenciados neste projeto são:

- Desenvolvimento de um conversor de *Pug* para *HTML* totalmente funcional e eficiente.
- Redução significativa do tempo necessário para converter arquivos *Pug* em *HTML*.
- Melhoria da produtividade dos desenvolvedores, eliminando a necessidade de conversão manual.
- Manutenção da precisão e fidelidade do código *HTML* gerado a partir dos arquivos *Pug*.
- Flexibilidade para lidar com recursos avançados do *Pug*, como *mixins*, variáveis e *loops*.

Estrutura do documento O documento está estruturado da seguinte forma:

1. Introdução: Apresentação do projeto, sua importância e o objetivo do relatório.
2. Análise e Especificação: Explicação sobre *Pug*, suas características e vantagens em relação ao *HTML* convencional.
3. Concepção/Desenho da Resolução: Descrição da abordagem e das ferramentas utilizadas no desenvolvimento do conversor.
4. Codificação e Testes: Detalhamento das etapas de implementação, destacando as principais decisões de projeto e os desafios enfrentados, assim como os testes.
5. Conclusão: Termina-se o relatório com uma síntese do que foi dito, as conclusões e trabalho futuro.
6. Apêndices: Conjunto de documentos, textos, informações cruciais a serem partilhadas após, ou não, terem sido utilizados como referência ao longo deste documento

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Para o enunciado 2.5 pretende-se trabalhar com um determinado arquivo Pug de forma a efetuar uma conversão para um arquivo HTML. São apresentadas então várias alíneas que expõem os problemas a resolver pelo grupo.

2.2 Especificação do Requisitos

2.2.1 Dados

Os dados a utilizar para este projeto encontram-se na diretoria *tests/pug/* dentro de cada ficheiro lá existente, tendo eles extensão *.pug*

2.2.2 Pedidos

É nos solicitado para processar ficheiros com extensão *.pug* de modo a:

- Identificar todos os elementos característicos de um ficheiro da linguagem Pug, podendo ser eles, *tags*, atributos, texto corrente, indetanção, entre outros.
- Identificar blocos de código de *javascript* e *css*
- Identificar elementos únicos de Pug utilizados para otimizar a escrita de ficheiros HTML, como *each loops*, *mixins*, *includes*, entre outros.
- Gerar uma página HTML com a respetiva conversão de cada token de Pug para HTML

Capítulo 3

Concepção/Desenho da Resolução

O código do nosso projeto está contido em quatro ficheiros localizados dentro da diretoria *src*. O principal (*main.py*) contém código relativo à inicialização da conversão do ficheiro de *input* em um ficheiro de *output*, de forma a efetuarmos uma espécie de modularidade na estruturação do nosso projeto. Seguidamente, o ficheiro (*lex_pug.py*) contém as funções relativas ao processo de tokenização do ficheiro Pug. Por fim obtemos o ficheiro (*yacc_pug.py*) relativo à identificação e definição da gramática do ficheiro de *output* HTML. Devido à óbvia separação de módulos optamos por fazer esta divisão. Por fim temos o ficheiro *parser_r.py* que é encarregado de efetuar todas as diferenciações entre elementos da gramática.[1]

3.1 Estruturas de Dados

Usamos várias estruturas de dados auxiliares ao longo do processo de análise léxica de forma a identificar diferentes dados obtidos pela leitura e parsing do ficheiro *.pug*. Desta forma armazenados diferentes identificadores de *tokens* e também prevemos casos de utillização de determinadas *tags* de forma a ajudar no processo de identificação.

1. *"What is PUG Syntax?"*, https://www.educative.io/answers/what-is-pug-syntax, Accessed: 2023-04-27.

3.2 Algoritmos

3.2.1 *Tokens*

Os *tokens_originais*, identificam os que originalmente estavam presentes no analisador léxico, porém como é visível durante os diferentes testes e pela nossa implementação, ou até mesmo no analisador sintático, nem todos os *tokens* são utilizados, daí existir o outro tuplo *tokens* que se encarrega de identificar os utilizados e os que estão presentes na versão de entrega do nosso conversor.

```
tokens_originais = (
    'ATRIBUTE\_NAME',
    'ATRIBUTE\_VALUE',
    'ATRIBUTE\_VALUE\_MULTILINE',
    'IGNORE\_COMMENT',
    'COMMENTLINE',
    'COMMENT\_BLOCK',
    'DOCTYPE',
    'DOT',
    'COMMA',
    'EQUALS',
    'DEDENT',
    'INDENT',
    'LITERAL\_TAG',
    'LPAREN',
    'RPAREN',
    'TAG',
    'TAG\_SELF\_CLOSE',
    'VERTBARS',
    'DIV',
    'STRING
)

tokens = (
    'COMMENT_BLOCK',
    'DEDENT',
    'INDENT',
    'TAG',
    'STRING',
    'TAG_SELF_CLOSE
)

states = (
    ("atributte", "exclusive"),
    ("multiline", "exclusive")
)
```

3.2.2 Linguagem da Gramática

```
elemList : elemList elem
```

```
          | elem
          | empty

elem : TAG INDENT elemList DEDENT
     | TAG INDENT elemList
     | TAG
     | COMMENT_BLOCK
```

3.2.3 *Parser Bottom-Up*

Uma vez que utilizamos uma gramática livre, o *Parser* utilizado na gramática acaba por ser o *default* do *YACC*, uma vez que lê os tokens recorrendo a um algoritmo *bottom-up*. O algoritmo utilizado é o LALR(1). LALR(1), é um tipo de analisador sintático utilizado em compiladores e geradores de analisadores sintáticos para processar gramáticas livres de contexto. É conhecido por ser eficiente em termos de espaço e tempo. Ele é uma combinação do método LR(0) e do método SLR(1), e geralmente resulta em tabelas de análise menores e mais rápidas de serem construídas do que o LR(1) puro. A principal ideia por trás do LALR(1) é reduzir a quantidade de estados e transições na tabela de análise, aproveitando a ideia de que muitos estados do LR(0) e SLR(1) são semelhantes. Isso é feito por meio de agrupamento de estados, permitindo que várias produções de redução compartilhem o mesmo estado.[2]

2. *"Bottom-Up Parsing"*, https://suif.stanford.edu/dragonbook/lecture-notes/Stanford-CS143/08-Bottom-Up-Parsing.pdf, Accessed: 2023-05-27.

Capítulo 4

Codificação

4.1 Alternativas, Decisões e Problemas de Implementação

Surgiram nos algumas ideias da forma como puderiamos implementar o conversor, especialmente na parte de análise semântica enquanto definiamos a gramática. Poderiamos separar os *idents* e depois o restante dos elementos. Esses outros elementos poderiam levar *parse* em outro ficheiro, utilizando os meus métodos que estão a ser utilizados ou outros. Através desta ideia surgiu a nossa ideia, o que nos fez reestruturar a arquitetura utilizada até então.

Optamos por utilizar dois ficheiros como analisador sintático sendo que um deles identifica a gramática e as regras a utilizar são efetuadas em outras. Este novo analisador em vez de receber todos os elementos no sentido que podia ter uma manopla de 19 *tokens*/elementos acaba por considerar tudo como *tag*.[1] Sendo assim não existe *strings*, atributos,[2] iguais, parentesis, entre outros.[3] Contudo, mantivemos a estrutura inicial dos *tokens* de modo a demonstrar que efetuamos uma troca de decisão de forma a corrigir alguns erros.

Possuimos alguns problemas na análise léxica relativo a conseguir diferenciar *tokens* do tipo *string* ou uma *tag*. Isto convergia em erro uma vez que a identação induzia em erro. Devido a isso tivemos que implementar uma função denominada de *get_identation* que recebia como argumentos a posição do *ident* anterior como *pos* e recebia a *string* de *input* respetiva, como *input_str*.

Obtivemos mais problemas na parte relativos à análise sintática. Isto mais uma vez relativo aos problemas de identação, maioritariamento ao elemento da gramática que definimos como *DEDENT* que representava menos uma (-1) identação relativo à linha anterior. Para resolver isto tivemos de repensar a maneira como estavamos a escrever o código. Optamos então por simplificar e baseamo-nos num exemplo dado pelo professor durante a aula. Para isso criamos também um novo ficheiro que nos ajudou com funções auxiliares.

1. *"Tags"*, https://pugjs.org/language/tags.html, Accessed: 2023-05-27.
2. *"Attributes"*, https://pugjs.org/language/attributes.html, Accessed: 2023-05-10.
3. *"Plain Text"*, https://pugjs.org/language/plain-text.html, Accessed: 2023-04-30.

4.2 Testes realizados e Resultados

Foram efetuados alguns testes, isto utilizando os ficheiros fornecidos na diretoria de testes. Estes testes ocupam diferentes graus de dificuldade sendo titulados segundo os mesmos, por uma ordem crescente, i.e. o ficheiro com nome *data2.pug* é de um grau dificuldade inferior comparativamente ao *data3.pug*. O exemplo mais simples é o que está contido em *data1.pug* e posteriormente submetido em apêndice.

Através dos resultados conseguimos verificar que a conversão é feita corretamente, mas isto contém alguns se nãos. Caso o ficheiro de *pug* seja demasiado complexo, tal como conter *includes*, *mixins*, *each loops*, entre outros, os nosso analisador léxico não irá conseguir identificar a linha em questão devidamente. Há probabilidades, i.e. depende o caso em questão, de conseguir converter de forma correta para o HTML, isto se esse *token* partilhar semelhanças com outro da gramática que nós criamos.

Capítulo 5

Conclusão

Este projeto permitiu-nos aprofundar os conhecimentos adquiridos nas aulas lecionas de Processamento de Linguagens de uma forma prática e interessante. Passamos a ter uma ideia de como é que os compiladores funcionam, algo que até então apenas tinhamos uma ideia muito vaga. Nenhuma das gramáticas e *tokenização* se demonstra perfeita uma vez que falta algumas implementações, mas acordamos que a reestruturação que efetuamos no conversor acaba por corrigir erros mais cruciais, i.e. erros de identação, comparativamente a erros que agora existem, mencionados durante o relatório. Gostaríamos de ter implementado mais funcionalidades, i.e. mais componentes, que existem na linguagem *Pug* e/ou *HTML*, e.g. *each loops*, *mixins* entre outros.

Apêndice A

Código do Programa

A.1 Main

```
import sys
import ply.lex as lex
import ply.yacc as yacc

import lex_pug
import yacc_pug

if __name__== "__main__":

    if len(sys.argv) != 2:
        print("Usage: python main.py <input_file>")
        sys.exit(1)

    input_file = sys.argv[1]

    lexer = lex.lex(module=lex_pug)
    lexer.tabcount = 0
    lexer.lasttab = 0
    lexer.tag = False
    parser = yacc.yacc(module=yacc_pug)

    with open(f"../tests/pug/{input_file}.pug", 'r') as file:
        content = file.read()

    html = parser.parse(content, lexer=lexer)


    with open(f"../tests/html/output.html", "w+") as f:
        f.write(html)
```

A.2 Lex

```
import ply.lex as lex

```

```
indent_level = 0

def get_indentation(input_str, pos):
    string = input_str[pos:].partition('\n')[0]
    indent = 0
    for char in string:
        if char == ' ':
            indent += 1
        elif char == '\t':
            indent += 4
        else:
            break
    return int(indent / 4)

special_chars = {
    '&': "&amp;",
    '\"': "&quot;",
    '<': "&lt;",
    '>': "&gt;",
    '‘': "\""
}

self_closing = (
    'area',
    'base',
    'br',
    'col',
    'embed',
    'hr',
    'img',
    'input',
    'link',
    'meta',
    'param',
    'source',
    'track',
    'wbr'
)

doctypes = {
    "xml": "?xml version=\"1.0\" encoding=\"utf-8\" ?",
    "transitional": "!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\"",
    "strict": "!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\"",
    "frameset": "!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Frameset//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd\"",
    "1.1": "!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" \"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\"",
    "basic": "!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML Basic 1.1//EN\" \"http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd\"",
    "mobile": "!DOCTYPE html PUBLIC \"-//WAPFORUM//DTD XHTML Mobile 1.2//EN\" \"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd\"",
```

```
50      "plist": "!DOCTYPE plist PUBLIC \"-//Apple//DTD PLIST 1.0//EN\" \"http://www.
            apple.com/DTDs/PropertyList-1.0.dtd\""
51  }
52
53  # Define the token names
54  tokens = (
55      'COMMENT_BLOCK',
56      'DEDENT',
57      'INDENT',
58      'TAG',
59      'STRING'
60  )
61
62  states = (
63      ("atributte", "exclusive"),
64      ("multiline", "exclusive")
65  )
66
67  def t_COMMENT_BLOCK(t):
68      r'\/\/([^\|]([\n\s{4}]*))*\|'
69      t.lexer.comment_start_line = t.lexer.lineno
70      return t
71
72  def t_DOCTYPE(t):
73      r'doctype([^\n]+)'
74      t.value = t.lexer.lexmatch.group(2).strip()
75      if (t.value in doctypes):
76          t.value = doctypes[t.value]
77      else:
78          t.value = "!DOCTYPE " + t.value
79      return t
80
81  def t_LITERALTAG(t):
82      r'<[^<]+>'
83      return t
84
85  def t_TAG(t):
86      r'[^\n]+'
87      if '#' in t.value:
88          t.lexer.tag = True
89      else:
90          t.lexer.tag = False
91      if t.value in self_closing:
92          t.type = 'TAG_SELF_CLOSE'
93      if t.value[-1] == '.' and t.value.find("script") == -1:
94          t.lexer.push_state('multiline')
95      return t
96
97  def t_multiline_TAG(t) :
98      r'[^\n]+'
99      t.type = 'STRING'
100     if (t.value[-1] == '|'):
101         t.lexer.pop_state()
102     return t
```

```
103
104 def t_DOT(t):
105     r'\.(?![^\s]+)'
106     return t
107
108 def t_COMMA(t):
109     r',(?![^\s]+)'
110     return t
111
112 def t_IGNORECOMMENT(t):
113     r'\/\/(\-.*|
```

A.3 Yacc

```
1  import ply.yacc as yacc
2
3  from lex_pug import tokens
4  from parser_r import *
5
6  def p_document(p):
7      '''
8      document : elemList
9      '''
10     p[0] = p[1]
11
12 def p_string(p) :
13     '''
14     string : string STRING
15            | STRING
16     '''
17     if len(p) == 3 :
18
19         p[0] = p[1]+string(p[2])
20     else :
21         p[0] = string(p[1])
22
23 def p_elemList(p):
24     '''
25     elemList : elemList elem
26              | elem
27              | string
28              | empty
29     '''
30     if len(p) == 3:
31         if p[1] == []:
32             p[0] = p[2]
33         elif p[2] == []:
34             p[0] = p[1]
35         else:
36             p[0] = p[1] + p[2]
37     else:
38         p[0] = p[1]
```

```
39
40 def p_elem(p) :
41     '''
42     elem : TAG INDENT elemList DEDENT
43          | TAG INDENT elemList
44          | TAG
45          | COMMENT_BLOCK
46     '''
47     if len(p) == 2:
48         if p[1][0] == '/':
49             p[0] = comment(p[1])
50         else :
51             p[0] = open_tag(p[1]) + close_tag(p[1])
52     else:
53         p[0] = open_tag(p[1]) + ponto(p[3]) + close_tag(p[1])
54
55 def p_empty(p):
56     '''
57     empty :
58     '''
59     pass
60
61 def p_error(p):
62     if p is None:
63         print("Erro de sintaxe: token inesperado no final do arquivo")
64     else:
65         print(f"Erro de sintaxe: token inesperado '{p.value}' na linha {p.lineno}")
```

A.4 Parser_r

```
1 import re
2
3 def open_tag (str) :
4     padrao = r"^([a-z1-9]+)(\((.*)\))*\s*(.*)"
5     padrao2 = r"^#(\w+)(.*)"
6     padrao3 = r"^script\\((\\w+)(.*)\\).\"
7     match = re.match(padrao, str)
8     match2 = re.match(padrao2,str)
9     match3 = re.match(padrao3,str)
10     str_final = ""
11
12     if match3 != None :
13         str_final = f'<script {match3.group(1){match3.group(2)}>\\n'
14     elif match2 != None :
15         match_stripped = re.sub(r'^\.', '', match2.group(2))
16         str_final = f'<div class = "{match_stripped}" id = "{match2.group(1)}">\n'
17     else :
18         if match != None:
19             if match.group(3) is None:
20                 str_final = f'<{match.group(1)}> {match.group(4)}\\n'
21             else:
22                 str_final = f'<{match.group(1)} {match.group(3)}>{match.group(4)}\\n'
```

```
23      return str_final
24
25  def close_tag (str) :
26      padrao = r"^(\w+)(.*)"
27      padrao3 = r"^script\((\w+)(.*)\)."
28      match = re.match(padrao, str)
29      match3 = re.match(padrao3,str)
30      str_final = ""
31
32      if match3 != None :
33          pass
34      elif match != None :
35          str_final = f'</{match.group(1)}>\n'
36      else :
37          pass
38
39      return str_final
40
41  def comment(str):
42      str_final_regex = re.sub(r'^\/\/','',str)
43      str_final = f'<!-- {str_final_regex} -->\n'
44      return str_final
45
46  def string(str) :
47      return re.sub(r'\|',''',str)defponto(str) : returnre.sub(r'.|.'
48
49
50  ,'',str)
```

A.5 Exemplos de Execução

A.5.1 Exemplo 1

Pug:

```
1 html
2     title My Pug Page
```

Html:

```
1 <html>
2 <title> My Pug Page
3 </title>
4 </html>
```

A.5.2 Exemplo 2

Pug:

```
1 html
2     title My Pug Page
3     // Comment
4     p ola ola
5     // Enterro de gata?
6         Pao de batate
7         Batata |
8     p Coracao de galinha
```

Html:

```
1 <html>
2 <title> My Pug Page
3 </title>
4 <!--  Comment
5     p ola ola
6     // Enterro de gata?
7         Pao de batate
8         Batata | -->
9 <p> Coracao de galinha
10 </p>
11 </html>
```

A.5.3 Exemplo 3

Pug:

```
1 html
2     head
3         title My Pug Page
4     body
5         h1 Welcome to my Pug page!
6         ul
7             li Item 1
8             li Item 2
9             li Item 3
```

Html:

```
1 <html>
2 <head>
3 <title>  My Pug Page
4 </title>
5 </head>
6 <body>
7 <h1>  Welcome to my Pug page!
8 </h1>
9 <ul>
10 <li>  Item 1
```

```
</li>
<li>  Item 2
</li>
<li>  Item 3
</li>
</ul>
</body>
</html>
```

A.5.4 Exemplo 4

Pug

:

```
html(lang="en")
    p Cenas e coisas
        title UAUA
    u teste
    p.
        Pug is a terse and simple templating language with a
        strong focus on performance and powerful features|

    //
        comentarios
        cenas |

    p Depos
```

Html

:

```
<html lang="en">
<p> Cenas e coisas
<title> UAUA
</title>
</p>
<u> teste
</u>
<p>
Pug is a terse and simple templating language with astrong focus on performance and
    powerful features</p>
<!--
        comentarios
        cenas | -->
<p> Depos
</p>
</html>
```

A.5.5 Exemplo 5

Pug

:

```
1  html(lang="en")
2      head
3          title= pageTitle
4          script(type='text/javascript').
5              if (foo) bar(1 + 5)
6      body
7          h1 Pug - node template engine
8          #container.col
9              if youAreUsingPug
10                 p You are amazing
11             else
12                 p Get on it!
```

Html:

```
1  <html lang="en">
2  <head>
3  <title> = pageTitle
4  </title>
5  <script type='text/javascript'>
6  <if> (foo) bar(1 + 5)
7  </if>
8  <body>
9  <h1> Pug - node template engine
10 </h1>
11 <div class = "col" id = "container">
12 <if> youAreUsingPug
13 <p> You are amazing
14 </p>
15 </if>
16 <else>
17 <p> Get on it!
18 </p>
19 </else>
20 </body>
21 </head>
22 </html>
```

Bibliografia

"Attributes". https://pugjs.org/language/attributes.html. Accessed: 2023-05-10.

"Bottom-Up Parsing". https://suif.stanford.edu/dragonbook/lecture-notes/Stanford-CS143/08-Bottom-Up-Parsing.pdf. Accessed: 2023-05-27.

"Plain Text". https://pugjs.org/language/plain-text.html. Accessed: 2023-04-30.

"Tags". https://pugjs.org/language/tags.html. Accessed: 2023-05-27.

"What is PUG Syntax?". https://www.educative.io/answers/what-is-pug-syntax. Accessed: 2023-04-27.