

# **LAPORAN TUGAS BESAR**

## ***COMPILER BAHASA PYTHON***

Laporan dibuat untuk memenuhi salah satu tugas besar mata kuliah

IF2124 Teori Bahasa Formal dan Otomata



Disusun oleh:

### **Kelompok 7**

<b>Maria Khelli</b>	<b>13520115</b>
<b>Alifia Rahmah</b>	<b>13520122</b>
<b>Hilda Carissa Widelia</b>	<b>13520164</b>

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**

**2021**

# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>1</b>
<b>BAB 1 : Teori Dasar</b>	<b>3</b>
1.1 Python	3
1.2. Finite Automata	3
1.3. Context Free Grammar	5
1.4. CNF (Chomsky Normal Form)	6
1.5. CYK (Cocke-Younger-Kasami)	6
1.6 Penjelasan Sintaks Python	8
1.6.1 Comments	8
1.6.2 Variabel	8
1.6.3 Operator	9
1.6.4 List	12
1.6.5 Tuples	13
1.6.6 Set	13
1.6.7 Dictionary	14
1.6.8 If... Else	14
1.6.9 While loop	15
1.6.10 For loop	15
1.6.11 Functions	16
1.6.12 Class	17
<b>BAB 2 : Hasil FA dan CFG</b>	<b>18</b>
2.1 Hasil FA	18
2.2 Hasil CFG	18
<b>BAB 3 : Implementasi dan Pengujian Program</b>	<b>24</b>
3.1. Spesifikasi Teknis Program	24
3.1.1 Struktur Folder	24
3.1.2 Struktur Data yang Dibuat	24
3.1.2.1 Lexer (lexer.py)	24
3.1.2.2 Token (lexer.py)	25
3.1.2.3 Position (lexer.py)	26
3.1.2.4 Parser (cyk_parser.py)	26
3.1.2.5 Error (error.py)	27
3.1.2.6 Variable Name Checker (var.py)	27
3.1.3 Header Fungsi	28

3.1.3.1 File cfg2cnf.py	28
3.1.4 Daftar Token	29
3.2. Screenshot contoh kasus	30
3.2.0. Contoh Kasus pada Spek Tugas	30
3.2.1. Test Case 1	30
3.2.2 Test Case 2	31
3.2.3 Test Case 3	31
3.2.4 Test Case 4	32
3.2.5 Test Case 5	32
3.2.6 Test Case 6	32
3.2.7 Test Case 7	32
3.2.8 Test Case 8	33
3.2.9 Test Case 9	33
3.2.10 Test Case 10	33
3.2.11 Test tambahan lain	34
3.3. Link Repository Github	35
<b>BAB 4 : Pembagian Tugas</b>	<b>36</b>
<b>Daftar Referensi</b>	<b>37</b>

# BAB 1 : Teori Dasar

## 1.1 Python

Python adalah bahasa pemrograman berbasis interpreter tingkat tinggi, yang diciptakan oleh Guido van Rossum dan dirilis pertama kali tahun 1991. Dalam sintaksnya, Python memprioritaskan code readability dengan penggunaan *whitespace* serta kesederhanaannya yang bisa membuat pengguna lebih fokus dalam menyelesaikan masalah dan memikirkan algoritma dibanding memusingkan penggunaan bahasa.

Python adalah bahasa pemrograman multi-paradigma dan mendukung banyak jenis pemrograman seperti pemrograman berorientasi objek, pemrograman terstruktur, pemrograman fungsional, dan pemrograman berorientasi objek. Paradigma lain juga banyak didukung dengan menggunakan ekstensi seperti desain berdasarkan kontrak dan pemrograman logika. Python dirancang untuk menjadi sangat dapat dikembangkan dibanding memiliki semua fungsionalitas dalam intinya. Maka dari itu, python menjadi sangat populer karena modularitasnya yang ringkas membuat python menjadi sarana yang cocok untuk menambahkan antarmuka yang dapat diprogram ke aplikasi yang ada.

Python sendiri dimaksudkan untuk menjadi bahasa yang dapat mudah dibaca. Oleh karena itu, pemformatannya tidak berantakan secara visual, dan seringkali menggunakan bahasa inggris untuk kata kunci. Python tidak membutuhkan tanda baca kurung awal untuk membatasi blok, dan tanda baca titik koma setelah pernyataan pun bersifat opsional.

Berikut contoh output dalam python :

```
print("Hello, World!")
```

Sedangkan berikut contoh input untuk python :

```
x = input("Masukkan x : ")
```

## 1.2. Finite Automata

Finite automata adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana yaitu bahasa reguler dan dapat diimplementasikan secara nyata, seperti contohnya *vending machine*, pengatur lampu lalu lintas, dan *lexical analyser*. Finite automata atau bisa disebut juga Finite state machine, memiliki lima komponen atau tuple. Finite automata memiliki jumlah state dan rules yang banyak untuk dapat berpindah-pindah dari sebuah state ke state lainnya. Sebuah Finite Automata terdiri dari :

Q : Kumpulan State

$\Sigma$  : Kumpulan simbol input

q : state awal

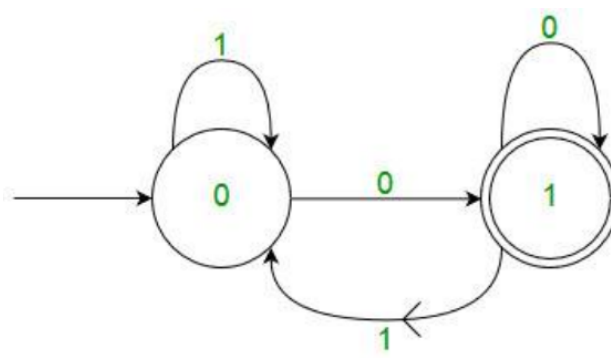
F : kumpulan state final

$\delta$  : Fungsi transisi

Finite Automata sendiri terbagi menjadi 2. Pertama ada DFA yaitu Deterministic Finite Automata. Terdiri dari 5 tuples, sama seperti yang sudah disebutkan diatas, dengan definisi dari fungsi transisinya adalah sebagai berikut :

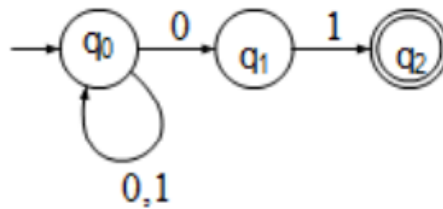
$\delta$  : Fungsi transisi ( $\delta : Q \times \Sigma \rightarrow Q$ )

Dalam DFA, untuk sebuah input character yang spesifik, mesin akan menuju ke satu state saja. Fungsi transisi didefinisikan untuk semua state untuk semua simbol input. Selain itu, di DFA, null atau  $\epsilon$  tidak diperbolehkan, sehingga DFA tidak dapat berubah statenya tanpa ada input character. Terdapat banyak DFA yang mungkin untuk sebuah pola tertentu, akan tetapi yang biasa digunakan adalah DFA dengan jumlah state paling minimal. Berikut contoh diagram DFA dengan input simbol 0 dan 1 yang menerima semua string berakhiran 0.



**Gambar 1** DFA dengan  $\Sigma = \{0, 1\}$

Selain DFA, terdapat juga NFA yaitu Nondeterministic Finite Automata. Sebenarnya, NFA mirip dengan DFA namun terdapat dua perbedaan. Pertama untuk simbol input, NFA memperbolehkan adanya  $\epsilon$  atau null sehingga dapat berubah state tanpa adanya input character. Kedua, kemampuan untuk menimbulkan transisi ke beberapa state untuk sebuah input tertentu. Dalam NFA, jika ada sebuah “jalur” untuk sebuah input string mencapai final state, maka input string itu diterima. Berikut contoh diagram NFA yang menerima semua string berakhiran 01.



**Gambar 2** NFA yang menerima string berakhiran 01.

Maka sebagai kesimpulan, semua DFA adalah NFA, yang berarti semua DFA dapat diubah menjadi NFA tapi tidak berarti semua NFA adalah DFA. Namun, ada ekuivalen DFA untuk semua NFA. Selain itu, bisa ada lebih dari 1 final state untuk DFA dan NFA.

### 1.3. Context Free Grammar

Context-Free Grammar adalah bahasa yang memiliki cakupan lebih luas dibanding *regular language*, yang menggunakan notasi rekursif. CFG ini berguna untuk struktur bersarang seperti tanda kurung dalam bahasa pemrograman. Idennya adalah dengan menggunakan variabel sebagai pengganti satu set string yang kemudian didefinisikan secara rekursif. CFG terdiri dari 4 komponen yaitu :

T adalah sekumpulan simbol-simbol yang menjadi string untuk bahasa yang sedang didefinisikan

N adalah non-terminal (atau bisa juga disebut variabel), yaitu sekumpulan simbol-simbol lain yang merepresentasikan sebuah bahasa

P adalah aturan produksi yang merupakan definisi rekursif sebuah bahasa. Suatu P memiliki bentuk  $X \rightarrow Y$ , dimana X adalah variabel dan Y adalah terminal atau variabel yang akan membentuk sebuah string lagi

S adalah start symbol variabel yang bahasanya sedang didefinisikan, atau secara singkat variabel yang akan dipanggil pertama kali.

Komponen CFG ini biasanya dinotasikan dengan *four-tuple* sebagai  $G = (N, T, P, S)$ . Misalnya grammar  $G = (\{S\}, \{0, 1\}, P, S)$ ,  $P : S \rightarrow 01, S \rightarrow 0S1$ , yang berarti grammar ini memiliki variabel atau non-terminal S, dua terminal yaitu 0 dan 1, *start symbol* S, dan production rule P yaitu  $S \rightarrow 01$ , dan  $S \rightarrow 0S1$ .

Context Free Grammar menjadi dasar dalam pembentukan suatu parser atau proses analisis sintaksis. Bagian sintaks dalam suatu kompilator biasanya didefinisikan dalam sebuah CFG. Simbol-simbol variabel akan digambarkan dengan menggunakan pohon penurunan atau *dervation tree/parser tree* menjadi simbol terminal. Setiap simbol variabel akan diturunkan

sampai tidak ada yang belum tergantikan. Puncak akar dari sebuah *parser tree* haruslah berupa input symbol, kemudian puncak-puncak dibawahnya adalah simbol-simbol nonterminal, dan daun atau *leave*-nya adalah simbol terminal atau  $\epsilon$ .

#### 1.4. CNF (Chomsky Normal Form)

CNF atau Chomsky Normal Form adalah bentuk formal lain dari sebuah CFG. Sebuah CFG dikatakan memiliki bentuk CNF jika semua aturan produksinya mengikuti kondisi sebagai berikut :

1. Start symbol menghasilkan  $\epsilon$ . Contoh :  $A \rightarrow \epsilon$
2. Non-terminal menghasilkan dua buah non-terminal lain. Contoh :  $A \rightarrow BC$
3. Non-terminal menghasilkan sebuah terminal. Contoh :  $A \rightarrow a$

Dimana A, B, C adalah simbol non-terminal, a adalah simbol terminal, S adalah start symbol,  $\epsilon$  merepresentasikan string kosong dan B dan C bukan start symbol. Semua grammar di Chomsky Normal Form ini adalah context-free grammar, dan sebaliknya, context-free grammar dapat diubah menjadi ekuivalen dari chomsky normal form yang memiliki ukuran tidak lebih besar dari kuadrat ukuran grammar aslinya.

Contoh :

$G1 = (S \rightarrow AB, S \rightarrow c, A \rightarrow a, B \rightarrow b)$

$G2 = (S \rightarrow aA, A \rightarrow a, B \rightarrow c)$

Aturan produksi dari G1 memenuhi syarat untuk CNF, maka grammar G1 adalah dalam CNF. Namun aturan produksi G2 tidak memenuhi syarat sebuah CNF karena dapat dilihat  $S \rightarrow aA$  adalah terminal yang diikuti dengan non-terminal. Maka G2 bukanlah sebuah CNF.

#### 1.5. CYK (Cocke-Younger-Kasami)

CYK atau Cocke-Younger-Kasami adalah algoritma parsing untuk CFG yang dipublikasikan oleh Itiroo Sakai pada tahun 1961. Algoritma ini dinamakan berdasarkan para penemunya yaitu John Cocke, Daniel Younger, Tadao Kasami dan Jacob T. Schwartz. Versi standar dari CYK hanya dapat bekerja untuk CFG yang sudah dalam bentuk CNF. CYK menyelesaikan problem-problem tertentu menggunakan pendekatan *dynamic programming*.

CYK menjadi penting karena efisiensinya dalam beberapa situasi. Dengan Big O Notation, run time terlama dari sebuah CYK adalah  $O(n^3 \cdot |G|)$  dengan n sebagai panjang dari string yang sudah di-*parse* dan G adalah besarnya CNF dari grammar G. Dengan Big O Notation itu, CYK menjadi algoritma parsing paling efisien meskipun terdapat algoritma lain dengan rata-rata waktu yang dibutuhkannya lebih sedikit.

Untuk contoh penggunaan CYK, berikut ada contoh grammar dalam bentuk CNF :

$S \rightarrow NP VP$   
 $VP \rightarrow VP PP$   
 $VP \rightarrow V NP$   
 $VP \rightarrow \text{eats}$   
 $PP \rightarrow P NP$   
 $NP \rightarrow \text{Det } N$   
 $NP \rightarrow \text{she}$   
 $V \rightarrow \text{eats}$   
 $P \rightarrow \text{with}$   
 $N \rightarrow \text{fish}$   
 $N \rightarrow \text{fork}$   
 $\text{Det} \rightarrow \text{a}$

**Gambar 3** Contoh grammar CNF untuk analisis CYK

**CYK table**

<b>S</b>						
	VP					
<b>S</b>						
	VP			PP		
<b>S</b>		NP			NP	
NP	V, VP	Det.	N	P	Det	N
she	eats	a	fish	with	a	fork

**Gambar 4** Tabel CYK untuk grammar di gambar 3.



## 1.6 Penjelasan Sintaks Python

Dalam bahasa pemrograman python, terdapat beberapa syntax yang sering digunakan. Dalam sub-bab ini, akan dijelaskan aturan penggunaan beberapa syntax python.

### 1.6.1 Comments

*Comments* dalam semua bahasa pemrograman digunakan untuk memperjelas sebuah kode, untuk membuat sebuah kode lebih mudah terbaca, dan dapat juga digunakan untuk menghindari sebuah blok kode dijalankan. Untuk menciptakan sebuah komen dalam sebuah file program dalam bahasa Python, dapat digunakan tanda pagar '#' untuk satu baris komentar. Namun, untuk komentar yang memiliki banyak baris, python belum memiliki syntaxnya. Akan tetapi, python akan mengabaikan *multiline* string yang tidak di-*assign* ke sebuah variabel. Oleh karena itu, dapat digunakan kutip 3 (di depan dan di belakang komentar) untuk menuliskan sebuah komentar *multiline*. Contoh komentar dalam python

```
# This is a comment
'''
This is also a comment
'''
```

### 1.6.2 Variabel

Variabel adalah semacam kotak untuk menyimpan nilai sebuah data. Dalam python sendiri, tidak ada command khusus untuk mendeklarasikan sebuah variabel. Variabel akan dibentuk saat pertama kali *assign* sebuah value. Tidak perlu spesifik menyatakan tipe sebuah variabel, karena python dapat langsung mengetahui variabel ini tipe nya apa. Setelah *assign* sebuah value ke dalam sebuah variabel, kita bisa mengganti value dalam variabel itu, bahkan mengganti dengan tipe data lain. Contoh

```
x = 3 #Disini x merupakan tipe data integer
x = "Apple" #sedangkan disini x adalah tipe data string
```

Beberapa hal yang dapat dilakukan di Python adalah casting (mengubah suatu tipe data) misalnya jika kita *assign* 3 ke dalam x, lalu kita ingin mengubah dari tipe data integer menjadi string, kita bisa tinggal *assign* ulang x dengan str(3) dan x akan menjadi '3'. Lalu kita bisa juga mengecek tipe sebuah variabel dengan type(<nama variabel>). Misalkan x yang tadi sudah kita assign sebagai str(3), jika kita kemudian menuliskan *command* print(type(x)), maka yang keluar adalah <class 'str'> menandakan tipe datanya adalah string. Untuk *assign* sebuah string, diperlukan tanda petik. Dalam python tanda petik yang digunakan tidak begitu penting asalkan sama. Misal saat awal menggunakan petik satu, maka harus diakhiri dengan petik satu, begitu juga saat menggunakan petik dua, maka harus diakhiri dengan petik dua.

Hal yang harus diingat dalam penamaan variabel :

1. Harus dimulai dengan huruf atau karakter underscore “\_”
2. Tidak dapat dimulai dengan angka
3. Nama sebuah variabel hanya dapat berisi huruf (A-z), angka (0-9) dan underscore “\_”
4. Variabel dalam python *case-sensitive* yang berarti variabel A dan a adalah dua variabel yang berbeda
5. Dapat *assign* beberapa data ke beberapa variabel dalam satu line yang sama. Contoh

```
x, y, z = 1, 2, 3
```

Maka saat kita `print(x)`, yang keluar adalah 1 dan jika kita menuliskan *command* `print(y+z)` maka yang keluar adalah 5.

Contoh variabel yang salah :

1. My-var → karena menggunakan “-”
2. 1variabel → karena diawali dengan angka
3. My var → karena ditengah penamaan terdapat spasi

### 1.6.3 Operator

Dalam python terdapat beberapa jenis operator

1. Operator Aritmatik

Operator	Nama	Contoh Penggunaan
+	Tambah / Addition	<code>x + y</code>
-	Kurang / Subtraction	<code>x - y</code>
*	Kali / Multiplication	<code>x * y</code>
/	Bagi / Division	<code>x / y</code>
%	Modulus	<code>x % y</code>
**	Pangkat / Power	<code>x ** y</code>
//	Floor Division (Pembagian dengan pembulatan kebawah)	<code>x // y</code>

## 2. Assignment Operation

Operator	Contoh	Sama dengan
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
=	x  = 5	x = x   5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

## 3. Operator perbandingan

Operator	Nama	Contoh
==	Sama dengan	x == y
!=	Tidak sama dengan	x != y
>	Lebih besar	x > y
<	Lebih kecil	x < y
>=	Lebih besar sama dengan	x >= y
<=	Lebih kecil sama dengan	x <= y

4. Operator logika

Operator	Deskripsi	Contoh
and	Mengembalikan true jika kedua pernyataan benar	$x > 1$ and $x < 5$
or	Mengembalikan true jika salah satu pernyataan benar	$x < 5$ or $x > 10$
not	Mengembalikan kebalikan hasil, kalau misal true jadi false, dan sebaliknya	not( $x < 5$ or $x > 10$ )

5. Operator Identitas

Operator	Deskripsi	Contoh
is	Mengembalikan True jika kedua variabel adalah objek yang sama	$x$ is $y$
Is not	Mengembalikan True jika kedua variabel adalah bukan objek yang sama	$x$ is not $y$

6. Operator keanggotaan

Operator	Deskripsi	Contoh
in	Mengembalikan True jika ada sebuah sekuens di $y$ yang merupakan $x$	$x$ in $y$
Not in	Mengembalikan True jika tidak ada sekuens di $y$ yang merupakan $x$	$x$ not in $y$

## 7. Operator bitwise

Operator	Nama	Deskripsi
&	AND	Menjadikan bit 1 jika keduanya 1
	OR	Menjadikan bit 1 jika salah satu dari keduanya 1
^	XOR	Menjadikan bit 1 jika hanya satu dari keduanya 1
~	NOT	Inverse semua bit
<<	Zero fill left shift	Shift left dengan mengisi 0 dari kanan dan membiarkan bit paling kiri hilang
>>	Zero fill right shift	Shift right dengan mengisi bit yang sama dengan bit paling kiri dan membiarkan bit paling kanan hilang

### 1.6.4 List

List digunakan untuk menyimpan beberapa objek dalam sebuah variabel. Untuk membuat sebuah list dapat menggunakan command seperti ini :

```
listnew = ["a", "b", "c"]
```

Sebuah list memiliki index yang selalu dimulai dari 0. List dapat diubah dan dapat berisi 2 atau lebih objek yang sama. List juga pasti teratur, maksudnya jika kita menambahkan item baru, pasti akan ditempatkan di urutan paling belakang. Sebuah list dapat berisi berbagai macam tipe data. Bisa berisi string, boolean, integer, float, bisa juga campuran dari beberapa data type. Selain langsung menuliskan dalam kurung siku, kita juga dapat membuat list dengan command list(). Contoh :

```
list2 = list((1, 2, 3, 4))
```

Untuk mengakses nilai dari sebuah list, dapat digunakan <nama variabel list>[<index>]. Contoh untuk list2, jika kita ingin mengakses angka 2, maka kita dapat menggunakan list2[1] karena angka 2 ada di index ke 1.

### 1.6.5 Tuples

Tuple sebenarnya mirip dengan list. Satu perbedaan yang paling mencolok adalah, tuple tidak dapat diubah. Element dalam list dapat ditambah, dihapus, diubah sedangkan dalam tuple, jika sudah membuat atau mendefinisikan sebuah tuple, maka tidak dapat diubah-ubah lagi. Cara pendefinisian mirip dengan list, yaitu

```
inituple = ("a", "b", "c")
```

Sama seperti list, sebuah tuple dapat diisi dengan berbagai macam tipe data seperti boolean, integer, float, dan dapat diakses dengan indexnya seperti `inituple[0]` yang akan menghasilkan "a". Untuk membuat tuple dengan hanya satu item, tetap harus menggunakan koma. Contoh

```
tuplesendiri = ("a", )
```

Jika koma tidak digunakan, maka akan menjadi string dan bukan tuple. Selain dengan langsung menuliskan elemen, sebuah tuple juga dapat didefinisikan dengan menggunakan `tuple()`. Contoh :

```
inituplejuga = tuple(11,2,3,4,5)
```

### 1.6.6 Set

Sama seperti list dan tuples, Set juga digunakan untuk menyimpan beberapa nilai dalam sebuah variabel. Namun, yang menjadi perbedaan adalah set tidak terurut beraturan dan tidak dapat diakses secara langsung menggunakan index seperti dalam tuple dan list. Sehingga jika kita menggunakan for loop untuk print semua elemen dalam sebuah set, maka hasilnya bisa berbeda urutannya. Elemen-elemen dalam set tidak dapat diubah, akan tetapi kita dapat menambah atau menghapus elemen. Selain itu, dalam set tidak diperbolehkan ada dua nilai yang sama. Untuk mendefinisikan sebuah set adalah sebagai berikut

```
iniset = {"a", "b", "c"}
```

Sama seperti list dan tuple, tipe data dalam set dapat bermacam-macam. Dalam sebuah set bisa jadi ada tipe data boolean, integer dan float sehingga tipe data sebuah set tidak terpaku pada satu tipe data saja. Untuk mengaksesnya, digunakan for loop, dan untuk mengecek apakah sebuah value ada di dalam set dapat menggunakan `in`.

### 1.6.7 Dictionary

Dictionary, mirip dengan list, tuple, dan set, berfungsi untuk menyimpan nilai sebuah data dalam variabel. Namun, perbedaannya, dalam dictionary data disimpan dalam pasangan pasangan yang disebut key:value. Value dalam sebuah dictionary dapat diakses dengan memanggil key-nya. Contoh pendefinisian dictionary adalah sebagai berikut :

```
dictionary = {  
    "NIM":13520000,  
    "Nama" : "contoh",  
    "IPK" : 5  
}
```

Maka jika kita menuliskan command `print(dict["NIM"])` akan keluar 13520000 sebagai value dari "NIM". Dalam sebuah dictionary, tipe data untuk value dapat berupa apa saja. Bisa string, integer, bisa juga berupa list. Dictionary adalah tipe data yang terurut, dapat diubah namun tidak memperbolehkan adanya duplikat nilai dalam satu dictionary.

### 1.6.8 If... Else

Penggunaan If adalah untuk kondisi logika matematika. Operator yang digunakan ada di subbab 1.6.3 bagian operator perbandingan. Keyword yang digunakan adalah if. Perlu diingat bahwa agar command dibawah kondisi ini berjalan, haruslah dengan indentasi yang tepat, kemudian setelah If yang diikuti kondisi, haruslah diberi tanda titik dua. Dalam if terdapat juga elif. Elif digunakan untuk menandakan "jika kondisi sebelumnya tidak benar, maka coba kondisi ini". Sama seperti if, elif juga memerlukan titik dua setelah pernyataan kondisi, dan perlu juga indentasi untuk baris setelah elif. Terakhir terdapat else. Else digunakan untuk kondisi lain selain yang disebutkan di if atau elif (jika menggunakan elif). Pada penggunaan else, tidak diperlukan lagi kondisi yang menggunakan operator perbandingan. Namun, penggunaan titik dua dan indentasi pada baris setelahnya tetap diperlukan.

Dalam penulisan kondisi setelah if atau elif, dapat berupa boolean, atau berupa penggunaan operator perbandingan seperti `==`, `<`, `>`, `<=`, `>=`, `!=`. Dapat juga digunakan operator logika seperti `and`, `or`, dan `not`. And akan melanjutkan kode jika dua kondisi dipenuhi, sedangkan or akan melanjutkan kode / mengerjakan command pada baris setelahnya jika minimal satu kondisi terpenuhi. Contoh penggunaan if :

```
if ((a>b) and ((a>0) or (b>0))):  
    print("A > B")  
elif(a==b):
```

```
print("A = B")
else:
    print("A < B")
```

Di dalam if, dapat menggunakan if lagi atau yang biasa disebut nested if. Python juga memfasilitasi *shorthand* if. *Shorthand* if ini berfungsi untuk mempersingkat penulisan if jika kita hanya memiliki satu statement. Bisa juga untuk if...else. Contoh penggunaan :

```
print("a > b") if (a>b) else print("b > a")
```

### 1.6.9 While loop

While loop akan menjalankan sejumlah pernyataan jika kondisinya true. Seperti if, while juga menggunakan operator perbandingan atau boolean setelah kata while dan menggunakan titik dua setelah penulisan kondisi. Setelah baris itu, harus ada indentasi agar dapat di compile. Pastikan pernyataan setelah while akan berlanjut dan kemudian dapat mengeluarkan program dari loop-nya. Jika tidak, bisa infinity loop dan program tidak akan berjalan dengan benar. Terdapat beberapa keyword yang dapat digunakan dalam while loop. Ada *break*, yaitu untuk keluar dari loop dan *continue* yaitu melanjutkan ke iterasi berikutnya. Berikut contoh penggunaan while loop

```
i = 0
while i < 5:
    i += 1
    if i == 3:
        continue
    print(i)
```

### 1.6.10 For loop

For dalam python bisa mengiterasi elemen-elemen yang ada dalam sebuah list, tuple, set. Bahkan for dapat juga digunakan untuk looping dalam sebuah string. Sama seperti while, terdapat break untuk keluar dari loop dan continue untuk melewati satu iterasi dan lanjut ke iterasi berikutnya. Selain untuk mengiterasi dalam sebuah list, tuple, string, atau set, for loop juga dapat digunakan dengan fungsi range(). Pada defaultnya, jika kita memasukkan range(6), maka for akan melakukan iterasi dari 0 hingga 5, dengan setiap iterasinya bertambah 1. Akan tetapi jika kita memasukkan angka lain di depan dan di belakang maka kita bisa mengubah iterasi dimulai dari angka berapa, dan kita juga bisa mengubah setiap iterasinya ditambah



berapa. Contoh range (1,10,2) maka akan dilakukan iterasi dari 1 hingga 9, dengan penambahan 2 setiap iterasinya. Contoh penggunaan for loop :

```
for x in "banana":  
    print(x)
```

Berikut contoh penggunaan range dalam for loop

```
For i in range(2, 11, 2):  
    print(i + "adalah angka genap")
```

Seperti while, indentasi dan titik dua dalam penggunaan for loop ini sangat penting karena jika tidak ada, maka program tidak dapat dijalankan.

### 1.6.11 Functions

Function adalah sebuah blok code yang akan dijalankan hanya ketika blok kode itu dipanggil. Sebuah fungsi dapat menerima data dan dapat mengembalikan data. Untuk membuat sebuah fungsi, digunakan keyword `def` yang diikuti dengan nama fungsi dan parameter. Tentu saja, titik dua dan indentasi juga penting dalam pembuatan fungsi ini. Sebuah fungsi dapat tidak memiliki parameter. Parameter sebuah fungsi dapat berupa apa saja, penggunaannya akan langsung disesuaikan. Jika sebuah fungsi memiliki 2 parameter namun saat dipanggil argumen yang dimasukkan hanya 1, maka akan keluar pesan error. Jika kita ingin fungsi mengembalikan nilai (bisa lebih dari 1), maka kita dapat menggunakan *keyword* `return` pada akhir pendefinisian fungsi. Contoh penggunaan fungsi

```
def my_function(x):  
    return (x, x+2, x+4)  
  
def print_angka(*x):  
    for i in x:  
        print(i)  
  
x, y, z = my_function(2)  
print_angka(x, y, z)
```

Pada fungsi `my_function`, terdapat data yang dikembalikan yang kemudian dimasukkan ke dalam variabel `x`, `y`, dan `z`. Pada fungsi `print_angka`, digunakan `*x` karena tidak tau ada berapa banyak parameter (tidak pasti). Digunakanlah loop untuk menuliskan semua angka yang ada dalam `x`.

### 1.6.12 Class

Python merupakan sebuah bahasa yang berorientasi objek. Class adalah pembuat objek. Untuk membuat sebuah class, keyword yang digunakan adalah `class`. Dalam python, terdapat fungsi *built-in* yaitu `__init__()` yang selalu dijalankan setiap sebuah class diinisiasi. Fungsi ini digunakan untuk *assign* nilai ke properti-properti dalam objek, atau operasi lain yang dibutuhkan saat objek ini dibentuk. Selain `__init__()`, kita dapat juga mendefinisikan methods. Methods adalah fungsi yang dimiliki oleh objek. Parameter dalam `__init__()` minimal berisikan `self` (dapat diganti, tidak wajib `self`) yang berguna untuk merujuk dirinya sendiri. Contoh pembuatan class

```
Class student :  
    def __init__(self, name, nim):  
        self.name = name  
        self.nim = nim  
    def kenalan(self):  
        print("Halo! Nama saya " + self.name)  
Student1 = student("Budi", 13520000)  
Student1.kenalan()
```

Setelah pembuatan class, kita dapat mengubah properti-properti dari sebuah objek. Misalnya kita ingin mengubah nim Student1 maka kita bisa tinggal menuliskan

```
Student1.nim = 13520001
```

Begitu juga untuk properti yang lainnya. Kemudian jika kita ingin menghapus properti sebuah objek atau menghapus objek itu sendiri, kita dapat menggunakan *keyword* `del`. Seperti biasanya, titik dua dan indentasi sangat penting dan dibutuhkan pada pembuatan class ini.

## BAB 2 : Hasil FA dan CFG

### 2.1 Hasil FA

Finite Automata yang sudah dibuat terkait pengecekan nama variabel adalah sebagai berikut

$$A = (Q, \Sigma, q, F, \delta)$$

State (Q) = q0, q1, q2

Input ( $\Sigma$ ) = seluruh huruf kapital alfabet (A..Z), huruf kecil alfabet (a..z), digit angka (0..9), dan garis bawah ( $\_$ )

Initial state (q) = q0

Final state (F) = q2

Fungsi transisi ( $\delta$ ):

	A..Z	a..z	0..9	_
-> q0	q1	q1	-	q1
q1	q2	q2	q2	q2
*q2	-	-	-	-

### 2.2 Hasil CFG

Berikut hasil Context-Free Grammar yang sudah kami buat terkait compiler python

$$G = (N, T, P, S)$$

**Nonterminal / Variables (N)**

SQ_COMMENT	ELIF	WITH	BITWISE_NOT	LCB
DQ_COMMENT	ELSE	IN	AS	RCB
BOOLEAN	FOR	NOT	POWER	DOT
AUG_ASSIGN	FROM	PLUS	NONE	COMMA
LOGICAL_OPR	IMPORT	MINUS	COMPARISON	SEMICOLON
BREAK	PASS	ARROW	ASSIGN	COLON
CLASS	RANGE	NUMBER	LP	SQUOTE

CONTINUE	RAISE	IDENTIFIER	RP	DQUOTE
DEF	RETURN	BINARY_OPR	LSB	DMARK
IF	WHILE	BITWISE_OPR	RSB	STAR

### Terminal Symbol

SQ_COMMENT	ELSE	NOT	BITWISE_NOT	RCB
DQ_COMMENT	FOR	PLUS	LOGICAL_OPR	DOT
BOOLEAN	FROM	MINUS	POWER	COMMA
NONE	IMPORT	ARROW	AUG_ASSIGN	SEMICOLON
AS	PASS	NUMBER	COMPARISON	COLON
BREAK	RANGE	IDENTIFIER	ASSIGN	SQUOTE
CLASS	RAISE	MINUS	LP	DQUOTE
CONTINUE	RETURN	PLUS	RP	QMARK
DEF	WHILE	STAR	LSB	STAR
IF	WITH	BINARY_OPR	RSB	STRING
ELIF	IN	BITWISE_OPR	LCB	

### Production

Production	Hasil
ARGS	FUNC   BOOLEAN   NUMBER   IDENTIFIER   INDEXING   STRING   ARITH_EXPR   STR_ARGS   ITERABLE   IDENTIFIER ASSIGN ATOM   IDENTIFIER ASSIGN VARIABLE   IDENTIFIER ASSIGN STRING   ARGS COMMA ARGS
ALLOWED_SIGN	PLUS   MINUS   BITWISE_NOT
NUMBER	ALLOWED_SIGN NUMBER
ALLOWED_OPR	BINARY_OPR   BITWISE_OPR
ARITH_NUM	NUMBER   VARIABLE   ALLOWED_SIGN ARITH_NUM
ARITH_EXPR	ARITH_NUM   ARITH_EXPR POWER   LP ARITH_EXPR

	RP   ARITH_EXPR ALLOWED_OPR ARITH_CNT
ARITH_CNT	ARITH_EXPR   ALLOWED_SIGN ARITH_EXPR   ARITH_EXPR ALLOWED_OPR ARITH_CNT
STRING_OPR	PLUS   STAR
STR_ATOM	STRING   VARIABLE   FUNC
STR_ARGS	LP STR_ARGS RP   STR_ATOM STRING_OPR STR_ATOM   STR_ATOM STRING_OPR STR_CONT
STR_CONT	STR_ARGS   STR_ATOM STRING_OPR STR_CONT
S	FOR FOR_STMT   IMPORT_STMT   DEF FUNCDEF   FUNC   EXPR_STMT   WITH WITH_STMT   BREAK   CONTINUE   RETURN_STMT   RETURN CMP_TEST LOGICAL_OPR CMP_TEST   RAISE_STMT   WHILE WHILE_STMT   IF IF_STMT   ELIF ELIF_STMT   IF NOT IF_STMT   ELIF NOT ELIF_STMT   ELSE COLON   CLASSDEF   ARRAYDEF   TUPLEDEF   IDENTIFIER ASSIGN DICT   PASS
FOR_STMT	IN_STMT COLON   IDENTIFIER IN RANGE_STMT COLON
IN_STMT	IDENTIFIER IN VARIABLE   IDENTIFIER IN ITERABLE   IDENTIFIER IN IN_STMT
RANGE_STMT	RANGE LP RANGE_CNT   NUMBER RP   NUMBER COMMA RANGE_CNT   VARIABLE RP   VARIABLE COMMA RANGE_CNT   FUNC RP   FUNC COMMA RANGE_CNT   ARITH_EXPR RP   ARITH_EXPR COMMA RANGE_CNT
IMPORT_STMT	IMPORT_T   IMPORT_T AS IDENTIFIER   FROM_T IMPORT_T   FROM_T IMPORT_T AS IDENTIFIER
FROM_T	FROM IDENTIFIER
IMPORT_T	IMPORT STAR   IMPORT IDENTIFIER   IMPORT IDENTIFIER COMMA IMPORT_CNT
IMPORT_CNT	IDENTIFIER   IDENTIFIER COMMA IMPORT_CNT
TEST	ATOM   VARIABLE   ITERABLE   ARITH_EXPR   LP TEST RP   STRING IN ITERABLE   STRING IN VARIABLE

	ATOM IN VARIABLE   VARIABLE IN VARIABLE   ATOM IN ITERABLE   VARIABLE IN ITERABLE   TEST LOGICAL_OPR CMP_TEST   CMP_TEST LOGICAL_OPR CMP_TEST   CMP_TEST BITWISE_OPR CMP_TEST   VARIABLE LOGICAL_OPR CMP_TEST   ATOM LOGICAL_OPR CMP_TEST   NOT TEST
CMP_TEST	LP CMP_TEST RP   FUNC COMPARISON NUMBER   IDENTIFIER COMPARISON FUNC   IDENTIFIER COMPARISON IDENTIFIER   IDENTIFIER COMPARISON STRING   NUMBER COMPARISON FUNC   NUMBER COMPARISON ATOM   VARIABLE COMPARISON FUNC   VARIABLE COMPARISON ATOM   ARITH_EXPR COMPARISON ARITH_NUM   ARITH_EXPR COMPARISON ARITH_EXPR
FUNCDEF	IDENTIFIER PARAMS COLON   IDENTIFIER LP ARROW_STMT
ARROW_STMT	IDENTIFIER COLON ARROW_STMT1
ARROW_STMT1	IDENTIFIER RP ARROW_STMT2
ARROW_STMT2	ARROW IDENTIFIER COLON
FUNC	IDENTIFIER LP ARGS RP   IDENTIFIER LP STR_ARGS RP   IDENTIFIER LP STRING STRING_OPR STR_CONT RP   IDENTIFIER PARAMS
PARAMS	LP RP   LP ARGS RP   LP IDENTIFIER COMMA ARGS RP   LP IDENTIFIER COLON IDENTIFIER RP
IDX_KEY	LSB NUMBER RSB   LSB IDENTIFIER RSB   LSB ARITH_EXPR RSB   LSB INDEXING RSB   LSB SQUOTE IDENTIFIER SQUOTE RSB   LSB DQUOTE IDENTIFIER DQUOTE RSB
INDEXING	INDEXING DOT IDENTIFIER   IDENTIFIER IDX_KEY
ALLOWED_ASSIGN	IDENTIFIER ASSIGN   VARIABLE ASSIGN
ALLOWED_ASSIGN1	IDENTIFIER AUG_ASSIGN   VARIABLE AUG_ASSIGN
EXPR_STMT	ALLOWED_ASSIGN1 ARITH_EXPR   ALLOWED_ASSIGN1 VARIABLE   ALLOWED_ASSIGN ATOM   ALLOWED_ASSIGN VARIABLE   ALLOWED_ASSIGN ITERABLE

WITH_STMT	FUNC AS IDENTIFIER COLON   IDENTIFIER METHOD AS IDENTIFIER COLON
RETURN_STMT	RETURN ATOM   RETURN VARIABLE   RETURN STRING   RETURN TEST   RETURN IN_STMT
RAISE_STMT	RAISE   RAISE TEST   RAISE IDENTIFIER   RAISE TEST FROM_RAISE
FROM_RAISE	FROM_STR TEST
WHILE_STMT	TEST COLON   VARIABLE COLON   CMP_TEST COLON   LP TEST RP COLON
IF_STMT	TEST COLON   CMP_TEST COLON   INDEXING METHOD COLON
ELIF_STMT	TEST COLON   CMP_TEST COLON   INDEXING METHOD COLON
METHOD	DOT FUNC   DOT IDENTIFIER   DOT METHOD DOT METHOD_CONT
METHOD_CONT	DOT FUNC   DOT IDENTIFIER
CLASSDEF	CLASS_T COLON   CLASS_T PARAMS COLON
CLASS_T	CLASS IDENTIFIER
ITERABLE	DICT   ARRAY   TUPLE   STRING
ARRAYDEF	IDENTIFIER ASSIGN ARRAY
ARRAY	LSB RSB   LSB ARRAYCONT RSB   LSB IDENTIFIER IN_STMT RSB   LSB NUMBER FOR ARRAYFOR1
ARRAYFOR1	IDENTIFIER IN RANGE_STMT RSB
ARRAYCONT	ATOM   VARIABLE   ARGS   ARGS COMMA ARRAYCONT   ATOM COMMA ARRAYCONT   VARIABLE COMMA ARRAYCONT
TUPLEDEF	IDENTIFIER ASSIGN TUPLE
TUPLE	LP RP   LP TUPLECONT RP
TUPLECONT	ATOM   VARIABLE   ARGS   ATOM COMMA TUPLECONT   VARIABLE COMMA TUPLECONT   ARGS COMMA

	TUPLECONT
DICT	LCB DICT1
DICT1	RCB   DICTKEY COLON DICTVAL COMMA   DICTKEY COLON DICTVAL RCB
DICTKEY	NUMBER   STRING   IDENTIFIER
DICTVAL	ARGS   ARGS COMMA   ARGS COMMA DICTCONT
DICTCONT	DICTKEY COLON DICTVAL   DICTKEY COLON DICTVAL COMMA DICTCONT
ATOM	NUMBER   ARITH_EXPR   NONE   BOOLEAN   IDENTIFIER   INDEXING
VARIABLE	IDENTIFIER   FUNC   INDEXING   INDEXING DOT IDENTIFIER   INDEXING METHOD
IDENTIFIER	IDENTIFIER DOT IDENTIFIER   IDENTIFIER METHOD



## BAB 3 : Implementasi dan Pengujian Program

### 3.1. Spesifikasi Teknis Program

#### 3.1.1 Struktur Folder

Di dalam folder python-compiler, terdapat:

1. database: menyimpan daftar context-free grammar (CFG.txt), daftar chomsky normal form (CNF.txt) hasil konversi, dan daftar rule regex untuk token (token\_db.py).
2. pycomp: folder utama, menyimpan source code konversi CFG ke CNF (cfg2cnf.py), parser (cyk\_parser.py), kelas error (error.py), kelas lexer (lexer.py), dan file pendukung (utils.py).
3. stash: folder berisi tester selama pengerjaan (tidak berkaitan dengan program utama, hanya sebagai folder “penyimpan”).
4. testcase: folder berisi kode yang lolos uji dari compiler yang dibuat.

#### 3.1.2 Struktur Data yang Dibuat

##### 3.1.2.1 Lexer (lexer.py)

Deskripsi: Lexer mengubah teks dalam file menjadi token-token sebelum dilakukan proses parsing. Pada lexer, program juga membuang atau menyaring token-token yang tidak perlu diproses, misalnya membuang komentar pada program sesuai syntax Python.

Selain itu, di dalam lexer juga terdapat pengecekan untuk multiline comment. Apabila multiline comment diawali dengan triple single quote dan diakhiri dengan triple double quote, akan terjadi error.

```
1 def get_rule_category(rule: dict) -> str:
2     ''' Get rule category in string. This category is also a key for its corresponding dictionary.
3     Input(dict) = rule
4     Output(str) = category of the rule
5
6     Example:
7     Input  = {'producer': 'N', 'product': ['people']}
8     Output = 'terminal'
9
10    rule_product = rule[PRODUCT_KEY]
11    if len(rule_product) > 0:
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler> py parserprogram.py testcase/TC3.txt
Checking testcase/TC3.txt...
In file testcase/TC3.txt, line 2.
    ''' Get rule category in string. This category is also a key for its corresponding dictionary.
    ^
Syntax Error: EOF while scanning triple-quoted string literal
PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler>
```

Kemudian, terdapat list representasi stack untuk membaca tanda buka / tutup kurung. Jika tidak match, akan muncul error.

```
1 | input1 = input(Apakah, kamu, lelah, "?", "ya/tidak")
2 | input2 = input(Apakah, kamu, deadliner, "?", "ya/tidak"))
3 |
```

---

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler> py parserprogram.py testcase/TC6.txt
Checking testcase/TC6.txt...
In file testcase/TC6.txt, line 2.
input2 = input(Apakah, kamu, deadliner, "?", "ya/tidak"))
                                     ^
Syntax Error: Did you forget to open the bracket? Unmatched ')'
PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler> █
```

Selengkapnya akan ditambahkan di dalam bagian berikutnya.

Berikut properti dan method kelas Lexer:

a. Properti:

- i. filename (string): menyimpan nama file yang sedang dibaca, tujuannya agar saat terjadi error, program dapat menunjukkan posisi error.
- ii. text (string): isi tulisan dari file yang dibaca, tujuannya agar dapat memunculkan teks ketika terjadi error.
- iii. position (Position): posisi awal saat belum membaca apa-apa, akan berubah ketika program sudah membaca dan melakukan tokenisasi.

b. Method

- i. tokenize: method tokenize menerima satu argumen yaitu pola atau rule regular expression sebagai aturan untuk melakukan tokenisasi. Dalam method ini, program diproses sesuai deskripsi lexer.

### 3.1.2.2 Token (lexer.py)

Deskripsi: Token adalah representasi teks / kalimat setelah dibaca. Daftar token dapat dilihat pada bagian 3.1.4.

a. Properti:

- i. tag (string): “penanda” token dengan tipe data string, contoh, karakter ‘(’ memiliki token ‘LP’ artinya left parentheses.
- ii. value (string): kata atau karakter asli dari token, dengan contoh yang sama, berarti token ‘LP’ memiliki value ‘(’.
- iii. pos\_start (Position) : posisi karakter dalam teks, tujuannya ketika error kita bisa langsung mengetahui posisinya.
- iv. pos\_end (Position): posisi akhir karakter dalam teks, misal kata “range” memiliki pos\_start pada karakter ‘r’ dan pos\_end pada karakter ‘e’.

### 3.1.2.3 Position (lexer.py)

Deskripsi: Position adalah posisi token seperti yang sudah dicontohkan pada bagian sebelumnya. Dalam program ini, Position menjadi bagian dari Token.

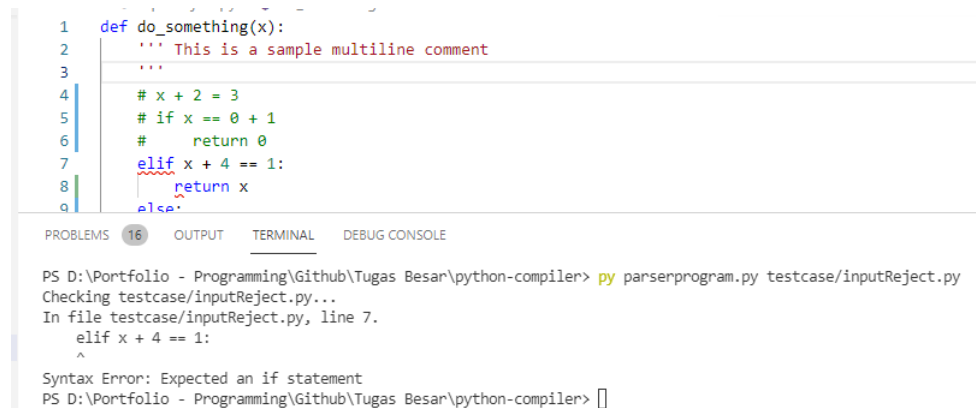
a. Properti:

- i. index (integer): indeks pada teks. Tidak ter-reset ketika newline.
- ii. line (integer): letak token pada file (baris ke-n). Line mulai dari 1.
- iii. col (kolom): kolom kata per baris. Misalkan pada kalimat “if True:”, karakter ‘T’ memiliki kolom ke-3 (mulai dari 0). Ter-reset ketika newline.
- iv. filename (string): berisi nama file yang akan dibaca.
- v. filetext (string): berisi teks yang dibaca dari filename.

### 3.1.2.4 Parser (cyk\_parser.py)

Deskripsi: Parser memproses teks sesuai dengan aturan grammar input. Masukan grammar ke parser haruslah sudah berbentuk Chomsky Normal Form. Parser menggunakan algoritma CYK (Cocke--Younger--Kasami), yaitu dengan membuat tabel, melakukan cartesian product, dan prosedur-prosedur lain sesuai dengan referensi (Ullman, et al. 2006).

Pada parser juga terdapat pengecekan blok if-else seperti berikut.



```
1 def do_something(x):
2     ''' This is a sample multiline comment
3     '''
4     # x + 2 = 3
5     # if x == 0 + 1
6     #     return 0
7     elif x + 4 == 1:
8         return x
9     else:
```

PROBLEMS 16 OUTPUT TERMINAL DEBUG CONSOLE

PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler> py parserprogram.py testcase/inputReject.py  
Checking testcase/inputReject.py...  
In file testcase/inputReject.py, line 7.  
 elif x + 4 == 1:  
 ^  
Syntax Error: Expected an if statement  
PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler>

a. Properti:

- ii. filename: nama file yang akan diproses
- iii. cnf\_file: nama file berisi grammar CNF. Bentuk grammar di dalamnya haruslah berbentuk seperti  $S \rightarrow A \mid BC$ .
- iv. text: teks di dalam file yang akan diproses.

c. Method

- i. parse\_cyk: melakukan implementasi algoritma CYK per baris. Method parse\_cyk menerima argumen berisi baris yang sudah ditokenisasi dan grammar CNF dalam representasi list (bukan dictionary).

- ii. `parse_text`: memanggil method `parse_cyk` dan melakukan pengecekan pada baris. Pada method ini juga terdapat pengecekan blok if-else seperti yang sudah ada di deskripsi.

#### 3.1.2.5 Error (error.py)

Deskripsi: Error merupakan kelas yang akan mengeluarkan error pada pengguna. Terbagi menjadi dua subkelas yaitu `illegal character error` dan `invalid syntax error`.

- a. Properti
  - i. `name (string)`: nama atau jenis error.
  - ii. `pos_start (Position)`: posisi awal error.
  - iii. `pos_end (Position)`: posisi akhir error.
  - iv. `message (string)`: pesan error (jika ada).
  - v. `txtline (string)`: string baris yang terdapat error.
- b. Method
  - i. `print_error`: memunculkan pesan error, jenis error, letak error, dan “kalimat” atau baris tempat terjadi error.
- c. Subkelas
  - i. `IllegalCharacterError`: ketika terdapat karakter di luar ASCII atau di luar cakupan tugas besar, misalnya decorator ‘@’.
  - ii. `InvalidSyntaxError`: error lain yang tidak termasuk karakter ilegal, misalnya tanda kurung yang tidak berpasangan.

#### 3.1.2.6 Variable Name Checker (var.py)

Deskripsi: Implementasi pengecekan nama variabel menggunakan Finite Automata. Nama variabel name diterima jika `VarFA.check(name) = True`, dan tidak diterima/invalid jika `VarFA.check(name) = False`.

- a. Method:
  - i. `check(name)` : Program utama, yang memasukkan nama variabel ke initial state `q0`.
  - ii. `q0(name)` : Initial state, melanjutkan ke state `q1` jika huruf pertama adalah huruf kapital (A-Z), huruf kecil (a-z), atau garis bawah (`_`). Jika tidak, langsung mengembalikan `False`.
  - iii. `q1(name)` : State yang menerima huruf kedua dan seterusnya adalah huruf kapital (A-Z), huruf kecil (a-z), digit angka (0-9), atau garis bawah (`_`). Jika name belum kosong, pembacaan akan dilanjutkan ke diri sendiri (`q1`). Namun jika name sudah kosong (tidak membaca apa-apa / “”), melanjutkan ke `q2`.
  - iv. `q2(name)` : Final state, mengembalikan `True` jika name sudah mencapai state ini.

### 3.1.3 Header Fungsi

Pada bagian ini, akan dituliskan fungsi-fungsi yang utama saja. Fungsi-fungsi antara tidak ditulis untuk mengurangi clutter.

#### 3.1.3.1 File cfg2cnf.py

- a. function `grammar_to_list(filename: string) → list`

Mengembalikan grammar dalam bentuk list. Di dalam file grammar, aturan produksi berbentuk, seperti

$$S \rightarrow A \mid C D \mid EF$$

$$A \rightarrow E$$

Huruf C dan D harus ditulis terpisah untuk menunjukkan bahwa mereka adalah dua variabel berbeda. Hasil list yang akan dikembalikan adalah sebagai berikut.

`[[‘S’, ‘A’], [‘S’, ‘C’, ‘D’], [‘S’, ‘EF’], [‘A’, ‘E’], dst..]`

- b. function `grammar_to_dict(filename: string) → dictionary`

Sama seperti fungsi sebelumnya, hanya saja mengembalikan dalam bentuk dictionary atau hash table. Dengan contoh yang sama, fungsi akan mengembalikan

`{‘S’: [[‘S’, ‘A’], [‘S’, ‘C’, ‘D’], [‘S’, ‘EF’]],  
‘A’: [[‘A’, ‘E’]], dst...}`

- c. procedure `handle_unit_procedure(input/output rule: dictionary)`

I.S. Bisa saja terdapat unit production di dalam grammar, program akan mensubstitusi dan mengulang sampai tidak ada lagi unit production.

F.S. Semua unit production di dalam dictionary grammar sudah tersubstitusi.

- d. function `CFG2CNF(rule: dictionary) → dictionary`

Mengembalikan dictionary dengan grammar sesuai aturan Chomsky, yaitu hanya memperbolehkan bentuk  $A \rightarrow B C$  dan  $A \rightarrow a$ .

Dalam program ini, **simbol terminal ditandai** dengan adanya **tanda petik satu**. Misalkan terdapat token “IF”, maka produksi terminalnya akan berbentuk seperti berikut.

$$IF \rightarrow \text{‘IF’}$$

Jika bukan terminal, bentuknya akan seperti berikut.

### IF\_STATEMENT → IF TEST COLON

Dengan kata lain, “IF” **tanpa tanda petik satu** merupakan variabel (atau nonterminal).

- e. procedure run\_converter(input filein: string, output fileout: string)

I.S. Tidak terdapat file output berisi grammar hasil konversi

F.S. Terdapat file baru yang sudah berisi grammar hasil konversi (CNF) pada tree folder.

#### 3.1.4 Daftar Token

LN_COMMENT	PASS	STAR
SQ_COMMENT	RANGE	BITWISE_NOT
DQ_COMMENT	RAISE	BITWISE_OPR
STRING	RETURN	ASSIGN
BOOLEAN	WHILE	LP
NONE	WITH	RP
AS	NOT	LSB
BREAK	IN	RSB
CLASS	LOGICAL_OPR	LCB
CONTINUE	COMPARISON	RCB
DEF	NUMBER	DOT
IF	IDENTIFIER	COMMA
ELIF	ARROW	SEMICOLON
ELSE	AUG_ASSIGN	COLON
FOR	PLUS	DQUOTE
FROM	MINUS	SQUOTE
IMPORT	BINARY_OPR	QMARK

## 3.2. Screenshot contoh kasus

### 3.2.0. Contoh Kasus pada Spek Tugas

```
python-compiler> python3 parserprogram.py testcase/inputAcc.py
Checking testcase/inputAcc.py...
Yay, your program is accepted!
python-compiler> █
```

Sintaks pada inputAcc.py sesuai dengan aturan sintaks pada Python, sehingga program diterima.

```
python-compiler> python3 parserprogram.py testcase/inputReject.py
Checking testcase/inputReject.py...
In file testcase/inputReject.py, line: 4
    x + 2 = 3
Syntax Error Found!
python-compiler> █
```

Pada inputReject.py, terdapat baris  $x + 2 = 3$  di line 4. Jika dilakukan pembacaan token pada baris tersebut dengan lexer, akan menjadi ARITH\_EXPR ASSIGN ARITH\_NUM, atau ARITH NUM ALLOWED\_OPR ARITH\_CNT. Pengolahan sintaks dilakukan dengan menggunakan CFG berikut

```
S → EXPR_STMT
EXPR_STMT → ALLOWED_ASSIGN ARITH_EXPR
ALLOWED_ASSIGN → VARIABLE ASSIGN
ARITH_EXPR → ARITH_NUM
ARITH_EXPR → ARITH_EXPR ALLOWED_OPR ARITH_CNT
```

Karena token pada baris tersebut tidak diterima oleh CFG, maka akan mengeluarkan pesan Syntax Error.

#### 3.2.1. Test Case 1

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC1.txt>

Hasil eksekusi program:

```
python-compiler> python3 parserprogram.py testcase/TC1.txt
Checking testcase/TC1.txt...
Yay, your program is accepted!
python-compiler> █
```

Sintaks pada TC1.txt sesuai dengan aturan sintaks pada Python, sehingga program diterima.

### 3.2.2 Test Case 2

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC2.txt>

```
python-compiler> python3 parserprogram.py testcase/TC2.txt
Checking testcase/TC2.txt...
In file testcase/TC2.txt, line: 18
    for c in text
Syntax Error Found!
python-compiler> █
```

Pada TC2.txt line 18, terdapat baris `for c in text`. Karena tidak adanya titik dua (:) setelah `for` loop, token yang diterima oleh lexer berupa `FOR IDENTIFIER IN VARIABLE`. CFG yang mem-parse baris tersebut adalah sebagai berikut.

```
S → FOR FOR_STMT
FOR_STMT → IN_STMT COLON
IN_STMT → IDENTIFIER IN VARIABLE
```

Dari CFG tersebut, tidak terdapat token `COLON` (titik dua/':'), sintaks menjadi tidak valid, sehingga mengeluarkan pesan `Syntax Error`.

### 3.2.3 Test Case 3

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC3.txt>

```
python-compiler> python3 parserprogram.py testcase/TC3.txt
Checking testcase/TC3.txt...
In file testcase/TC3.txt, line: 17
    elif len(rule_product) = 2:
Syntax Error Found!
python-compiler> █
```

Pada TC3.txt line 18, terdapat `elif len(rule_product) = 2:`, yang dapat diubah menjadi token `ELIF FUNC ASSIGN NUMBER COLON`. CFG yang mengolah sintaks pada baris tersebut dapat dijabarkan sebagai berikut.

```
S → ELIF ELIF_STMT
ELIF_STMT → CMP_TEST COLON
CMP_TEST → VARIABLE COMPARISON ATOM
VARIABLE → FUNC
ATOM → NUMBER
```



Karena token pada baris tersebut tidak diterima oleh CFG, maka akan mengeluarkan pesan Syntax Error.

### 3.2.4 Test Case 4

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC4.txt>

```
python-compiler> python3 parserprogram.py testcase/TC4.txt
Checking testcase/TC4.txt...
Yay, your program is accepted!
python-compiler> █
```

Sintaks pada TC4.txt sesuai dengan aturan sintaks pada Python, sehingga program diterima.

### 3.2.5 Test Case 5

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC5.txt>

```
python-compiler> python3 parserprogram.py testcase/TC5.txt
Checking testcase/TC5.txt...
Yay, your program is accepted!
python-compiler> █
```

Sintaks pada TC5.txt sesuai dengan aturan sintaks pada Python, sehingga program diterima.

### 3.2.6 Test Case 6

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC6.txt>

```
python-compiler> python3 parserprogram.py testcase/TC6.txt
Checking testcase/TC6.txt...
In file testcase/TC6.txt, line 2.
input2 = input(Apakah, kamu, deadliner, "?", "ya/tidak"))
                                     ^
Syntax Error: Did you forget to open the bracket? Unmatched ')'
python-compiler> █
```

Pada TC6.txt line 2, baris tersebut kekurangan kurung buka, sehingga satu token LP terbaca saat pembacaan token menggunakan lexer. Dengan pengecekan bracket, dapat terlihat error unmatched brackets.

### 3.2.7 Test Case 7

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC7.txt>

```
python-compiler> python3 parserprogram.py testcase/TC7.txt
Checking testcase/TC7.txt...
Yay, your program is accepted!
python-compiler> █
```

Sintaks pada TC7.txt sesuai dengan aturan sintaks pada Python, sehingga program diterima.

### 3.2.8 Test Case 8

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC8.txt>

```
python-compiler> python3 parserprogram.py testcase/TC8.txt
Checking testcase/TC8.txt...
In file testcase/TC8.txt, line 7.
2abc = 123
  ^
Syntax Error: Invalid variable name
python-compiler> █
```

Pada TC8.txt baris 4, terdapat `2abc = 123`. Dalam Python, nama variabel yang valid terdiri dari huruf, angka, dan garis bawah (`_`), dan diawali dengan huruf atau garis bawah. Variabel `2abc` bukan variabel yang valid, sehingga mengeluarkan Syntax Error.

### 3.2.9 Test Case 9

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC9.txt>

```
python-compiler> python3 parserprogram.py testcase/TC9.txt
Checking testcase/TC9.txt...
Yay, your program is accepted!
python-compiler> █
```

Sintaks pada TC9.txt sesuai dengan aturan sintaks pada Python, sehingga program diterima.

### 3.2.10 Test Case 10

Link: <https://github.com/khelli07/python-compiler/raw/main/testcase/TC10.txt>

```
python-compiler> python3 parserprogram.py testcase/TC10.txt
Checking testcase/TC10.txt...
In file testcase/TC10.txt, line 16.
x = 5 * 2 + (5 + 2 * (3 + 4)
  ^
Syntax Error: Did you forget to close the bracket? Unmatched '('
python-compiler> █
```

Pada TC10.txt line 4, terdapat ekspresi assignment  $x = 5 * 2 + (5 + 2 * (3 + 4))$ . Baris ini kekurangan kurung tutup, sehingga satu token RP tidak dapat terbaca saat pembacaan token menggunakan lexer. Dengan pengecekan bracket, dapat terlihat error unmatched brackets.

### 3.2.11 Test tambahan lain

```
testcase > TC7.txt
1 elif 1 == 2:
2     return a
3 a = [1,2,3]
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler> py parser.py  
Checking testcase/TC7.txt...  
In file testcase/TC7.txt, line 13.  
elif 1 == 2:  
^  
Syntax Error: Expected an if statement

Compiler menolak jika program masuk dalam blok elif atau else tanpa didahului statement if. Namun, program tetap menerima jika blok tersebut berbentuk comment.

```
testcase > TC7.txt
1 # elif 1 == 2:
2 #     return a
3 a = [1,2,3]
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler> py parser.py  
Checking testcase/TC7.txt...  
Yay, your program is accepted!  
PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler>

Selain itu, di compiler menerima if statement yang berbentuk satu baris seperti berikut.

```
10
11 if {1: 2}:
12     print("Hello + 123asd")
13
14 isApple = 'Yes' if fruit == 'Apple' else 'No'
15 print("tes" + "True")
16 x = 5 * 2 + (5 + 2 * (3 + 4))
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler> py parser.py  
Checking testcase/TC10.txt...  
Yay, your program is accepted!  
PS D:\Portfolio - Programming\Github\Tugas Besar\python-compiler>

### 3.3. Link Repository Github

<https://github.com/khelli07/python-compiler>

## BAB 4 : Pembagian Tugas

Nama	Tugas
Alifia Rahmah	Mencari testcase, melakukan testing, dan laporan bagian Screenshot contoh kasus. hasil FA
Hilda Carissa Widelia	Mencari testcase, melakukan testing, dan laporan bagian teori dasar, hasil CFG
Maria Khelli	Lexer, parser, mencari CFG yang cocok antar dengan lexer dan token, converter CFG to CNF, error, utils dan membuat laporan bagian implementasi program.

## Daftar Referensi

Python (bahasa pemrograman) - Wikipedia bahasa Indonesia, ensiklopedia bebas,

[https://id.wikipedia.org/wiki/Python\\_\(bahasa\\_pemrograman\)](https://id.wikipedia.org/wiki/Python_(bahasa_pemrograman))

Teknik Kompilasi : PERBEDAAN DFA dan NFA, Bina Nusantara University

<https://socs.binus.ac.id/2019/12/21/teknik-kompilasi-perbedaan-dfa-dan-nfa/>

Designing Deterministic Finite Automata Set-1, Geeksforgeeks

<https://www.geeksforgeeks.org/designing-deterministic-finite-automata-set-1/>

Finite Automata, Bina Nusantara University

<http://library.binus.ac.id/eColls/eThesiscoll/Bab2/2011-2-00004-MTIF%20Bab2001.pdf>

Context-Free Grammar Introduction, Tutorialspoint

[https://www.tutorialspoint.com/automata\\_theory/context\\_free\\_grammar\\_introduction.htm](https://www.tutorialspoint.com/automata_theory/context_free_grammar_introduction.htm)

Slide-slide perkuliahan IF2124 Teori Bahasa Formal dan Otomata

Automata Chomsky's Normal Form (CNF) - Javatpoint,

<https://www.javatpoint.com/automata-chomskys-normal-form>

CYK algorithm - Wikipedia, the free encyclopedia,

[https://en.wikipedia.org/wiki/CYK\\_algorithm](https://en.wikipedia.org/wiki/CYK_algorithm)

Python Syntax - W3Schools Online Web Tutorials,

[https://www.w3schools.com/python/python\\_syntax.asp](https://www.w3schools.com/python/python_syntax.asp)