

 Basic

Miss Ylva Llywelyn

2023/10/14



# CONTENTS

CH. 1: GRAMMER.....	1	CH. 2: CODE LISTING .....	2
---------------------	---	---------------------------	---



# CHAPTER 1: GRAMMER

This is a notation for writing down the grammar of the language. It uses regex syntax, with the components themselves being italicised.

<i>block</i>	<i>{ statement* }</i>
<i>statement</i>	<i>expression ;</i>
<i>expression</i>	
<i>const-expression</i>	"[a-zA-Z0-9 ]*" [+-]?[0-9]+(\.[0-9]+)? true false



## CHAPTER 2: CODE LISTING

```
1 #####
2 ### IMPORTS ###
3 #####
4
5 from __future__ import annotations
6 import sys, signal
7 from enum import Enum
8
9 #####
10 ### CONSTANTS ###
11 #####
12
13 DIGITS = '1234567890'
14
15 #####
16 ### ERRORS ###
17 #####
18
19 class ShorkError(Exception):
20     def __init__(self, startPosition:Position, endPosition:Position, errorName:str, details:
21         self.startPosition = startPosition
22         self.endPosition = endPosition
23         self.errorName = errorName
24         self.details = details
25
26     def __repr__(self) -> str:
27         return f""{self.errorName}: {self.details}
28 File: {self.startPosition.filename}, Line {self.startPosition.line}""
29
30 class IllegalCharacterError(ShorkError):
31     def __init__(self, startPosition:Position, endPosition:Position, details: str) -> None:
32         super().__init__(startPosition, endPosition, "Illegal Character", details)
33
34 #####
35 ### POSITION ###
36 #####
37
38 class Position:
39     def __init__(self, index:int, line:int, column:int, filename:str, filetext:str) -> None:
40         self.index = index
41         self.line = line
42         self.column = column
43         self.filename = filename
44         self.filetext = filetext
45
46     def Advance(self, currentChar) -> Position:
47         self.index += 1
48         self.column += 1
49
50         if currentChar == '\n':
51             self.line += 1
52             self.column = 0
53
54         return self
55
56     def Copy(self) -> Position:
57         return Position(self.index, self.line, self.column, self.filename, self.filetext)
```



```

58
59 #####
60 ### TOKENS ###
61 #####
62
63 class TokenType(Enum):
64     INT          = 1
65     FLOAT        = 2
66
67     PLUS         = 4
68     MINUS        = 8
69     MULTIPLY     = 16
70     DIVIDE       = 32
71
72     LPAREN       = 64
73     RPAREN       = 128
74
75     EOF          = 256
76
77 class Token:
78     def __init__(self, tokenType:TokenType, value:any = None) -> None:
79         self.tokenType = tokenType
80         self.value = value
81
82     def __repr__(self) -> str:
83         if self.value: return f'{self.tokenType}:{self.value}'
84         else: return f'{self.tokenType}'
85
86 #####
87 ### LEXER ###
88 #####
89
90 class Lexer:
91     def __init__(self, text: str, filename:str) -> None:
92         self.text:str = text
93         self.position:Position = Position(-1, 0, -1, filename, text)
94         self.currentChar:str = None
95         self.Advance()
96
97     def Advance(self) -> None:
98         self.position.Advance(self.currentChar)
99         self.currentChar = self.text[self.position.index] if self.position.index < len(self
100
101     def MakeTokens(self) -> list[Token]:
102         tokens = []
103
104         while self.currentChar != None:
105             if self.currentChar in '\t':
106                 self.Advance()
107
108             elif self.currentChar in DIGITS:
109                 tokens.append(self.MakeNumber())
110
111             elif self.currentChar == '+':
112                 tokens.append(Token(TokenType.PLUS))
113                 self.Advance()
114             elif self.currentChar == '-':
115                 tokens.append(Token(TokenType.MINUS))
116                 self.Advance()
117             elif self.currentChar == '*':
118                 tokens.append(Token(TokenType.MULTIPLY))
119                 self.Advance()

```



```

120         elif self.currentChar == '/':
121             tokens.append(Token(TokenType.DIVIDE))
122             self.Advance()
123         elif self.currentChar == '(':
124             tokens.append(Token(TokenType.LPAREN))
125             self.Advance()
126         elif self.currentChar == ')':
127             tokens.append(Token(TokenType.RPAREN))
128             self.Advance()
129
130         else:
131             char = self.currentChar
132             startPosition = self.position.Copy()
133             self.Advance()
134             raise IllegalCharacterError(startPosition, self.position, f"'{char}'")
135
136     tokens.append(Token(TokenType.EOF))
137     return tokens
138
139     def MakeNumber(self) -> Token:
140         numString = ''
141         dotCount = 0
142
143         while self.currentChar != None and self.currentChar in DIGITS+'.':
144             if self.currentChar == '.':
145                 if dotCount == 1: break
146                 dotCount += 1
147                 numString += '.'
148             else:
149                 numString += self.currentChar
150             self.Advance()
151
152         if dotCount == 0:
153             return Token(TokenType.INT, int(numString))
154         else:
155             return Token(TokenType.FLOAT, float(numString))
156
157     #####
158     ### RUN ###
159     #####
160
161     def Run(text:str, filename:str) -> None:
162         try:
163             lexer: Lexer = Lexer(text, filename)
164             tokens: list[Token] = lexer.MakeTokens()
165
166             print(tokens)
167         except ShorkError as e:
168             print(e.__repr__())
169
170     def __SignalHandler(sig, frame):
171         sys.exit(0)
172
173     if __name__ == "__main__":
174         signal.signal(signal.SIGINT, __SignalHandler)
175         while True:
176             text = input("> ")
177             Run(text, "<STDIN>")

```