**Laporan Tugas Besar**

**Compiler Bahasa Python**

Mata Kuliah IF2124 - Teori Bahasa Formal dan Otomata



Kelompok 5 Kelas 2

Nama Anggota:

Rahmat Rafid Akbar 13502090

Firizky Ardiansyah 13520095

Roby Purnomo 13520106

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021/2022

Daftar Isi

[**Bab I : Teori Dasar** 2](#_Toc88681820)

[**Finite Automata** 2](#_Toc88681821)

[**Context-Free Grammar** 3](#_Toc88681822)

[**Chomsky Normal Form** 3](#_Toc88681823)

[**Bab II : Hasil** 4](#_Toc88681824)

[**Finite Automata** 4](#_Toc88681825)

[**Context-Free Grammar** 8](#_Toc88681826)

[**Bab III : Implementasi Dan Pengujian** 15](#_Toc88681827)

[**Implementasi pada source code** 15](#_Toc88681828)

[**Testing** 16](#_Toc88681829)

[**BAB IV : Pembagian Tugas** 22](#_Toc88681830)

[**Bab V : Kesimpulan dan Saran** 23](#_Toc88681831)

[**Kesimpulan** 23](#_Toc88681832)

[**Saran** 23](#_Toc88681833)

# **Bab I : Teori Dasar**

## **Finite Automata**

Finite Automata adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa reguler) dan dapat diimplementasikan secara nyata di mana sistem dapat berada di salah satu dari sejumlah berhingga konfigurasi internal disebut state. Beberapa contoh sistem dengan state berhingga antara lain pada mesin minuman otomatis atau *vending machine*, pengatur lampu lalu lintas dan *lexical analyser*.

Suatu finite automata terdiri dari beberapa bagian. Finite automata mempunyai sekumpulan state dan aturan-aturan untuk berpindah dari state yang satu ke state yang lain, tergantung dari simbol nya. Finite automata mempunyai state awal, sekumpulan state dan state akhir. Finite automata merupakan kumpulan dari lima elemen atau dalam bahasa matematis dapat disebut sebagai 5-tuple. Definisi formal dari finite automata dikatakan bahwa finite automata merupakan list dari 5 komponen : kumpulan state, input , aturan perpindahan, state awal, dan state akhir.

Dalam DFA sering digunakan istilah fungsi transisi untuk mendefinisikan aturan perpindahan, biasanya dinotasikan dengan δ. Jika finite automata memiliki sebuah panah dari suatu state x ke suatu state y,dan memiliki label dengan simbol input 0, ini berarti bahwa, jika automata berada pada state x ketika automata tersebut membaca 0, maka automata tersebut dapat berpindah ke state y dapat diindikasikan hal yang sama dengan fungsi transisi dengan mengatakan bahwa δ(x, 0) = y.

Sebuah finite automata terdiri dari lima komponen (Q, Σ, δ, q0, F ), di mana :

1. Q adalah himpunan set berhingga yang disebut dengan himpunan states.
2. adalah himpunan berhingga alfabet dari simbol .
3. adalah fungsi transisi, merupakan fungsi yang mengambil states dan alfabet input sebagai argumen dan menghasilkan sebuah state. Fungsi transisi sering dilambangkan dengan δ.
4. adalah states awal.
5. adalah himpunan states akhir.

## **Context-Free Grammar**

Dalam teori bahasa formal, Context-Free Grammar(CFG) adalah sebuah tata bahasa formal dengan bentuk

Dengan A adalah sebuah symbol non-terminal, dan α adalah terminal dan atau nonterminal. Context-Free Grammar (CFG) adalah tata bahasa yang mempunyai tujuan sama seperti tata bahasa regular yaitu menunjukkan bagaimana menghasilkan suatu bagian-bagian (untai) dalam sebuah bahasa. Context-Free Grammar (CFG) menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan di definisikan dalam tata bahasa bebas konteks. Pohon penurunan (*derivation tree* / *parse tree*) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan di turunkan menjadi terminal sampai tidak ada yang belum tergantikan. Contoh, terdapat CFG dengan aturan produksi sebagai berikut dengan simbol awal S :

Dengan penurunan sebagai berikut, akan menghasilkan aabbaa :

## **Chomsky Normal Form**

Chomsky Normal Form (CNF) merupakan salah satu bentuk normal yang sangat berguna untuk Context-Free Grammar (CFG) . Bentuk normal Chomsky dapat dibuat dari sebuah tata bahasa bebas konteks yang telah mengalami penyederhanaan yaitu penghilangan produksi useless, unit, dan ε. Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi bentuk normal Chomsky dengan syarat tata bahasa bebas kontesk tersebut:

1. Tidak memiliki produksi useless
2. Tidak memiliki produksi unit
3. Tidak memiliki produksi ε

Bentuk normal Chomsky (Chomsky Normal Form, CNF) adalah Context-Free Grammar (CFG) dengan setiap produksinya berbentuk :

atau

# **Bab II : Hasil**

## **Finite Automata**

1. import string
2. def enumAtoms():
3. with open('./src/grammar/syntax-atoms.txt', 'r') as f:
4. lines = f.read()
5. synatom = lines.split('\n')
6. enum = dict(zip(synatom, [chr(ord('A')+i) for i in range(len(synatom))]))
7. return enum
9. def read(filepath: str):
10. f = open(filepath, 'r')
11. w = f.read()
12. alp = string.ascii\_lowercase
13. Alp = string.ascii\_uppercase
14. num = string.digits
15. res = ""
16. atom = enumAtoms()
17. while(w):
18. # string processing
19. if(w[0:3]=="\'\'\'"):
20. nl = 0
21. tmp = w[3:]
22. while(tmp):
23. if(tmp[0]=='\n'):
24. nl+=1
25. if(tmp[0:3]=="\'\'\'"):
26. break
27. tmp = tmp[1:]
28. if(tmp):
29. if(tmp[0:3]=="\'\'\'"):
30. for \_ in range(nl):
31. res += 's\n'
32. res += 's'
33. w = tmp[2:]
34. elif(w[0:3]=="\"\"\""):
35. nl = 0
36. tmp = w[3:]
37. while(tmp):
38. if(tmp[0]=='\n'):
39. nl+=1
40. if(tmp[0:3]=="\"\"\""):
41. break
42. tmp = tmp[1:]
43. if(tmp):
44. if(tmp[0:3]=="\"\"\""):
45. for \_ in range(nl):
46. res += 's\n'
47. res += 's'
48. w = tmp[2:]
49. elif(w[0]=='\"'):
50. tmp = w[1:]
51. while(tmp):
52. if(tmp[0]=='\n' or tmp[0]=='\"'):
53. break
54. tmp = tmp[1:]
55. if(tmp):
56. if(tmp[0]=='\n'):
57. res += w[0]
58. else:
59. res += 's'
60. w = tmp
61. else:
62. res += w[0]
63. elif(w[0]=='\''):
64. tmp = w[1:]
65. while(tmp):
66. if(tmp[0]=='\n' or tmp[0]=='\''):
67. break
68. tmp = tmp[1:]
69. if(tmp):
70. if(tmp[0]=='\n'):
71. res += w[0]
72. else:
73. res += 's'
74. w = tmp
75. else:
76. res += w[0]
77. # cek comment
78. elif(w[0] == '#'):
79. tmp = w
80. w = w[1:]
81. while(w):
82. if(w[0]=='\n'):
83. break
84. tmp = w
85. w = w[1:]
86. if(not w):
87. break
88. else:
89. w = tmp
90. elif(w[0] not in (alp+Alp+num+'\_')):
91. res += w[0]
92. else:
93. if(w[0] not in (alp+Alp+'\_')):
94. # cek floating point dan integer kemudian disubstitusi menjadi 'n'
95. # cek ada var tidak valid, kemudian di assign menjadi 'x'
96. if(w[0] in num):
97. tmp = w
98. w = w[1:]
99. while(w):
100. if(w[0] not in num):
101. break
102. tmp = w
103. w = w[1:]
104. if(not w):
105. res += 'n'
106. break
107. if(w[0]=='.'):
108. tmp = w
109. w = w[1:]
110. while(w):
111. if(w[0] not in num):
112. break
113. tmp = w
114. w = w[1:]
115. if(not w):
116. res += 'n'
117. break
118. if(w[0] in (alp+Alp+'\_')):
119. tmp = w
120. w = w[1:]
121. while(w):
122. if(w[0] not in (alp+Alp+'\_'+num)):
123. break
124. tmp = w
125. w = w[1:]
126. if(not w):
127. res += 'n.x'
128. break
129. w = tmp
130. res += 'n.x'
131. else:
132. res += 'n'
133. w = tmp
135. elif(w[0] in (alp+Alp+'\_')):
136. tmp = w
137. w = w[1:]
138. while(w):
139. if(w[0] not in (alp+Alp+'\_'+num)):
140. break
141. tmp = w
142. w = w[1:]
143. if(not w):
144. res += 'x'
145. break
146. w = tmp
147. res += 'x'
148. else:
149. w = tmp
150. res += 'n'
151. elif(w[0]=='.'):
152. tmp = w
153. w = w[1:]
154. while(w):
155. if(w[0] not in num):
156. break
157. tmp = w
158. w = w[1:]
159. if(not w):
160. res += 'n'
161. break
162. if(w[0] in (alp+Alp+'\_')):
163. tmp = w
164. w = w[1:]
165. while(w):
166. if(w[0] not in (alp+Alp+'\_'+num)):
167. break
168. tmp = w
169. w = w[1:]
170. if(not w):
171. res += 'x'
172. break
173. w = tmp
174. res += 'x'
175. else:
176. res += 'n'
177. else:
178. cur = w[0]
179. tmp = w
180. w = w[1:]
181. while w:
182. if(w[0] not in (alp+Alp+num+'\_')):
183. break
184. else:
185. cur += w[0]
186. tmp = w
187. w = w[1:]
188. if(cur in atom):
189. if(w):
190. res += atom[cur]
191. w = tmp
192. else:
193. res += atom[cur]
194. w = tmp
195. else:
196. if(w):
197. res += 'v'
198. w = tmp
199. else:
200. res+='v'
201. w = tmp
202. tmp = w
203. w = w[1:]
204. return res.replace(" ", "")

## **Context-Free Grammar**

START -> STATEMENTS | NEWLINE | NEWLINE START | START NEWLINE

STATEMENTS -> STATEMENT NEWLINE STATEMENTS | NEWLINE STATEMENTS | STATEMENTS NEWLINE | STATEMENT

STATEMENT -> SIMPLE\_STATEMENT | COMPOUND\_STATEMENT

COMPOUND\_STATEMENT -> DEF\_STATEMENT | IF\_STATEMENT | CLASS\_STATEMENT | WITH\_STATEMENT | FOR\_STATEMENT | WHILE\_STATEMENT

SIMPLE\_STATEMENT -> ASSIGNMENT | STAR\_EXPRESSIONS | RETURN\_STATEMENT | IMPORT\_STATEMENT | RAISE\_STATEMENT | PASS | BREAK | CONTINUE

BLOCK -> NEWLINE STATEMENTS | SIMPLE\_STATEMENT

ATOM -> NAME | TRUE | FALSE | NONE | STRINGS | NUMBER | TUPLE | GROUP | GENERAL\_EXPRESSION | DICT | LIST | SET

// Index processing

SLICE -> EXPRESSION | ASSIGNMENT\_EXPRESSION | COLON | EXPRESSION COLON | COLON EXPRESSION | EXPRESSION COLON EXPRESSION | COLON COLON | EXPRESSION COLON COLON | COLON EXPRESSION COLON | EXPRESSION COLON EXPRESSION COLON | COLON COLON EXPRESSION | EXPRESSION COLON COLON EXPRESSION | COLON EXPRESSION COLON EXPRESSION | EXPRESSION COLON EXPRESSION COLON EXPRESSION

SLICES -> SLICE COMMA SLICES | SLICE

// Argument fungsi

KWARG\_STARRED -> NAME EQ EXPRESSION | STAR EXPRESSION

KWARG\_DOUBLE\_STARRED -> NAME EQ EXPRESSION | STAR STAR EXPRESSION

KWARGS\_STARRED -> KWARG\_STARRED COMMA KWARGS\_STARRED | KWARG\_STARRED

KWARGS\_DOUBLE\_STARRED -> KWARG\_DOUBLE\_STARRED COMMA KWARGS\_DOUBLE\_STARRED | KWARG\_DOUBLE\_STARRED

KWARGS -> KWARGS\_STARRED | KWARGS\_DOUBLE\_STARRED | KWARG\_STARRED COMMA KWARGS\_DOUBLE\_STARRED

NORM\_ARGUMENT -> STAR EXPRESSION COMMA NORM\_ARGUMENT | ASSIGNMENT\_EXPRESSION COMMA NORM\_ARGUMENT | EXPRESSION COMMA NORM\_ARGUMENT | STAR EXPRESSION | ASSIGNMENT | EXPRESSION

ARGS -> NORM\_ARGUMENT | KWARGS | NORM\_ARGUMENT COMMA KWARGS

ARGUMENTS -> ARGS COMMA ARGUMENTS | ARGS

// object

OBJECT -> OBJECT DOT NAME | OBJECT GENERAL\_EXPRESSION | OBJECT L\_BRACHET ARGUMENTS R\_BRACHET | OBJECT LS\_BRACHET SLICES RS\_BRACHET | ATOM

STAR\_OBJECT -> LIST COMMA STAR\_OBJECT | TUPLE COMMA STAR\_OBJECT | OBJECT COMMA STAR\_OBJECT | STAR OBJECT COMMA STAR\_OBJECT | LIST | TUPLE | OBJECT | STAR OBJECT

LIST -> LS\_BRACHET STAR\_NAMED\_EXPRESSIONS RS\_BRACHET | LS\_BRACHET RS\_BRACHET

TUPLE -> L\_BRACHET STAR\_NAMED\_EXPRESSIONS R\_BRACHET | L\_BRACHET R\_BRACHET

GROUP -> L\_BRACHET NAMED\_EXPRESSION R\_BRACHET

SET -> LC\_BRACHET STAR\_EXPRESSIONS RC\_BRACHET

PAIR -> STAR STAR LOGIC\_OBJECT | EXPRESSION COLON EXPRESSION

PAIRS -> PAIR COMMA PAIRS | PAIR

DICT -> LC\_BRACHET PAIRS RC\_BRACHET | LC\_BRACHET RC\_BRACHET

STRINGS -> STRING | STRING STRINGS

// Aritmatika

SINGLETON -> UNARY\_OP SINGLETON | OBJECT

IDENTIFIER -> NAME | OBJECT | L\_BRACHET IDENTIFIER R\_BRACHET | OBJECT DOT NAME | OBJECT LS\_BRACHET SLICES RS\_BRACHET

TERM -> SINGLETON BINARY\_OP TERM | SINGLETON

ARITHMETIC\_OBJECT -> ARITHMETIC\_OBJECT PLUS TERM | ARITHMETIC\_OBJECT MINUS TERM | TERM

// Logic

LOGIC\_OBJECT -> ARITHMETIC\_OBJECT LOGIC\_OP LOGIC\_OBJECT | ARITHMETIC\_OBJECT

// Comparison

COMPARISON\_OBJECT -> NOT COMPARISON\_OBJECT | LOGIC\_OBJECT COMPARISON\_OP COMPARISON\_OBJECT | LOGIC\_OBJECT

COMPOUND\_OBJECT -> COMPARISON\_OBJECT BINARY\_COMPARISON\_OP COMPOUND\_OBJECT | COMPARISON\_OBJECT

// EXPRESSION

EXPRESSION -> COMPOUND\_OBJECT IF COMPOUND\_OBJECT ELSE EXPRESSION | COMPOUND\_OBJECT

EXPRESSIONS -> EXPRESSION COMMA EXPRESSIONS | EXPRESSION

ASSIGNMENT\_EXPRESSION -> NAME SPECIAL\_ASSIGNMENT EXPRESSION

NAMED\_EXPRESSION -> ASSIGNMENT\_EXPRESSION | EXPRESSION

STAR\_NAMED\_EXPRESSION -> STAR LOGIC\_OBJECT | NAMED\_EXPRESSION

STAR\_EXPRESSION -> STAR LOGIC\_OBJECT | EXPRESSION

STAR\_NAMED\_EXPRESSIONS -> STAR\_NAMED\_EXPRESSION COMMA STAR\_NAMED\_EXPRESSIONS | STAR\_NAMED\_EXPRESSION

STAR\_EXPRESSIONS -> STAR\_EXPRESSION COMMA STAR\_EXPRESSIONS | STAR\_EXPRESSION

// Compound EXPRESSION

IF\_EXPRESSION -> IF\_EXPRESSION IF\_EXPRESSION | IF COMPOUND\_OBJECT

FOR\_IF\_EXPRESSION -> FOR STAR\_OBJECT IN COMPOUND\_OBJECT | FOR STAR\_OBJECT IN COMPOUND\_OBJECT IF\_EXPRESSION

FOR\_IF\_EXPRESSIONS -> FOR\_IF\_EXPRESSION FOR\_IF\_EXPRESSIONS | FOR\_IF\_EXPRESSION

GENERAL\_EXPRESSION -> NAMED\_EXPRESSION FOR\_IF\_EXPRESSIONS

// statement

ASSIGNMENT -> NAME COLON EXPRESSION | NAME COLON EXPRESSION EQ STAR\_EXPRESSIONS | IDENTIFIER ASSIGNMENT\_OP STAR\_EXPRESSIONS

// Import

PACKAGE -> PACKAGE DOT NAME | NAME

PACKAGES -> PACKAGES AS NAME COMMA PACKAGES | PACKAGES DOT NAME COMMA PACKAGES | NAME COMMA PACKAGES| PACKAGES DOT NAME | NAME | PACKAGES AS NAME

FROM\_PACKAGES -> STAR | PACKAGES

IMPORT\_STATEMENT -> IMPORT PACKAGES | FROM ELIPSIS PACKAGE IMPORT FROM\_PACKAGES | FROM ELIPSIS IMPORT FROM\_PACKAGES | FROM PACKAGE IMPORT FROM\_PACKAGES

// if statement

IF\_STATEMENT -> IF NAMED\_EXPRESSION COLON BLOCK | IF NAMED\_EXPRESSION COLON BLOCK ELIF\_STATEMENT | IF NAMED\_EXPRESSION COLON BLOCK ELSE\_STATEMENT

ELIF\_STATEMENT -> ELIF NAMED\_EXPRESSION COLON BLOCK | ELIF NAMED\_EXPRESSION COLON BLOCK ELSE\_STATEMENT | ELIF NAMED\_EXPRESSION COLON BLOCK ELIF\_STATEMENT

ELSE\_STATEMENT -> ELSE COLON BLOCK

// while loop

WHILE\_STATEMENT -> WHILE NAMED\_EXPRESSION COLON BLOCK | WHILE NAMED\_EXPRESSION COLON BLOCK ELSE\_STATEMENT

// for loop

FOR\_STATEMENT -> FOR STAR\_OBJECT IN STAR\_EXPRESSIONS COLON BLOCK | FOR STAR\_OBJECT IN STAR\_EXPRESSIONS COLON BLOCK ELSE\_STATEMENT

// with statement

WITH\_STATEMENT -> WITH L\_BRACHET WITH\_CONTENT R\_BRACHET COLON BLOCK | WITH WITH\_CONTENT COLON BLOCK

WITH\_CONTENT -> WITH\_CONTENT COMMA WITH\_CONTENT | EXPRESSION AS STAR\_OBJECT | EXPRESSION

// class statement

CLASS\_STATEMENT -> CLASS NAME COLON BLOCK | CLASS NAME L\_BRACHET R\_BRACHET COLON BLOCK | CLASS NAME L\_BRACHET ARGUMENTS R\_BRACHET COLON BLOCK

// raise

RAISE\_STATEMENT -> RAISE | RAISE EXPRESSION | RAISE EXPRESSION FROM EXPRESSION

// return

RETURN\_STATEMENT -> RETURN | RETURN STAR\_EXPRESSIONS

// def

DEF\_STATEMENT -> DEF NAME L\_BRACHET R\_BRACHET COLON BLOCK | DEF NAME L\_BRACHET PARAMS R\_BRACHET COLON BLOCK

// DEF\_PARAMS

PARAMS -> PAR\_ONE | PAR\_ONE STAR\_ETC | PAR\_TWO | PAR\_TWO STAR\_ETC | PAR\_THREE | PAR\_THREE STAR\_ETC | PAR\_FOUR | PAR\_FOUR STAR\_ETC | STAR\_ETC

PAR\_ONE -> PAR\_ONE\_SEC | PAR\_ONE\_SEC PARAM\_WITH\_DEFAULT | PAR\_ONE PARAM\_WITH\_DEFAULT

PAR\_ONE\_SEC -> SLASH\_NO\_DEFAULT | SLASH\_NO\_DEFAULT PARAM\_NO\_DEFAULT | PAR\_ONE\_SEC PARAM\_NO\_DEFAULT

PAR\_TWO -> SLASH\_WITH\_DEFAULT | SLASH\_WITH\_DEFAULT PARAM\_WITH\_DEFAULT | PAR\_TWO PARAM\_WITH\_DEFAULT

PAR\_THREE -> PAR\_THREE\_SEC | PAR\_THREE\_SEC PARAM\_WITH\_DEFAULT | PAR\_THREE PARAM\_WITH\_DEFAULT

PAR\_THREE\_SEC -> PARAM\_NO\_DEFAULT PAR\_THREE\_SEC | PARAM\_NO\_DEFAULT

PAR\_FOUR -> PARAM\_WITH\_DEFAULT PAR\_FOUR | PARAM\_WITH\_DEFAULT

// SLASH\_PARAMS

SLASH\_NO\_DEFAULT -> SLASH\_NO\_DEFAULT\_ONE SLASH COMMA | SLASH\_NO\_DEFAULT\_ONE SLASH

SLASH\_NO\_DEFAULT\_ONE -> PARAM\_NO\_DEFAULT | PARAM\_NO\_DEFAULT SLASH\_NO\_DEFAULT\_ONE

SLASH\_WITH\_DEFAULT -> SLASH\_WITH\_DEFAULT\_ONE | SLASH\_WITH\_DEFAULT\_TWO

SLASH\_WITH\_DEFAULT\_ONE -> SLASH\_WITH\_DEFAULT\_ONE\_SEC SLASH COMMA | PARAM\_NO\_DEFAULT SLASH\_WITH\_DEFAULT\_ONE\_SEC SLASH COMMA | PARAM\_NO\_DEFAULT SLASH\_WITH\_DEFAULT\_ONE

SLASH\_WITH\_DEFAULT\_ONE\_SEC -> PARAM\_WITH\_DEFAULT | PARAM\_WITH\_DEFAULT SLASH\_WITH\_DEFAULT\_ONE\_SEC

SLASH\_WITH\_DEFAULT\_TWO -> SLASH\_WITH\_DEFAULT\_ONE\_SEC SLASH | PARAM\_NO\_DEFAULT SLASH\_WITH\_DEFAULT\_ONE\_SEC SLASH | PARAM\_NO\_DEFAULT SLASH\_WITH\_DEFAULT\_TWO

STAR\_ETC -> STAR\_ETC\_ONE | STAR\_ETC\_ONE KWDS | STAR\_ETC\_TWO | STAR\_ETC\_TWO KWDS | KWDS

STAR\_ETC\_ONE -> STAR PARAM\_NO\_DEFAULT | STAR PARAM\_NO\_DEFAULT PARAM\_MAYBE\_DEFAULT | STAR\_ETC\_ONE PARAM\_MAYBE\_DEFAULT

STAR\_ETC\_TWO -> STAR COMMA STAR\_ETC\_TWO\_SEC

STAR\_ETC\_TWO\_SEC -> PARAM\_MAYBE\_DEFAULT | PARAM\_MAYBE\_DEFAULT STAR\_ETC\_TWO\_SEC

KWDS -> STAR STAR PARAM\_NO\_DEFAULT

PARAM\_NO\_DEFAULT -> PARAM COMMA | PARAM

PARAM\_WITH\_DEFAULT -> PARAM DEFAULT COMMA | PARAM DEFAULT

PARAM\_MAYBE\_DEFAULT -> PARAM COMMA | PARAM DEFAULT COMMA | PARAM | PARAM DEFAULT

PARAM -> NAME | NAME ANNOTATION

ANNOTATION -> COLON EXPRESSION

DEFAULT -> EQ EXPRESSION

ASSIGNMENT\_OP -> + = | - = | \* = | @ = | / = | % = | & = | OR\_OP = | ^ = | < < = | > > = | \* \* = | / / = | =

SPECIAL\_ASSIGNMENT -> : =

UNARY\_OP -> PLUS | MINUS | TILDE

BINARY\_OP -> \* | / | / / | % | @ | \* \*

LOGIC\_OP -> > > | < < | & | ^ | OR\_OP

EXISTENTIAL\_OP -> NOT IN | IN

SIMILARITY\_OP -> IS NOT | IS

COMPARISON\_OP -> < | > = | < | < = | = = | ! = | EXISTENTIAL\_OP | SIMILARITY\_OP

BINARY\_COMPARISON\_OP -> AND | OR

ELIPSIS -> DOT | DOT DOT DOT | ELIPSIS ELIPSIS

COMMA -> ,

COLON -> :

PLUS -> +

MINUS -> -

TILDE -> ~

STAR -> \*

ARROW -> ->

DOT -> .

EQ -> =

SLASH -> /

AT\_SIGN -> @

LC\_BRACHET -> {

RC\_BRACHET -> }

LS\_BRACHET -> [

RS\_BRACHET -> ]

L\_BRACHET -> (

R\_BRACHET -> )

NEWLINE -> NEWLINE NEWLINE | \_\_nl\_\_

OR\_OP -> \_\_or\_\_

FALSE -> A

CLASS -> B

IS -> C

RETURN -> D

NONE -> E

CONTINUE -> F

FOR -> G

TRUE -> H

DEF -> I

FROM -> J

WHILE -> K

AND -> L

NOT -> M

WITH -> N

AS -> O

ELIF -> P

IF -> Q

OR -> R

ELSE -> S

IMPORT -> T

PASS -> U

BREAK -> V

IN -> W

RAISE -> X

NAME -> v

NUMBER -> n

STRING -> s

# **Bab III : Implementasi Dan Pengujian**

## **Implementasi pada source code**

1. readGrammar

Fungsi read grammar digunakan untuk membaca grammar yang berasal dari file .txt yang telah dibuat (grammar.txt). Fungsi ini mengembalikan CFG dalam bentuk dictionary (pada python).

1. CFGtoCNF

Fungsi CFGtoCNF memiliki parameter masukan berupa CFG lalu CFG ini akan dimodifikasi sehingga memenuhi bentuk CNF (Chomsky Normal Form) yang juga menggunakan tipe data dictionary.

1. CYK

Algoritma CYK disini digunakan untuk memeriksa substring dari string input mana yang dikenali oleh CFG yang diberikan setiap line dari file .py yang akan dievaluasi syntaxnya.

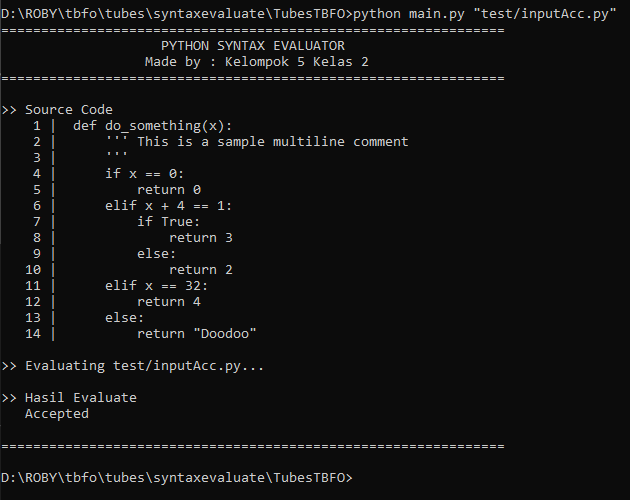
1. Main Program

Pada main program akan dilakukan pemanggilan fungsi-fungsi diatas, dan akan menghandle input dan output pada program dengan pengguna.

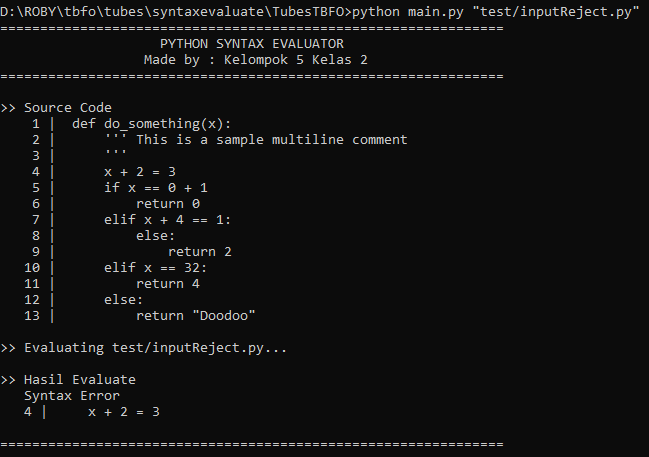
Link Repository : <https://github.com/firizky29/TubesTBFO>

## **Testing**

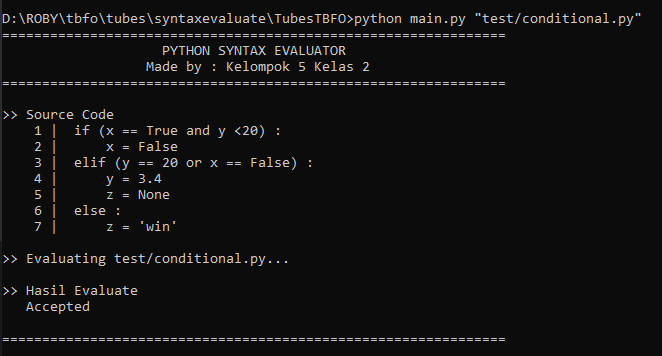
1. Testing dari Spek
   1. inputAcc.py



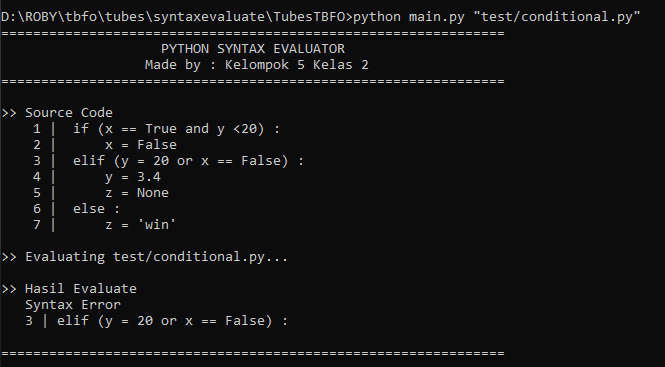
* 1. inputReject.py



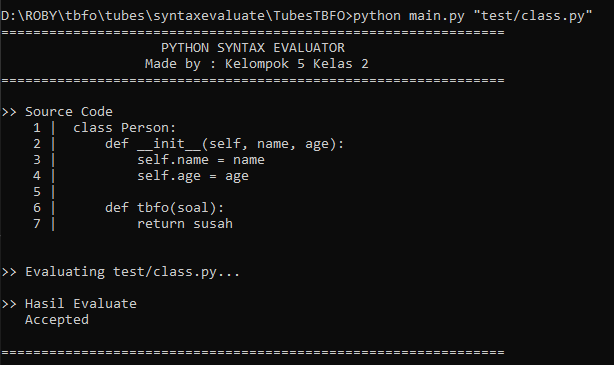
1. Conditional
2. Accepted



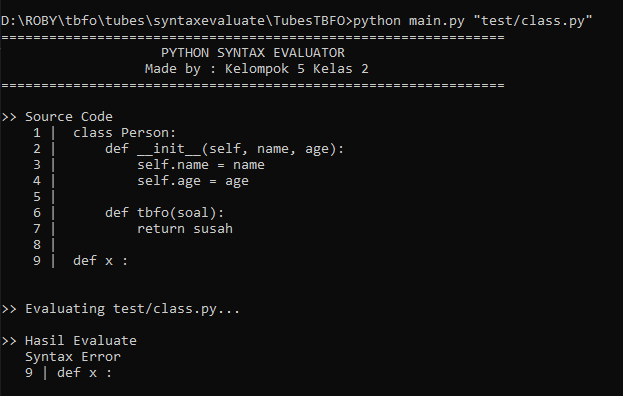
1. Syntax Error



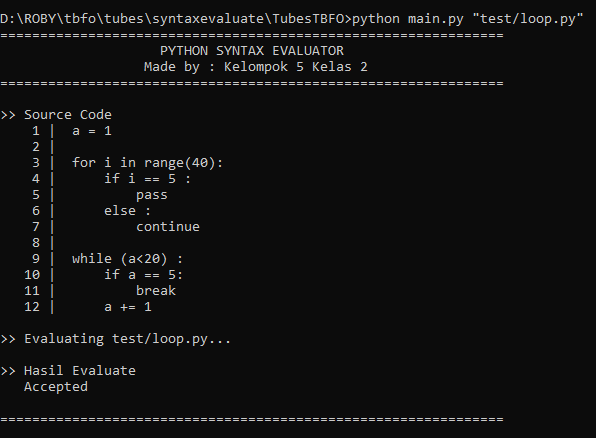
1. Class + def
   1. Accepted



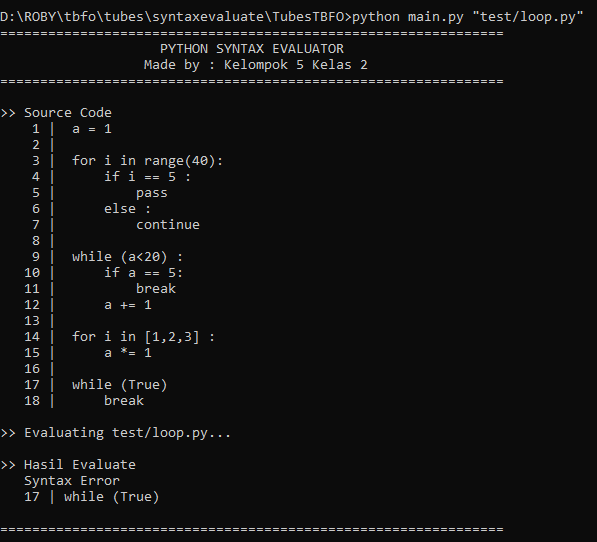
* 1. Syntax Error



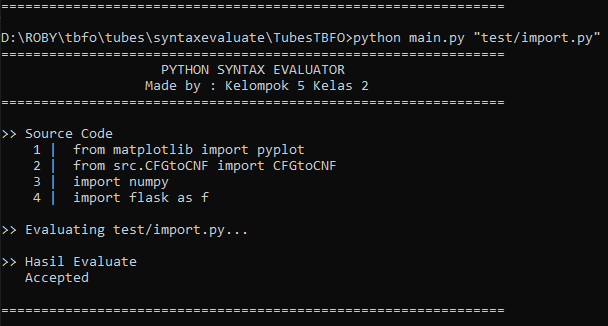
1. Loop (for and while)
   1. Accepted



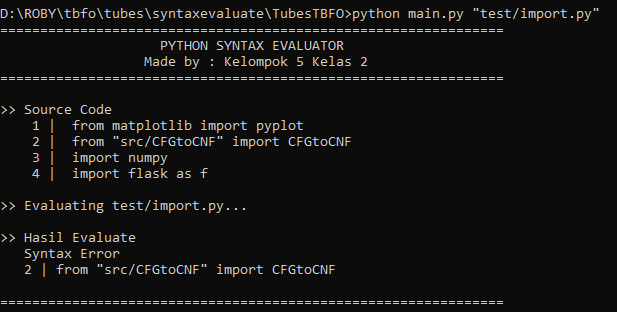
* 1. Syntax Error



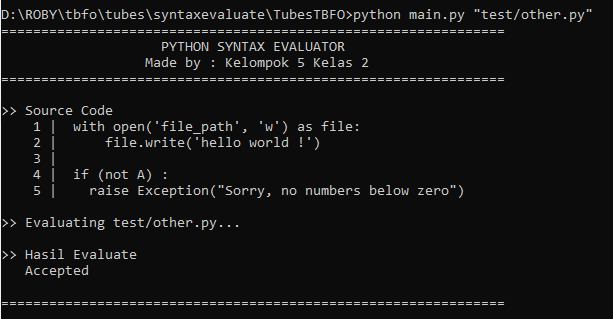
1. Import
   1. Accepted



* 1. Syntax Error



1. Other



# **BAB IV : Pembagian Tugas**

|  |  |  |
| --- | --- | --- |
| Anggota | NIM | Tugas |
| Rahmat Rafid Akbar | 13502090 | * Membuat CFG * Debugging * Laporan |
| Firizky Ardiansyah | 13520095 | * Algoritma read file to grammar * Algoritma CYK * Membuat CFG * Debugging |
| Roby Purnomo | 13520106 | * Algoritma convert CFG to CNF * Laporan * Main Program * Debugging |

# **Bab V : Kesimpulan dan Saran**

## **Kesimpulan**

1. Kami telah berhasil membuat FA dan CFG untuk mengecek string apakah diterima oleh syntax python.
2. Kami berhasil membuat program dengan python yang memanfaatkan FA dan CFG yang kami buat dengan menggunakan bantuan beberapa fungsi yang telah dipaparkan diatas.

## **Saran**

Dikarenakan dalam tugas besar ini diberikan banyak pengecualian syntax, mungkin kedepannya dapat lebih menyempurnakan tugas besar TBFO seperti penggunaan indentasi, r-string, f-string, dll.