

LAPORAN TUGAS BESAR
IF2124 Teori Bahasa Formal dan Otomata
Compiler Bahasa Python



Disusun oleh:

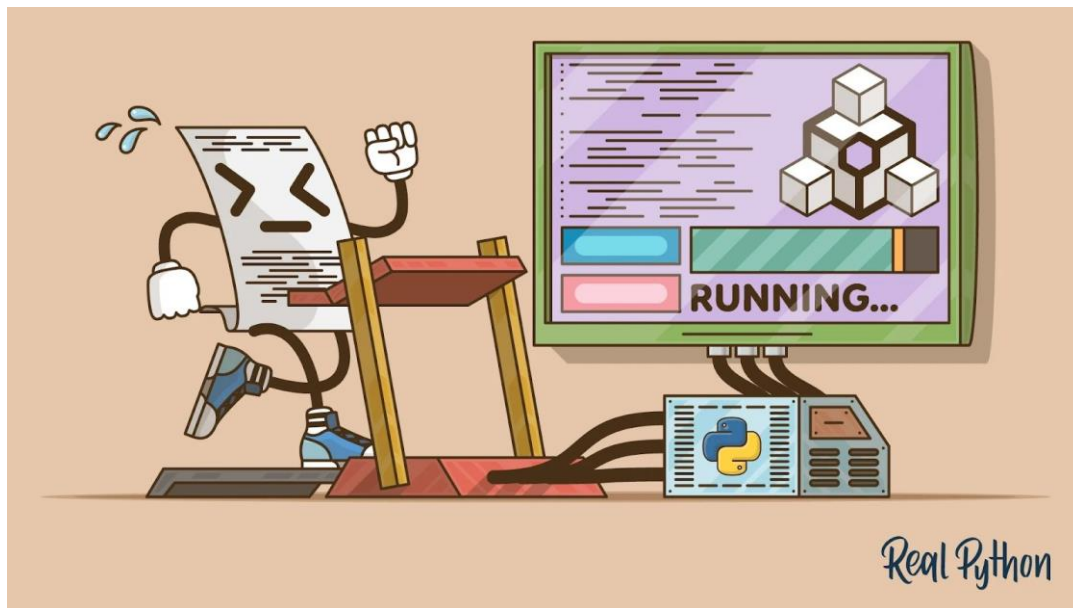
1. Johannes Winson Sukiatmodjo (13520123)
2. Ignasius Ferry Priguna (13520126)
3. Nelsen Putra (13520130)

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2021

DAFTAR ISI

Daftar Isi	2
Deskripsi Masalah	3
Teori Dasar	5
Finite Automata	5
Context-Free Grammar	6
Syntax Python	8
Hasil	18
Finite Automata	18
Context-Free Grammar	18
Implementasi	25
CFG2CNF.py	25
helper.py	25
cyk.py	26
Pengujian	27
testcase1.py	27
testcase2.py	28
testcase3.py	29
testcase4.txt	30
Kesimpulan dan Saran	31
Kesimpulan	31
Saran	32
Lampiran	33
Pembagian Tugas	33
Link Repository	33
Referensi	34

DESKRIPSI MASALAH



Sumber: <https://realpython.com/>

Python adalah bahasa *interpreter* tingkat tinggi (*high-level*), dan juga *general-purpose*. Python diciptakan oleh Guido van Rossum dan dirilis pertama kali pada tahun 1991. Filosofi desain pemrograman Python mengutamakan *code readability* dengan penggunaan *whitespace*-nya. Python adalah bahasa multiparadigma karena mengimplementasi paradigma fungsional, imperatif, berorientasi objek, dan reflektif.

Dalam proses pembuatan program dari sebuah bahasa menjadi instruksi yang dapat dieksekusi oleh mesin, terdapat pemeriksaan sintaks atau kompilasi bahasa yang dibuat oleh programmer. Kompilasi ini bertujuan untuk memastikan instruksi yang dibuat oleh programmer mengikuti aturan yang sudah ditentukan oleh bahasa tersebut. Baik bahasa berjenis *interpreter* maupun *compiler*, keduanya pasti melakukan pemeriksaan sintaks. Perbedaannya terletak pada apa yang dilakukan setelah proses pemeriksaan (kompilasi/*compile*) tersebut selesai dilakukan.

Dibutuhkan *grammar* bahasa dan algoritma *parser* untuk melakukan kompilasi. Sudah sangat banyak *grammar* dan algoritma yang dikembangkan untuk menghasilkan *compiler* dengan performa yang tinggi. Terdapat CFG, CNF^e , CNF^{+e} , 2NF, 2LF, dll untuk *grammar* yang dapat digunakan, dan terdapat LL(0), LL(1), CYK, Earley's Algorithm, LALR, GLR, Shift-reduce, SLR, LR(1), dll untuk algoritma yang dapat digunakan untuk melakukan *parsing*.

Pada tugas besar ini, kami diminta untuk mengimplementasikan *compiler* untuk Python untuk *statement-statement* dan sintaks-sintaks bawaan Python. *Compiler* Python dibuat dengan mengaplikasikan konsep *Context-Free Grammar* (CFG) untuk pengerjaan compiler yang

mengevaluasi syntax program, dan konsep *Finite Automata* (FA) untuk pengaturan nama variabel dalam program.

Dalam implementasinya, kami memilih untuk menggunakan algoritma CYK (Cocke-Younger-Kasami). Algoritma CYK harus menggunakan *grammar* CNF (*Chomsky Normal Form*) sebagai *grammar* masukannya. Oleh karena itu, untuk menggunakannya, dibuat terlebih dahulu *grammar* dalam CFG yang kemudian dikonversikan ke dalam *grammar* CNF agar dapat digunakan sebagai masukan pada algoritma CYK.

TEORI DASAR

1. 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 simbolnya. 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 $\delta(x, 0) = y$.

Sebuah finite automata terdiri dari lima komponen $(Q, \Sigma, \delta, q_0, F)$, di antaranya:

1. Q adalah himpunan set berhingga yang disebut dengan himpunan states.
2. Σ adalah himpunan berhingga alfabet dari simbol.
3. $\delta : Q \times \Sigma$ adalah fungsi transisi, merupakan fungsi yang mengambil states dan alfabet input sebagai argumen dan menghasilkan sebuah state. Fungsi transisi sering dilambangkan dengan δ .
4. $q_0 \in Q$ adalah states awal.
5. $F \subseteq Q$ adalah himpunan states akhir.

Suatu finite automata $M = (Q, \Sigma, \delta, q_0, F)$ akan menerima sebuah string w jika kumpulan states $r_0 r_1 \dots r_n$ dalam Q memenuhi tiga kondisi:

1. $r_0 = q_0$.
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ untuk $i = 0, \dots, n-1$.
3. $r_n \in F$.

dengan $w = w_1 w_2 \dots w_n$ adalah string masing-masing w_i adalah anggota alfabet Σ .

Kondisi yang pertama dinyatakan bahwa suatu finite automata dimulai dari start state. Pada kondisi yang kedua dinyatakan bahwa finite automata akan berpindah dari satu state ke state yang lain berdasarkan fungsi transisi, dan kondisi yang ketiga menyatakan bahwa finite automata akan menerima string apabila tersebut berakhir pada final state. Dapat dinyatakan bahwa M mengenali bahasa A jika $A = \{w \mid M \text{ menerima } w\}$.

Menyatakan suatu finite automata dengan menggunakan notasi 5-tuple akan sangat merepotkan. Cara yang lebih dianjurkan dalam menuliskan finite automata, yaitu dengan menggunakan:

1. Diagram transisi (transition diagram), yaitu berupa suatu graf.
2. Tabel transisi (transition table), yaitu daftar berbentuk tabel untuk fungsi δ , yang merupakan hubungan antara himpunan states dengan alfabet input.

2. Context Free Grammar

Context Free Grammar (CFG) atau Bahasa Bebas Konteks adalah sebuah tata bahasa di mana tidak terdapat pembatasan pada hasil produksinya, contoh pada aturan produksi:

$$\alpha \rightarrow \beta$$

Batasannya hanyalah ruas kiri (α) adalah sebuah simbol variabel. Sedangkan contoh aturan produksi yang termasuk CFG adalah seperti di bawah:

$$B \rightarrow CDeFg$$

$$D \rightarrow BcDe$$

Context Free Grammar (CFG) adalah tata bahasa yang mempunyai tujuan sama seperti halnya tata bahasa regular yaitu merupakan suatu cara untuk menunjukkan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa.

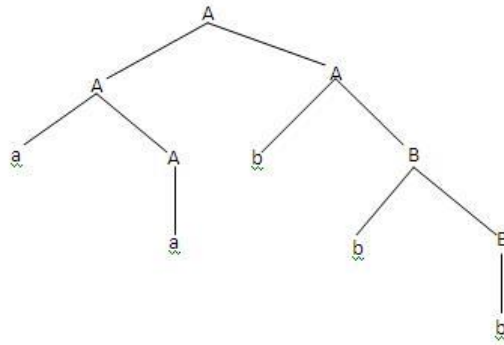
Context Free Grammar (CFG) menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan didefinisikan dalam tata bahasa bebas konteks. Pohon penurunan (derivation tree/parse tree) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan diturunkan menjadi terminal sampai tidak ada yang belum tergantikan. Sebagai contoh, terdapat CFG dengan aturan produksi sebagai berikut dengan simbol awal S :

$$S \rightarrow AB$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Maka, jika ingin dicari gambar pohon penurunan dengan string 'aabb' hasilnya adalah seperti di bawah:



Proses penurunan / parsing bisa dilakukan dengan cara sebagai berikut:

- Penurunan terkiri (leftmost derivation): simbol variabel terkiri yang diperluas terlebih dahulu.
- Penurunan terkanan (rightmost derivation): simbol variabel terkanan yang diperluas terlebih dahulu.

Misalkan terdapat grammar sebagai berikut:

$$S \rightarrow aAS \mid a$$

$$A \rightarrow SbA \mid ba$$

Untuk memperoleh string 'aabbba' dari grammar di atas dilakukan dengan cara:

- Penurunan terkiri: $S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbba$
- Penurunan terkanan: $S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aAbbaa \Rightarrow aabbba$

Ambiguitas terjadi bila terdapat lebih dari satu pohon penurunan yang berbeda untuk memperoleh suatu string. Misalkan terdapat tata bahasa sebagai berikut:

$$S \rightarrow A \mid B$$

$$A \rightarrow a$$

$$B \rightarrow a$$

Untuk memperoleh untai 'a' bisa terdapat dua cara penurunan sebagai berikut:

$$S \Rightarrow A \Rightarrow a$$

$$S \Rightarrow B \Rightarrow a$$

Sebuah string yang mempunyai lebih dari satu pohon sintaks disebut string ambigu (ambiguous). Grammar yang menghasilkan paling sedikit sebuah string ambigu disebut grammar ambigu.

3. Syntax Python

Dalam pembuatan CFG, ada beberapa peraturan per-syntax-an yang perlu diperhatikan terkait penamaan maupun struktur program yang dirinci sebagai berikut.

a. Penamaan Variabel

Tidak berbeda dengan bahasa pemrograman lainnya, Python juga memiliki variabel. Sederhananya, variabel ini digunakan untuk proses penyimpanan dan bekerja dengan berbagai tipe data. Variabel dalam Python tidak memiliki batas jumlah karakter. Karakter awal suatu variabel harus termasuk huruf, baik besar maupun kecil, atau karakter *underscore* (`_`), sedangkan karakter selain karakter awal harus termasuk huruf, baik besar maupun kecil, angka, atau karakter *underscore*. Python sendiri punya standar pendeklarasian variabel. Variabel di Python dapat berupa nama singkat (seperti `x` dan `y` tadi) atau nama yang lebih mendeskripsikan seperti umur, nama, alamat, dan lain sebagainya. Aturan penamaan variabel di Python seperti:

- Variabel tidak bisa diawali dengan angka,
- Variabel harus diawali dengan huruf, atau karakter garis bawah (*underscore*),
- Variabel hanya bisa mengandung karakter alfa-numerik dan karakter garis bawah,
- Variabel di Python bersifat *case-sensitive*

Dengan demikian, contoh variabel yang valid adalah `v`, `V`, `_v`, `var`, `var3`, `v0r`, dan `vAr`. Namun berbeda dengan bahasa pemrograman lainnya, Python tidak memerlukan inisiasi variabel serta pendefinisian tipe untuk mendeklarasikan variabel. Ini berarti sebuah variabel terbuat ketika pertama kali kita menambahkan nilai ke dalamnya dan secara otomatis akan langsung memberikan tipe variabel sesuai dengan nilai yang diberikan pada variabel tersebut.

b. Assignment

Assignment dilakukan untuk mendefinisikan isi suatu variabel dengan aturan berikut:

- Struktur *assignment* terdiri atas variabel yang ingin didefinisikan, diikuti operator *assignment*, diikuti dengan nilai dari variabel yang ingin didefinisikan.
- Variabel yang ingin didefinisikan bisa lebih dari 1. Untuk pendefinisian lebih dari 1 variabel, setiap variabel dipisahkan oleh tanda koma.
- Operator *assignment* meliputi `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `//=`, `**=` & `=`, `|=`, `^=`, `>=>`, `<<=`.
- Nilai variabel yang ingin didefinisikan dapat berupa angka, float, string, list, tuple, set, dictionary, boolean, kata “None”, fungsi, dan variabel.

c. Class

Class merupakan prototipe yang ditentukan pengguna untuk objek yang mendefinisikan seperangkat atribut yang menjadi ciri objek kelas apa pun. Atribut adalah data anggota (variabel kelas dan variabel contoh) dan metode, diakses melalui notasi titik.

Statement class digunakan untuk membuat definisi kelas baru. Nama kelas ditulis mengikuti 'class' yang kemudian diikuti pula oleh titik dua seperti yang digambarkan pada contoh berikut.

```
class ClassName:
```

Dibawah ini adalah contoh cara membuat class dan penggunaannya :

```
class Employee:
    'Common base class for all employees'
    empCount = 0
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1
    def displayCount(self):
        print "Total Employee %d" % Employee.empCount
    def displayEmployee(self):
        print "Name : ", self.name, " , Salary: ", self.salary
```

d. Function

Fungsi pada Python adalah kumpulan perintah atau baris kode yang dikelompokkan menjadi satu kesatuan untuk kemudian bisa dipanggil atau digunakan berkali-kali. Sebuah fungsi bisa menerima parameter, bisa mengembalikan suatu nilai, dan bisa dipanggil berkali-kali secara independen. Dengan fungsi kita bisa memecah program besar yang kita tulis, menjadi bagian-bagian kecil dengan tugasnya masing-masing. Juga, fungsi akan membuat kode program kita menjadi lebih "*reusable*" dan lebih terstruktur.

Di dalam python, sintaks pembuatan fungsi terlihat seperti berikut:

```
def <nama_fungsi>(parameters): statements
```

Sintaks di atas secara umum terbagi menjadi 4 bagian:

1. Kata kunci `def` yang menjadi pertanda bahwa blok kode program adalah sebuah fungsi
2. Nama fungsi yang kita buat
3. Parameters yang akan diterima oleh fungsi yang kita buat (tidak wajib)
4. Blok kode fungsi yang di sana akan kita tulis perintah-perintah yang harus dilakukan oleh sebuah fungsi

Sebagai catatan, blok kode program dalam Python didefinisikan dengan indentasi. Selain itu, pemanggilan fungsi dapat dilakukan hanya dengan mengetikkan nama fungsinya dengan menambahkan tanda kurung `()` di belakangnya. Apabila fungsi memiliki parameter, maka masukkan pula *instance* parameter ke dalam tanda kurung tersebut.

e. With

Statement with digunakan untuk membungkus eksekusi sejumlah kode dalam satu blok yang terdapat pada *methods* yang didefinisikan oleh *context manager*. *Context*

manager sendiri merupakan kelas yang mengimplementasikan metode `__enter__` dan `__exit__`. Penggunaan *statement with* memastikan bahwa metode `__exit__` dipanggil di akhir blok bersarang. Konsep ini serupa dengan blok `try..finally`. Berikut ini adalah contoh penggunaannya.

```
with open('example.txt', 'w') as my_file:
    my_file.write('Hello world!')
```

Contoh di atas menuliskan teks Hello World! ke dalam *file* example.txt. Obyek *file* mempunyai metode `__enter__` dan `__exit__` yang terdefinisi di dalamnya sehingga mereka bertindak sebagai *context manager*.

Pada implementasi kode di atas, metode `__enter__` pertama-tama dipanggil, kemudian kode di dalam *statement with* dieksekusi. Pada akhirnya, metode `__exit__` dipanggil meskipun terdapat *error*. Secara umum, eksekusi tersebut akan menutup *file stream*.

f. While Loop

Perulangan *while* pada Python adalah proses pengulangan suatu blok kode program selama sebuah kondisi terpenuhi. Singkatnya, perulangan *while* adalah perulangan yang bersifat *indefinite* alias tidak pasti, atau bahkan tidak terbatas.

Sebuah blok kode akan dilakukan terus-menerus selama suatu kondisi terpenuhi. Jika suatu kondisi ternyata tidak terpenuhi pada iterasi ke 10, maka perulangan akan berhenti. Jika kondisi yang sama pada saat yang berbeda ternyata berhenti pada iterasi ke 100, maka perulangan akan berhenti pada jumlah tersebut.

Kita bisa menulis sintaks *while* dengan cara berikut:

```
while <kondisi>: # blok kode yang akan diulang-ulang
```

Terdapat 3 komponen utama:

1. Yang pertama adalah keyword *while*, ini harus kita isi.
2. Yang kedua adalah *<kondisi>*: ini bisa berupa variabel boolean atau ekspresi logika.
3. Dan yang terakhir adalah blok (atau kumpulan baris) kode yang akan diulang-ulang kondisi terpenuhi.

Perulangan *while* sangat berkaitan dengan variabel boolean, atau *logical statement*. Karena penentuan kapan suatu blok kode akan diulang-ulang ditinjau dari True or False dari suatu pernyataan logika. Dengan demikian, jika suatu kondisi itu selalu benar, maka perulangannya pun akan selalu dieksekusi dan menciptakan sebuah kondisi yang disebut *infinite loop*.

g. For Loop

Perulangan *for* pada Python adalah perintah yang digunakan untuk melakukan iterasi dari sebuah nilai *sequence* atau data koleksi pada Python seperti List, Tuple,

String dan lain-lain. For pada python memiliki perilaku yang berbeda dengan for pada kebanyakan bahasa pemrograman yang lain, karena pada Python ia sangat berkaitan dengan data *sequence* atau data kolektif. Mungkin kalau dibandingkan dengan bahasa lain, for pada Python lebih dikenal sebagai *foreach*.

Berikut ini adalah struktur sintaks metode *for*:

```
for nilai in sequence: # blok kode for
```

Jadi, ada 3 bagian penting.

1. *sequence*: adalah sebuah nilai yang bersifat *iterable* alias bisa diulang-ulang. Tipe data yang bersifat *sequence* atau *iterable* adalah:
 - a. list
 - b. tuple
 - c. string
 - d. dan lain sebagainya
2. *nilai*: adalah setiap item yang diekstrak dari *sequence*
3. Blok kode: yaitu statemen-statemen atau perintah-perintah tertentu yang akan dieksekusi secara berulang.

h. Import Module

Modul adalah sebuah file yang berisi kode pemrograman python. Sebuah file yang berisi kode python, misalnya: *example.py*, disebut modul dan nama modulnya adalah *example*. Modul digunakan untuk memecah sebuah program besar menjadi file – file yang lebih kecil agar lebih mudah di-*manage* dan diorganisir. Modul membuat kode bersifat *reusable*, artinya satu modul bisa dipakai berulang dimana saja diperlukan. Modul tidak lain adalah program python biasa.

Kita bisa mengimpor modul python ke dalam program yang kita buat. Dengan mengimpor modul, maka definisi, variabel, fungsi dan yang lainnya yang ada di dalam modul itu bisa kita pergunakan. Kita mengimpor modul dengan menggunakan kata kunci *import*. Misalnya, kita akan mengimpor modul *example*, maka kita bisa mengetikkan perintah berikut di IDLE maupun di command prompt.

```
>>> import example
```

Setelah kita *import*, maka kita bisa mengakses isi dari modul *example*. Kita bisa mengakses fungsi maupun variabel global di dalam modul dengan menggunakan operasi titik (*.*).

Python memiliki banyak modul bawaan, misalnya modul *math*, *os*, *sys* dan lain sebagainya. Modul – modul tersebut berada di dalam direktori *Lib* ditempat Python ter-*install*. Ada beberapa sintaks yang bisa digunakan untuk mengimpor modul, yaitu sebagai berikut:

- Cara import standard, formatnya `import module_name`
- Cara import dengan rename (alias), formatnya `import module_name as alias`
- Cara mengimport sebagian, formatnya `from...import something`

- Cara mengimport semua isi modul, formatnya `import *`

i. Conditional

Percabangan –dalam dunia pemrograman– adalah proses penentuan keputusan atau dalam bahasa Inggris ini biasa disebut sebagai *conditional statement*. Ketika kita membutuhkan perbandingan antara kondisi satu dengan yang lain, Python dapat digunakan untuk mendukung kondisi logis dari matematika. Aturan logika ini biasanya digunakan untuk memberikan syarat sebelum sebuah baris program diambil.

Terdapat enam kondisi logis yang dapat digunakan di Python; sama dengan (`a == b`), tidak sama dengan (`a != b`), kurang dari (`a < b`), kurang dari atau sama dengan (`a <= b`), lebih besar dari (`a > b`), lebih besar atau sama dengan (`a >= b`). Kondisi ini dapat digunakan dengan beberapa modifikasi, lebih sering digunakan untuk “pernyataan If” dan perulangan.

Selain penggunaan “If”, kita dapat menggunakan “Elif dan Else”. Elif berarti “jika kondisi sebelumnya tidak benar, maka coba kondisi ini. Sedangkan Else menyatakan “lakukan perintah berikut jika semua kondisi tidak sesuai”. Secara singkat, pengambilan keputusan (kondisi if) digunakan untuk mengantisipasi kondisi yang terjadi saat jalannya program dan menentukan tindakan apa yang akan diambil sesuai dengan kondisi.

j. Komentar

Komentar adalah sebuah baris kode atau *statement* yang diabaikan oleh interpreter Python. Ia hanya ditulis dengan tujuan agar dapat dibaca oleh manusia sebagai sebuah catatan, bukan mesin. Komentar juga sangat penting sebagai penjelasan alur dari kode program yang kita tulis. Jika tidak, kita sendiri (si penulis kode) bisa lupa dan kebingungan jika harus menjelaskan kode program lama yang pernah kita tulis pada masa lalu. Selain itu, komentar juga dapat memudahkan *programmer* dalam melakukan proses *debugging* serta dalam mengerjakan *team-based project*. Penulisan komentar pada Python terdiri dari 2 jenis:

1. Satu baris
2. Multi baris

Komentar satu baris ditulis dengan tanda `#`. Sedangkan komentar lebih dari satu baris ditulis dengan *triple doublequote* (tanda petik dua sebanyak 3x).

k. List

Tipe data list adalah tipe data koleksi yang bersifat *ordered* (terurut) dan juga bersifat *changable* (bisa diubah). Tipe data ini bisa kita definisikan dengan tanda kurung siku `[]` di dalam Python. Cara membuat list dalam Python dapat dilihat pada kode berikut ini.

```
# list kosong
list_kosong = []
# list yang berisi kumpulan string
```

```
list_buah = ['Pisang', 'Nanas', 'Melon', 'Durian']
# list yang berisi kumpulan integer
list_nilai = [80, 70, 90, 60]
# list campuran berbagai tipe data
list_jawaban = [150, 33.33, 'Presiden Sukarno', False]
```

Pada **kode program di atas**, kita lihat bahwa **sebuah list didefinisikan menggunakan tanda kurung siku ([])**. Kita juga saksikan bahwa list pada Python, bisa berisi berbagaimacam tipe data. Bisa terdiri dari tipe data yang sejenis mau pun dari tipe data yang berbeda-beda.

Untuk menampilkan list, kita bisa menggunakan perintah `print()` untuk melihat isi dari sebuah list, baik secara menyeluruh maupun sebagian.

```
print('list_kosong:', list_kosong)
print('list_buah:', list_buah)
print('list_nilai:', list_nilai)
print('list_jawaban:', list_jawaban)
```

Jika dijalankan, kita akan mendapatkan output sebagai berikut:

```
list_kosong: []
list_buah: ['Pisang', 'Nanas', 'Melon', 'Durian']
list_nilai: [80, 70, 90, 60]
list_jawaban: [150, 33.33, 'Presiden Sukarno', False]
```

Kita juga bisa **menampilkan isi tertentu dari list dengan menggunakan indeks**. Setiap data pada list memiliki indeks sebagai alamat. Dan indeks adalah sebuah nilai integer dimulai dari 0 yang menjadi acuan di mana sebuah data disimpan di dalam list.

1. Tuple

Tuple adalah 1 dari 4 tipe data kolektif pada python yang berguna untuk menyimpan lebih dari satu nilai dalam satu variabel secara sekaligus. Tuple bersifat *ordered* (terurut) dan juga bersifat **unchangeable** (tidak bisa diubah). *Ordered* berarti datanya bisa kita akses menggunakan indeks, dan **unchangeable** berarti datanya tidak akan pernah bisa diubah setelah pertama kali didefinisikan. Dalam python, tipe data tuple didefinisikan dengan tanda kurung ().

Lalu, apa bedanya tuple dengan list? Tuple sama saja dengan list. Dia sama-sama digunakan untuk menyimpan data himpunan. Sama-sama bisa menampung berbagai macam tipe data dalam satu himpunan. Hanya saja **setelah diberi nilai, tuple tidak bisa diubah lagi**. Hal ini berbeda dengan list. Dari segi penulisan, list menggunakan kurung siku [], sedangkan tuple menggunakan kurung biasa ().

Ada 3 cara untuk membuat tuple. Perhatikan contoh berikut:

```
# cara standar
tuple_jenis_kelamin = ('laki-laki', 'perempuan')
# tanpa kurung
tuple_status_perkawinan = 'menikah', 'lajang'
# menggunakan fungsi tuple()
tuple_lulus = tuple(['lulus', 'tidak lulus'])
```

Keterangan:

- Cara yang pertama adalah cara standar dan paling dasar
- Cara yang kedua tanpa tanda kurung. Ini mungkin kelihatan agak aneh, tapi yang seperti ini normal di python
- Cara yang ketiga adalah dengan menggunakan fungsi tuple() dan melemparkan list sebagai parameternya.

Mengakses data pada tuple tidak jauh berbeda dengan cara mengakses data pada list, bahkan bisa kita bilang sama persis dalam keumumannya. Kita bisa mengakses nilai pada tuple dengan langsung mendefinisikan indeks-nya seperti berikut:

```
# cara standar
tuple_jenis_kelamin = ('laki-laki', 'perempuan')
print(tuple_jenis_kelamin[1]) # indeks satu
print(tuple_jenis_kelamin[0]) # indeks nol
```

Hasil yang akan keluar adalah:

```
perempuan
laki-laki
```

Kita juga bisa mengakses nilai pada tuple dengan negatif indeks:

```
print(tuple_jenis_kelamin[-2])
print(tuple_jenis_kelamin[-1])
```

Hasil yang akan keluar adalah:

```
laki-laki
perempuan
```

m. Set

Set dalam bahasa pemrograman python adalah tipe data kolektif yang digunakan untuk **menyimpan banyak nilai** dalam satu variabel dengan ketentuan:

- nilai anggota yang disimpan harus unik (tidak duplikat)
- nilai anggota yang sudah dimasukkan tidak bisa diubah lagi
- set bersifat unordered alias tidak berurut –yang artinya tidak bisa diakses dengan index.

Secara umum kita bisa membuat set dengan 2 cara: dengan kurung kurawal {}, atau dengan sebuah list yang kita passing ke dalam fungsi set().

```
# menggunakan kurung kurawal
himpunan_siswa = {'Huda', 'Lendis', 'Wahid', 'Basith'}
print(himpunan_siswa)
# mengkonversi list ke dalam set
```

```

himpunan_buah = set(['mangga', 'apel'])
print(himpunan_buah)
# set dengan tipe data yang berbeda-beda
set_campuran = {'manusia', 'hewan', 5, True, ('A', 'B')}
print(set_campuran)

```

Jika kita jalankan kode di atas, kita akan mendapatkan output sebagai berikut.

```

{'Wahid', 'Lendis', 'Basith', 'Huda'}
{'apel', 'mangga'}
{True, 5, ('A', 'B'), 'hewan', 'manusia'}

```

n. Dictionary

Dictionary adalah tipe data pada python yang berfungsi untuk menyimpan kumpulan data/nilai dengan pendekatan “key-value”. Dictionary sendiri memiliki dua buah komponen inti:

1. Key, merupakan nama atribut suatu item pada dictionary.
2. Value, ia adalah nilai yang disimpan pada suatu atribut.

Dictionary items memiliki 3 sifat, yaitu:

1. Unordered - tidak berurutan
2. Changeable - bisa diubah
3. Unique - alias tidak bisa menerima dua keys yang sama

Unordered artinya ia tidak berurutan, sehingga key/atribut yang pertama kali kita definisikan, tidak berarti dia akan benar-benar menjadi yang “pertama” dibandingkan dengan key yang lainnya. Juga, unordered berarti **tidak bisa diakses menggunakan index** (integer) sebagaimana halnya list. Sedangkan **changeable** artinya kita bisa kita mengubah value yang telah kita masukkan ke dalam sebuah dictionary. Hal ini berbeda dengan tipe data set maupun tuple yang mana keduanya bersifat *immutable* alias tidak bisa diubah. Dan yang terakhir, dictionary **tidak bisa memiliki lebih dari satu key yang sama** karena ia bersifat **unique**. Sehingga jika ada dua buah key yang sama, key yang didefinisikan terakhir akan menimpa nilai dari key yang didefinisikan lebih awal.

Untuk membuatnya dictionary, terdapat 2 cara:

1. Menggunakan tanda kurung kurawal {}.
2. Menggunakan fungsi atau konstruktor dict().

```

# cara pertama
buku = {
    "judul": "Daun Yang Jatuh Tidak Pernah Membenci Angin",
    "penulis": "Tere Liye"
}

# cara kedua
buku = dict(
    judul = "Daun Yang Jatuh Tidak Pernah Membenci Angin",
    penulis = "Tere Liye"
)

```

o. Ternary

Operator ternary juga dikenal dengan operator kondisi, karena digunakan untuk membuat sebuah ekspresi kondisi seperti percabangan if/else. Operator ternary sebenarnya tidak ada dalam Python, tapi Python punya cara lain untuk menggantikan operator ini. Pada bahasa pemrograman lain operator ternary menggunakan tanda tanya (?) dan titik dua (:).

```
kondisi ? <nilai true> : <nilai false>
```

Perhatikan contoh berikut:

```
aku = (umur < 10) ? "bocah" : "dewasa"
```

Dalam Python bentuknya berbeda, yaitu menggunakan IF/ELSE dalam satu baris.

```
<Nilai True> if Kondisi else <Nilai False>
```

Contoh:

```
umur = input("berapa umur kamu? ")
aku = "bocah" if umur < 10 else "dewasa"
print aku
```

Cara lain untuk membuat operasi ternary juga bisa menggunakan *Tuple* dan *List*.

```
jomblo = True
status = ("Menikah", "Single")[jomblo]
print status
```

p. Pass

Kata kunci pass adalah sebuah statemen pada Python yang tidak memiliki tugas apa pun. Tidak menginstruksi sistem untuk melakukan satu hal pun. Ia ada, tapi **keberadaannya seolah tidak ada**.

Statement pass berguna sebagai *placeholder* untuk suatu fungsi atau suatu class yang belum kita implementasikan secara nyata. Contohnya, pada saat kita ingin membuat 3 buah fungsi tapi kita masih belum ingin menuliskan semua kode programnya, maka kita bisa memanfaatkan statement pass. Berikut ini adalah penulisan sintaksnya.

```
pass
```

q. Raise

Raise merupakan salah satu cara penanganan *error* pada Python yang dilakukan dengan menggunakan *statement* raise yang dapat mengeluarkan *error* secara sengaja. Biasanya **raise** ini digunakan bersama dengan **if..else** atau pemeriksaan kondisi

lainnya. Objek *error* yang dapat dilemparkan beragam macamnya. Anda dapat melempar *error* `NetworkError`, `KeyError`, `ImportError`, `IOError`, atau *error* lainnya.

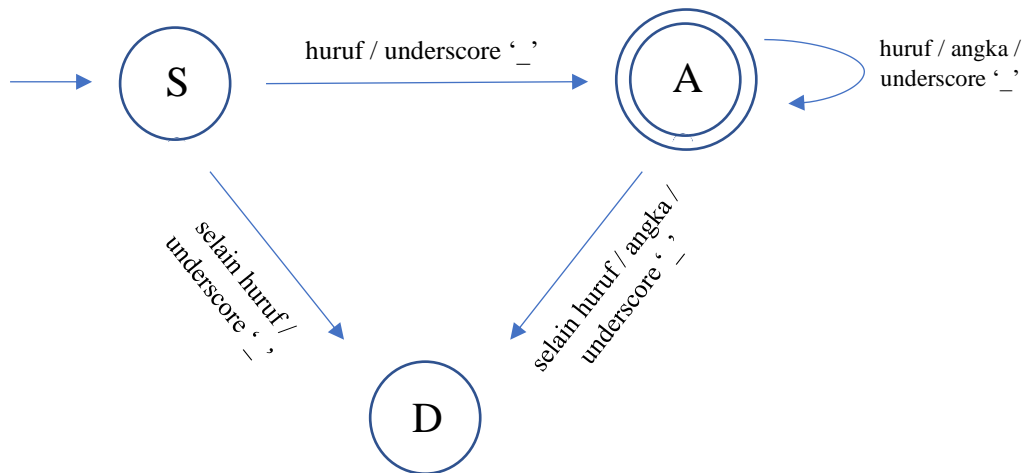
Misal pada kode dibawah ini, kita akan mencoba membangkitkan eksepsi dengan tipe `TypeError` menggunakan perintah `raise`, meskipun sebenarnya di dalam kode tersebut tidak ada kesalahan penggunaan tipe data yang tidak sesuai.

```
>>> raise TypeError
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError
```

HASIL

1. Finite Automata

Finite automata digunakan untuk mengecek penamaan setiap *identifier* yang meliputi nama variabel, fungsi, dan kelas. Finite automata yang dibuat adalah sebagai berikut:



2. Context-Free Grammar

Context-Free Grammar digunakan untuk mengecek kesesuaian *file* masukan secara keseluruhan. Context-Free Grammar yang dibuat adalah sebagai berikut:

$$G = (V, \Sigma, R, S)$$

Terminals Symbols (Σ):

'0' '1' '2' '3' '4' '5' '6' '7' '8' '9' 'a' 'b' 'c' 'd' 'e' 'f'
'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r' 's' 't' 'u' 'v'
'w' 'x' 'y' 'z' 'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L'
'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z' '!' ' "'
'#' '\$' '%' '&' ' ' ' (') ' * ' + ' , ' - ' . ' / ' : ' < ' = '
'> ' ? ' @ ' [' \ '] ' ^ ' _ ' ` ' { ' | ' } ' ~ ' space '
'newline' IdentifierName

Non-Terminal Symbols/Variables (V):

S InProgram Letter Number OtherChar AnyChar AnyString
MutlilineAnyString Numbers FloatNumbers NegativeNumbers Boolean
EOL ArithmeticOp AssignmentOp ComparisonOp LogicalOp IdentityOp
MembershipOp BitwiseOp Operator Space SpaceOrEmpty Variable
MultivarLeft Assignment Value ValueNoTernary OpParValue
OperatedValue List Tuple Set Dictionary DictionaryContents
Contents Class ClassHead ClassContents Function FunctionHead

Arguments Argument FunctionContents InFunction Return
 FunctionCall Parameter Parameters Loop ForLoopHead Iterator
 WhileLoopHead Conditions LoopContents InLoop Break Continue
 LoopInFunc FuncLoopContents InFuncLoop Import ImportOptionalAs
 With WithHead WithContents WithInLoop WithInFunc WithInFuncLoop
 If Elif Else IfHead ElifHead ElseHead IfContents IfInLoop
 ElifInLoop ElseInLoop IfInFunc ElifInFunc ElseInFunc
 IfInFuncLoop ElifInFuncLoop ElseInFuncLoop Ternary0 Ternary1
 Ternary TernaryLeft Statement StatementNoEOL SinglelineComment
 MultilineComment Pass Raise

Productions (R):

S -> InProgram S | InProgram | EOL S | EOL

InProgram -> Statement | Class | Function | Import | Loop | With | If
 | Ternary | SinglelineComment | MultilineComment

Letter -> 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' |
 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v'
 | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' |
 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S'
 | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

Number -> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

OtherChar -> '!' | '"' | '#' | '\$' | '%' | '&' | ''' | '(' | ')' | '*'
 | '+' | ',' | '-' | '.' | '/' | ':' | '<' | '=' | '>' | '?' | '@' |
 '[' | '\' | ']' | '^' | '_' | `` | '{' | '|' | '}' | '~' | 'space'

AnyChar -> Letter | Number | OtherChar

AnyString -> AnyChar AnyString | AnyChar

MutlilineAnyString -> AnyChar MutlilineAnyString | AnyChar | EOL
 MutlilineAnyString | EOL

Numbers -> Number | Number Numbers

FloatNumbers -> Numbers '.' Numbers

NegativeNumbers -> '-' SpaceOrEmpty Numbers | '-' SpaceOrEmpty
 FloatNumbers

Boolean -> 'T' 'r' 'u' 'e' | 'F' 'a' 'l' 's' 'e' | IdentifierName |
 FunctionCall | OperatedValue SpaceOrEmpty Operator SpaceOrEmpty

OperatedValue | 'n' 'o' 't' Space Boolean | 'n' 'o' 't' SpaceOrEmpty
'(' Boolean ')'

EOL -> 'newline' | SinglelineComment | Space SinglelineComment

ArithmeticOp -> '+' | '-' | '*' | '/' | '%' | '*' '*' | '/' '/'

AssignmentOp -> '=' | '+' '=' | '-' '=' | '*' '=' | '/' '=' | '%' '='
| '/' '/' '=' | '*' '*' '=' | '&' '=' | '|' '=' | '^' '=' | '>' '>'
'=' | '<' '<' '='

ComparisonOp -> '=' '=' | '!' '=' | '<' '>' | '<' '=' | '>' '='

LogicalOp -> 'a' 'n' 'd' | 'o' 'r'

IdentityOp -> 'i' 's' | 'i' 's' Space 'n' 'o' 't'

MembershipOp -> 'i' 'n' | 'n' 'o' 't' Space 'i' 'n'

BitwiseOp -> '&' | '|' | '^' | '~' | '<' '<' | '>' '>'

Operator -> ArithmeticOp | ComparisonOp | Space LogicalOp Space |
Space IdentityOp Space | Space MembershipOp Space | BitwiseOp

Space -> 'space' | 'space' Space

SpaceOrEmpty -> Space | e

Variable -> IdentifierName | IdentifierName SpaceOrEmpty '.'
SpaceOrEmpty Variable

MultivarLeft -> Variable | Variable SpaceOrEmpty ',' SpaceOrEmpty
MultivarLeft

Assignment -> MultivarLeft SpaceOrEmpty AssignmentOp SpaceOrEmpty
OperatedValue

Value -> Numbers | FloatNumbers | NegativeNumbers | ''' AnyString '''
| ''' AnyString ''' | ''' ''' | ''' ''' | List | Tuple | Set |
Dictionary | Boolean | 'N' 'o' 'n' 'e' | FunctionCall | Variable |
Ternary1 | Value SpaceOrEmpty '[' SpaceOrEmpty Value SpaceOrEmpty']'
SpaceOrEmpty

ValueNoTernary -> Numbers | FloatNumbers | ''' AnyString ''' | '''
AnyString ''' | ''' ''' | ''' ''' | List | Tuple | Set | Dictionary |
Boolean | 'N' 'o' 'n' 'e' | FunctionCall | Variable

OpParValue -> Value | '(' OpParValue ')'

OperatedValue -> OpParValue | OpParValue SpaceOrEmpty Operator
SpaceOrEmpty OperatedValue

List -> '[' Contents ']' | '[' SpaceOrEmpty ']

Tuple -> '(' Contents ')' | '(' SpaceOrEmpty ')'

Set -> '{' Contents '}' | '{' SpaceOrEmpty '}'

Dictionary -> '{' DictionaryContents '}' | '{' SpaceOrEmpty '}'

DictionaryContents -> SpaceOrEmpty OperatedValue SpaceOrEmpty ':'
SpaceOrEmpty OperatedValue SpaceOrEmpty | SpaceOrEmpty OperatedValue
SpaceOrEmpty ':' SpaceOrEmpty OperatedValue SpaceOrEmpty ','
DictionaryContents | SpaceOrEmpty

Contents -> SpaceOrEmpty OperatedValue SpaceOrEmpty | SpaceOrEmpty
OperatedValue SpaceOrEmpty ',' Contents | Space

Class -> ClassHead ClassContents

ClassHead -> 'c' 'l' 'a' 's' 's' Space IdentifierName SpaceOrEmpty ':'
EOL | 'c' 'l' 'a' 's' 's' Space IdentifierName SpaceOrEmpty '('
SpaceOrEmpty Arguments SpaceOrEmpty ')' SpaceOrEmpty ':' EOL

ClassContents -> InProgram | InProgram ClassContents

Function -> FunctionHead FunctionContents

FunctionHead -> 'd' 'e' 'f' Space IdentifierName SpaceOrEmpty '('
SpaceOrEmpty Arguments SpaceOrEmpty ')' SpaceOrEmpty ':' EOL

Arguments -> Argument SpaceOrEmpty ',' SpaceOrEmpty Arguments |
Argument | e

Argument -> IdentifierName | IdentifierName SpaceOrEmpty '='
SpaceOrEmpty OperatedValue

FunctionContents -> InFunction | InFunction FunctionContents | Return
| Return FunctionContents

InFunction -> Statement | Class | Function | Import | LoopInFunc |
WithInFunc | IfInFunc | Ternary | SinglelineComment | MultilineComment
| Return

Return -> 'r' 'e' 't' 'u' 'r' 'n' Space OperatedValue EOL

FunctionCall -> IdentifierName SpaceOrEmpty '.' SpaceOrEmpty
IdentifierName SpaceOrEmpty '(' Parameters ')' | IdentifierName
SpaceOrEmpty '(' Parameters ')'

Parameters -> Parameter SpaceOrEmpty ',' SpaceOrEmpty Parameters |
Parameter | e

Parameter -> Argument | OperatedValue

Loop -> ForLoopHead LoopContents | WhileLoopHead LoopContents

ForLoopHead -> 'f' 'o' 'r' Space IdentifierName Space 'i' 'n' Space
Iterator SpaceOrEmpty ':' EOL

Iterator -> List | Tuple | Set | Dictionary | ''' AnyString ''' | '''
AnyString ''' | ''' ''' | ''' ''' | FunctionCall | Variable

WhileLoopHead -> 'w' 'h' 'i' 'l' 'e' Conditions ':' EOL

Conditions -> Space Boolean SpaceOrEmpty | Space Boolean SpaceOrEmpty
Operator SpaceOrEmpty Boolean SpaceOrEmpty | SpaceOrEmpty '('
SpaceOrEmpty Boolean SpaceOrEmpty ')' SpaceOrEmpty | SpaceOrEmpty '('
SpaceOrEmpty Boolean SpaceOrEmpty Operator SpaceOrEmpty Boolean
SpaceOrEmpty ')' SpaceOrEmpty

LoopContents -> InLoop | InLoop LoopContents

InLoop -> Statement | Class | Function | Import | Loop | WithInLoop |
IfInLoop | Ternary | SinglelineComment | MultilineComment | Break |
Continue

Break -> 'b' 'r' 'e' 'a' 'k' EOL

Continue -> 'c' 'o' 'n' 't' 'i' 'n' 'u' 'e' EOL

LoopInFunc -> ForLoopHead FuncLoopContents | WhileLoopHead
FuncLoopContents

FuncLoopContents -> InFuncLoop | InFuncLoop FuncLoopContents

InFuncLoop -> Statement | Class | Function | Import | LoopInFunc |
WithInFuncLoop | IfInFuncLoop | Ternary | SinglelineComment |
MultilineComment | Return | Break | Continue

Import -> 'i' 'm' 'p' 'o' 'r' 't' Space IdentifierName
ImportOptionalAs EOL | 'f' 'r' 'o' 'm' Space IdentifierName Space 'i'
'm' 'p' 'o' 'r' 't' Space IdentifierName ImportOptionalAs EOL | 'f'
'r' 'o' 'm' Space IdentifierName Space 'i' 'm' 'p' 'o' 'r' 't' Space
'*' EOL

ImportOptionalAs -> e | Space 'a' 's' Space IdentifierName
With -> WithHead WithContents

WithHead -> 'w' 'i' 't' 'h' Space OperatedValue Space 'a' 's' Space
IdentifierName SpaceOrEmpty ':' EOL

WithContents -> InProgram | InProgram WithContents

WithInLoop -> WithHead LoopContents

WithInFunc -> WithHead FunctionContents

WithInFuncLoop -> WithHead FuncLoopContents

If -> IfHead IfContents | IfHead IfContents Elif | IfHead IfContents
Else

Elif -> ElifHead IfContents | ElifHead IfContents Elif | ElifHead
IfContents Else

Else -> ElseHead IfContents

IfHead -> 'i' 'f' Conditions ':' EOL

ElifHead -> 'e' 'l' 'i' 'f' Conditions ':' EOL

ElseHead -> 'e' 'l' 's' 'e' SpaceOrEmpty ':' EOL

IfContents -> InProgram | InProgram IfContents

IfInLoop -> IfHead LoopContents | IfHead LoopContents ElifInLoop |
IfHead LoopContents ElseInLoop

ElifInLoop -> ElifHead LoopContents | ElifHead LoopContents ElifInLoop
| ElifHead LoopContents ElseInLoop

ElseInLoop -> ElseHead LoopContents

IfInFunc -> IfHead FunctionContents | IfHead FunctionContents
ElifInFunc | IfHead FunctionContents ElseInFunc

ElifInFunc -> ElifHead FunctionContents | ElifHead FunctionContents
ElifInFunc | ElifHead FunctionContents ElseInFunc

ElseInFunc -> ElseHead FunctionContents

IfInFuncLoop -> IfHead FuncLoopContents | IfHead FuncLoopContents
ElifInFuncLoop | IfHead FuncLoopContents ElseInFuncLoop

ElifInFuncLoop -> ElifHead FuncLoopContents | ElifHead
FuncLoopContents ElifInFuncLoop | ElifHead FuncLoopContents
ElseInFuncLoop

ElseInFuncLoop -> ElseHead FuncLoopContents

Ternary0 -> ValueNoTernary | ValueNoTernary Space 'i' 'f' Conditions
'e' 'l' 's' 'e' Space Ternary0

Ternary1 -> TernaryLeft Space 'i' 'f' Conditions 'e' 'l' 's' 'e' Space
Ternary0

Ternary -> Ternary1 EOL

TernaryLeft -> Assignment | Raise | ValueNoTernary

Statement -> StatementNoEOL EOL

StatementNoEOL -> Assignment | FunctionCall | Pass | Raise

SinglelineComment -> '#' AnyString SinglelineComment | '#'
SinglelineComment | '#' AnyString 'newline' | '#' 'newline'

MultilineComment -> ''' ''' ''' MutlilineAnyString ''' ''' ''' EOL |
''' ''' ''' MutlilineAnyString ''' ''' ''' EOL | ''' ''' ''' '''
''' EOL | ''' ''' ''' ''' ''' ''' EOL

Pass -> 'p' 'a' 's' 's'

Raise -> 'r' 'a' 'i' 's' 'e' Space FunctionCall

Start Symbol (S): S

IMPLEMENTASI

Program terbagi menjadi 2 bagian utama, yaitu pengkonversi *context free grammar* ke *chomsky normal form* dan pemeriksa masukan menggunakan algoritma CYK (Cocke-Younger-Kasami) berdasarkan CNF. Pengkonversi CFG ke CNF diimplementasikan di *file* CFG2CNF.py dan helper.py sedangkan pemeriksa input diimplementasikan di cyk.py. Di dalam *source code* tersedia CFG di *file* cfg.txt dan hasil konversinya ke CNF di *file* cnf.txt.

1. CFG2CNF.py

CFG2CNF.py adalah *file* utama dari pengkonversi *context free grammar* ke *chomsky normal form*. Saat program ini dijalankan, CFG yang telah disediakan di *file* cfg.txt akan dikonversi menjadi CNF dalam bentuk *file* cnf.txt. Kami mengambil referensi dari <https://github.com/adelmassimo/CFG2CNF> dan melakukan beberapa modifikasi untuk menyesuaikan dengan *format* dan kebutuhan kami. Cara kerja program ini adalah dengan membaca *file* CFG, menyimpan datanya pada suatu array, dan memprosesnya dengan fungsi START, TERM, BIN, DEL, dan UNIT.

No.	Fungsi / Prosedur	Tujuan
1.	isUnitary	Mengecek apakah suatu <i>rule</i> menghasilkan tepat 1 buah variabel
2.	isSimple	Mengecek apakah suatu <i>rule</i> menghasilkan tepat 1 buah <i>terminal symbol</i>
3.	START	Menambahkan <i>non terminal symbol</i> S dan membuat <i>rule</i> $S \rightarrow S$
4.	TERM	Menggantikan <i>rule</i> yang menghasilkan <i>terminal</i> dan <i>non terminal symbol</i> dengan <i>rule</i> yang hanya mengandung <i>non terminal symbol</i> dan <i>rule</i> yang hanya mengandung 1 <i>terminal symbol</i> .
5.	BIN	Menggantikan <i>rules</i> sehingga bagian kanan setiap <i>rule</i> tidak lebih panjang dari 2 <i>non terminal symbol</i>
6.	DEL	Melakukan penghapusan epsilon
7.	unit_routine	Memeriksa apakah suatu <i>rule</i> <i>unary</i> dan memeriksa apakah <i>rule</i> tersebut bisa digantikan
8.	UNIT	Menggantikan <i>rules</i> yang hanya menghasilkan 1 variabel

2. helper.py

File helper.py adalah *file* pembantu dari CFG2CNF.py. *File* ini berisi fungsi dan prosedur yang membantu berjalannya konversi CFG ke CNF.

No.	Fungsi / Prosedur	Tujuan
1.	union	Menggabungkan 2 set menjadi sebuah list

2.	loadModel	Memuat dan mengambil data dari <i>file</i> masukan
3.	cleanProduction	Melakukan <i>formatting</i> terhadap suatu baris <i>rule</i>
4.	cleanAlphabet	Melakukan pemisahan <i>terminal</i> dan <i>non terminal symbol</i> dari <i>file</i> masukan
5.	seekAndDestroy	Melakukan eliminasi variabel yang tidak diperlukan
6.	rewrite	Menulis ulang <i>rule</i> menjadi kombinasi yang tidak mengandung epsilon
7.	pprintRules	Menampilkan suatu <i>rule</i> ke layar
8.	prettyForm	Melakukan <i>formatting</i> suatu <i>rule</i> untuk mempersiapkan keluaran

3. cyk.py

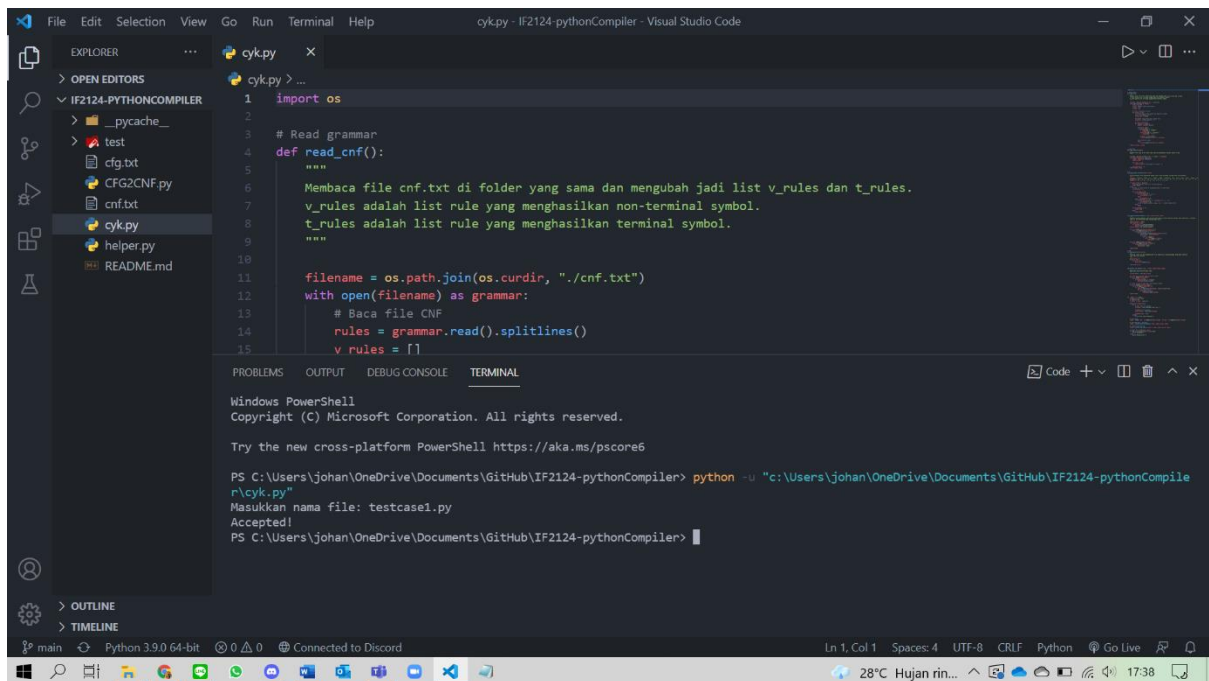
File cyk.py adalah *file* utama dari pemeriksa masukan menggunakan algoritma CYK berdasarkan CNF dan FA. Saat program dijalankan, pengguna akan diminta untuk memasukkan nama *file* yang ingin diperiksa *syntax*-nya. *File* yang ingin diperiksa harus diletakkan di folder test. Cara kerja program ini adalah dengan membaca *rules* dari *cnf.txt*, membaca *file* masukan sesuai dengan yang dipilih pengguna, membuat tabel CYK, mengecek bagian yang berpotensi menjadi variabel dengan *finite automata*, dan melakukan pengisian tabel CYK. Apabila pada blok paling ujung tabel mengandung S_0 , maka *file* masukan diterima. Jika tidak, akan ditampilkan pesan “Syntax Error”.

No.	Fungsi / Prosedur	Tujuan
1.	read_cnf	Membaca <i>file</i> <i>cnf.txt</i> dan memproses data di dalamnya hingga didapat array yang berisi <i>rule</i> yang menghasilkan <i>non terminal symbol</i> dan <i>rule</i> yang menghasilkan <i>terminal symbol</i> .
2.	read_input	Membaca <i>file</i> masukan dan mengubahnya menjadi sebuah string
3.	is_possible_variable	Menerima suatu string dan mensimulasikan finite automata untuk mengetahui apakah string dapat diterima sebagai variabel
4.	checkForIdentifierNames	Mengecek kombinasi dari string masukan, menentukan bagian yang mungkin merupakan variabel, dan menyesuaikan tabel CYK
5.	make_pairs	Menerima 2 buah set dan mengembalikan array yang berisi setiap pasangan yang dapat dibentuk oleh kedua set tersebut
6.	process_cyk_table	Menerapkan algoritma CYK pada string berdasarkan <i>rules</i> yang ada dan melakukan pengisian tabel

PENGUJIAN

1. testcase1.py

```
A = 10
B = int(input("B = "))
if ((A <= 9) or (A > 10)) and not (A == 10)):
    print("A != 10")
elif (A is B):
    print("A = B")
else: # A != B
    print("A != B")
```



```
File Edit Selection View Go Run Terminal Help
cyk.py - IF2124-pythonCompiler - Visual Studio Code

EXPLORER
> OPEN EDITORS
  IF2124-PYTHONCOMPILER
    __pycache__
    test
    cfg.txt
    CFG2CNF.py
    cnf.txt
    cyk.py
    helper.py
    README.md

cyk.py
1 import os
2
3 # Read grammar
4 def read_cnf():
5     """
6     Membaca file cnf.txt di folder yang sama dan mengubah jadi list v_rules dan t_rules.
7     v_rules adalah list rule yang menghasilkan non-terminal symbol.
8     t_rules adalah list rule yang menghasilkan terminal symbol.
9     """
10
11     filename = os.path.join(os.getcwd(), "./cnf.txt")
12     with open(filename) as grammar:
13         # Baca file CNF
14         rules = grammar.read().splitlines()
15         v_rules = []

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

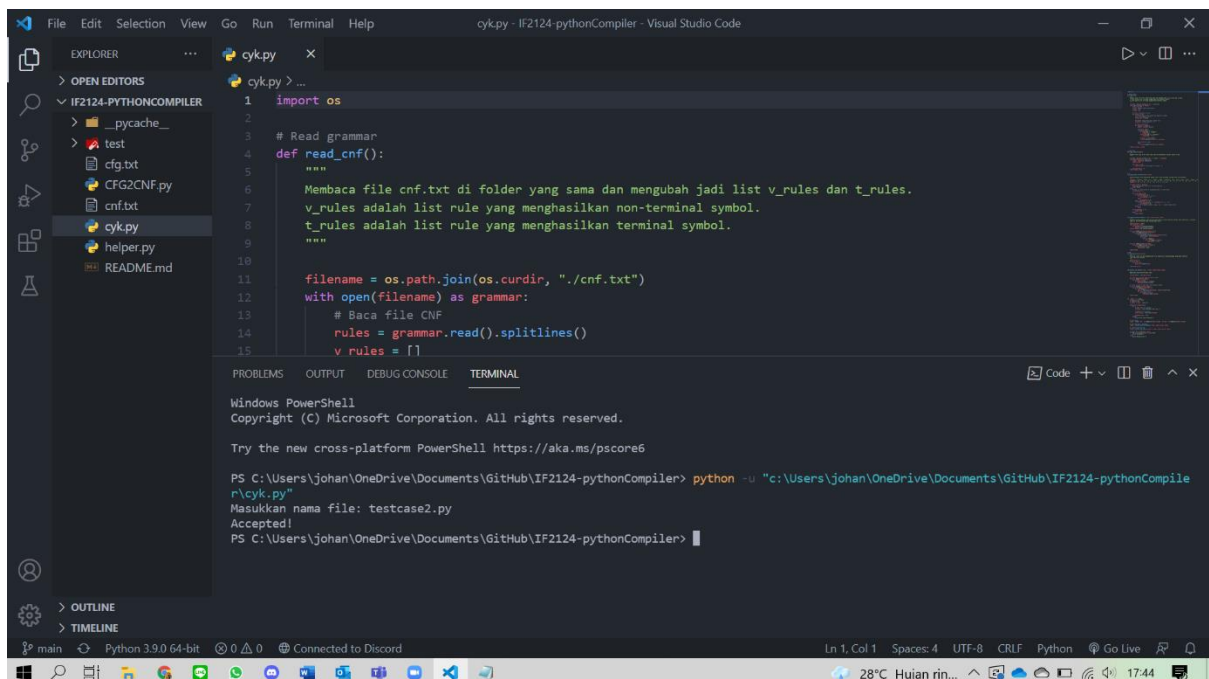
PS C:\Users\johan\OneDrive\Documents\GitHub\IF2124-pythonCompiler> python -u "c:\Users\johan\OneDrive\Documents\GitHub\IF2124-pythonCompiler\cyk.py"
Masukkan nama file: testcase1.py
Accepted!
PS C:\Users\johan\OneDrive\Documents\GitHub\IF2124-pythonCompiler>
```

Dari hasil percobaan pertama, hasil yang dikeluarkan sesuai dengan yang diharapkan, yaitu “Accepted!” yang berarti compile berhasil. Dalam gambar di atas, kami hanya menampilkan keluaran hasil evaluasi program dikarenakan proses parsing yang sangatlah panjang sehingga akan membutuhkan beberapa screenshot. Karena start symbol dari input kami adalah S_0 dan hasil CYK kami saat proses akhir menunjukkan start symbol, maka input diterima dan menjadi bagian dari language CFG Python yang telah kami definisikan sebelumnya.

Pada percobaan pertama ini, keyword-keyword yang sedang kami tes adalah and, or, not, if, elif, else, dan is. Hal lain yang juga kami tes adalah input/output, komentar, dan operator relasional.

2. testcase2.py

```
def kali(x, y):  
    """"  
    me-return hasil dari x dikali y  
    """"  
    hasil = x * y  
    return hasil  
for i in range(1, 11, 2):  
    if (i == 5):  
        continue  
    print(i, end=" ")  
print()  
i = 1  
while (i < 128):  
    if (i == 64):  
        break  
    print(i, end=" ")  
    i *= 2
```



Dari hasil percobaan kedua, hasil yang dikeluarkan sesuai dengan yang diharapkan, yaitu “Accepted!” yang berarti compile berhasil. Dalam gambar di atas, kami hanya menampilkan keluaran hasil evaluasi program dikarenakan proses parsing yang sangatlah panjang sehingga akan membutuhkan beberapa screenshot. Karena start symbol dari input

kami adalah S_0 dan hasil CYK kami saat proses akhir menunjukkan start symbol, maka input diterima dan menjadi bagian dari language CFG Python yang telah kami definisikan sebelumnya.

Pada percobaan kedua ini, keyword-keyword yang sedang kami tes adalah for, while, break, continue, def, return, dan in. Hal lain yang juga kami tes adalah operator aritmatika dan operator assignment.

3. testcase3.py

```
from matplotlib import numpy as np

with open('example.txt', 'w') as my_file:

    my_file.write('Hello, World!')

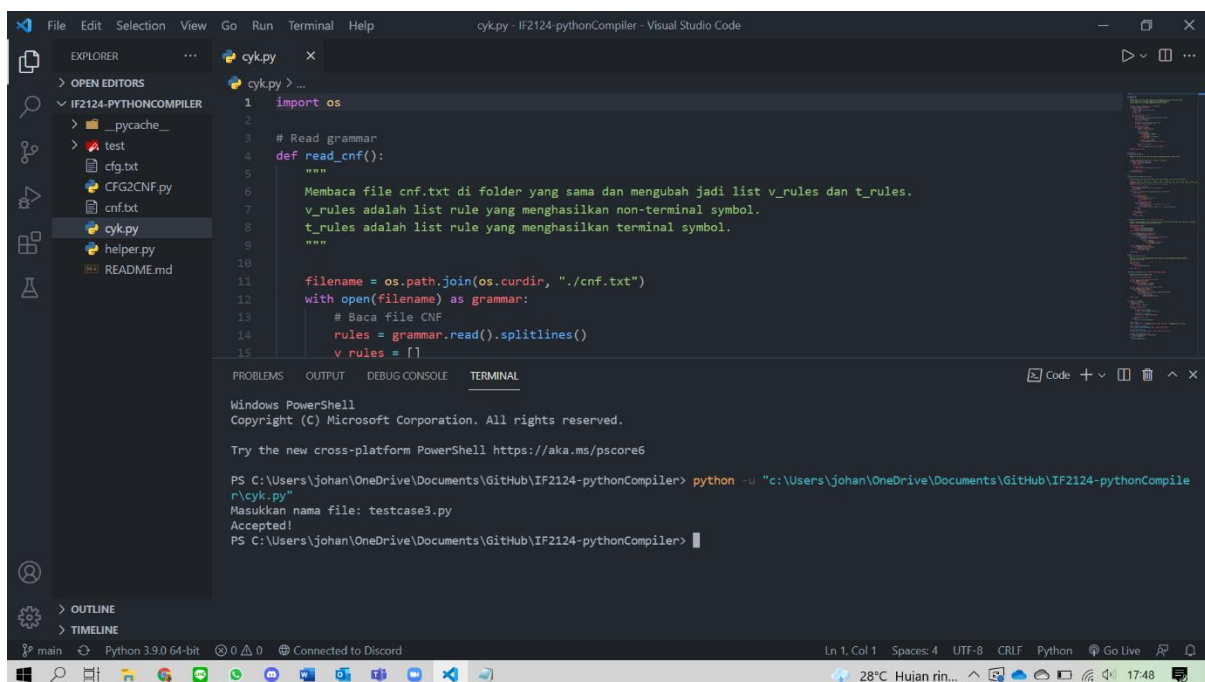
class Contoh:

    pass

def hitung_luas_lingkaran(diameter):

    pass

raise hitung_luas_lingkaran(7)
```

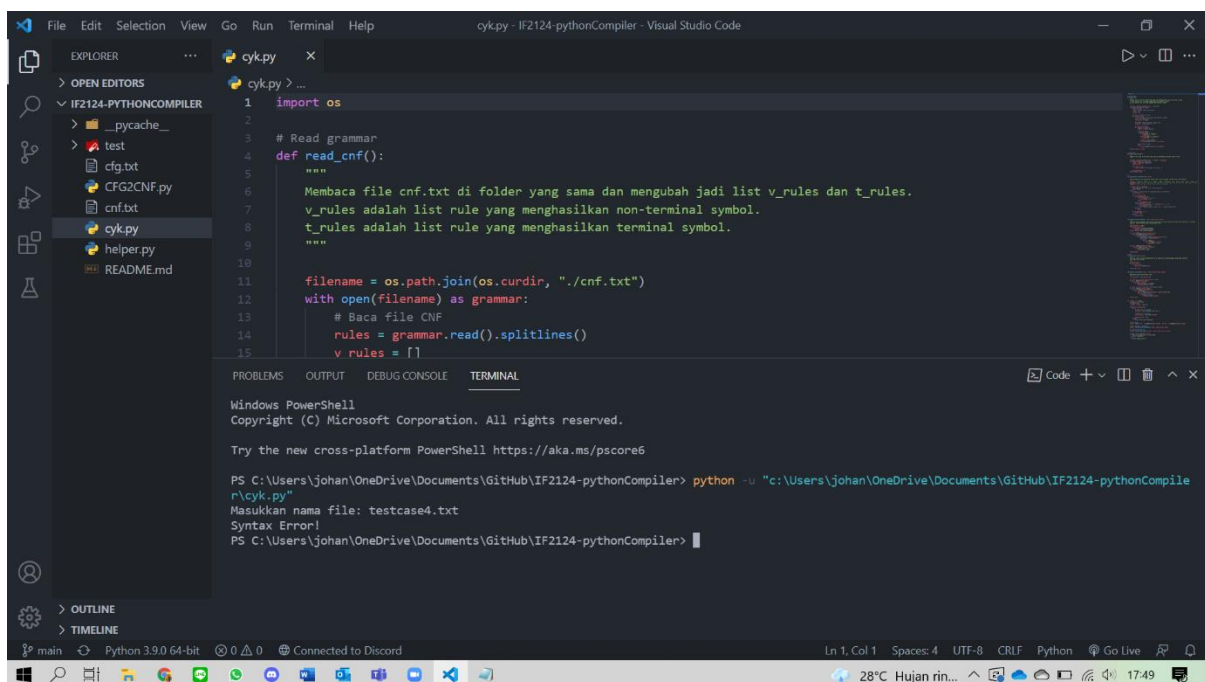


Dari hasil percobaan ketiga, hasil yang dikeluarkan sesuai dengan yang diharapkan, yaitu “Accepted!” yang berarti compile berhasil. Dalam gambar di atas, kami hanya menampilkan keluaran hasil evaluasi program dikarenakan proses parsing yang sangatlah panjang sehingga akan membutuhkan beberapa screenshot. Karena start symbol dari input kami adalah S_0 dan hasil CYK kami saat proses akhir menunjukkan start symbol, maka input diterima dan menjadi bagian dari language CFG Python yang telah kami definisikan sebelumnya.

Pada percobaan ketiga ini, keyword-keyword yang sedang kami tes adalah from, import, with, as, class, pass, dan raise.

4. testcase4.txt

```
def error(x):  
    elif (x == 0):  
        return 0  
    else: # x != 0  
        return 1
```



Dari hasil percobaan keempat, hasil yang dikeluarkan sesuai dengan yang diharapkan, yaitu “Syntax Error!” yang berarti compile gagal. Dalam gambar di atas, kami hanya menampilkan keluaran hasil evaluasi program dikarenakan proses parsing yang sangatlah panjang sehingga akan membutuhkan beberapa screenshot. Karena start symbol dari input kami adalah `S0` dan hasil CYK kami saat proses akhir tidak menunjukkan start symbol, maka input tidak diterima dan bukan bagian dari language CFG Python yang telah kami definisikan sebelumnya.

Pada percobaan keempat ini, file tersebut gagal compile dikarenakan keyword `if` belum didefinisikan sebelum keyword `elif`, sehingga pesan yang dikeluarkan adalah “Syntax Error!”.

KESIMPULAN DAN SARAN

1. Kesimpulan

Telah berhasil diimplementasikan sebuah program berupa *compiler* untuk bahasa Python yang dapat melakukan proses compiling sesuai dengan yang diminta dalam spesifikasi Tugas Pemrograman IF2124 Tata Bahasa Formal dan Otomata Semester 1 Tahun 2021/2022. Hal mengenai *compiler* bahasa Python yang berhasil diimplementasikan dalam program ini meliputi:

1. Konsep *Finite Automata* (FA) yang dapat mengevaluasi penamaan variabel
2. Konsep *Context-Free Grammar* (CFG) untuk evaluasi terhadap *syntax* program
3. Konversi CFG ke CNF (*Chomsky Normal Form*)
4. Algoritma Cocke-Younger-Kasami (CYK)
5. *Syntax* dan penggunaan bahasa pemrograman Python
6. Penggunaan dan pengolahan *library* Python, seperti *sys*, *helper*, *os*, dll.

Semua implementasi dari konsep-konsep di atas kemudian berhasil digunakan untuk menyelesaikan seluruh fitur yang ada di dalam spesifikasi. Fitur-fitur tersebut telah terdapat pada program *compiler* bahasa Python yang kami buat. Setidaknya terdapat 4 fitur utama yang dapat digunakan pada *compiler* bahasa Python kami, antara lain:

1. Program dapat menerima input berupa *file* eksternal berisi string yang merupakan kode sebuah program Python,
2. Program melakukan evaluasi sintaks dengan CFG,
3. Program melakukan evaluasi nama-nama variabel yang ada dengan FA,
4. Program akan memberikan keluaran hasil evaluasi program antara “*Accepted*” jika input diterima atau “*Syntax Error*” jika input tidak diterima.

Dengan pengimplementasian konsep *Finite Automata* dan *Context-Free Grammar* terhadap suatu bahasa pemrograman tertentu, khususnya dengan memanfaatkan algoritma *parser* Cocke-Younger-Kasami untuk melakukan evaluasi program, kita dapat membuat sebuah *compiler* terhadap bahasa pemrograman tersebut. Dalam hal ini, khususnya, kami mengimplementasikannya terhadap bahasa Python. Sebagai *output* atau keluaran dari program *compiler* kami ialah sebuah *statement* yang menyatakan diterima atau tidaknya *input* dari *file* masukan yang dibaca oleh program algoritma CYK. Bahasa yang kami gunakan pada keseluruhan pengerjaan *compiler* kami ialah bahasa Python itu sendiri. Dalam implementasinya, diperlukan konversi terlebih dahulu dari *grammar* dalam bentuk CFG ke *grammar* dalam bentuk CNF (*Chomsky Normal Form*) mengingat algoritma CYK yang digunakan hanya dapat menerima masukan *grammar* dalam wujud CNF. Setelah itu, *file* yang berisi *grammar* yang sudah dalam bentuk CNF dapat di-*input* ke dalam program untuk kemudian dilakukan evaluasi terkait *syntax* dan penggunaannya dalam bahasa Python.

Dengan demikian, kelompok menyimpulkan bahwa dengan mengerjakan Tugas Pemrograman IF2124 Tata Bahasa Formal dan Otomata Semester 1 Tahun 2021/2022 ini, dapat diketahui bahwa untuk melakukan proses kompilasi terhadap suatu bahasa pemrograman tertentu, dapat dibuat sebuah program berupa *compiler* dengan memanfaatkan konsep *Finite Automata* (FA) untuk mengatur penamaan variabel pada program serta konsep *Context-Free Grammar* (CFG) untuk mengevaluasi *syntax* program. Selain itu, digunakan pula algoritma CYK sebagai bentuk penerapan dari konsep yang telah dipelajari pada kuliah IF2124.

2. Saran

Tugas Pemrograman IF2124 Tata Bahasa Formal dan Otomata Semester 1 Tahun 2021/2022 menjadi salah satu proses pembelajaran bagi kelompok dalam menerapkan ilmu-ilmu yang telah diajarkan pada kuliah IF2124 maupun melakukan eksplorasi materi secara mandiri. Berikut ini merupakan sejumlah saran dari kelompok untuk pihak-pihak yang ingin melakukan atau mengerjakan hal serupa.

1. Program yang diminta adalah program yang memanfaatkan konsep FA dan CFG secara mendalam sehingga diperlukan eksplorasi lebih jauh dan pencarian referensi terkait kedua konsep tersebut. Selain itu, mengingat program yang dibuat ialah sebuah *compiler* bahasa Python, maka perlu disiapkan berbagai *test case* yang dapat melakukan uji terhadap *compiler* yang telah dibuat secara menyeluruh dan bertahap. Oleh karena itu, kelompok merekomendasikan agar disediakan waktu yang cukup untuk melakukan eksplorasi terkait berbagai konsep dan algoritma yang akan digunakan serta mempersiapkan berbagai *test case* yang dapat mencakup keseluruhan spesifikasi sebelum mengimplementasikannya ke dalam program. Hal ini akan meningkatkan efektivitas kerja tim dalam pembuatan suatu program.
2. Terdapat beberapa algoritma *parser* yang dapat digunakan dalam implementasi pembuatan *compiler* bahasa Python dengan memanfaatkan konsep FA dan CFG seperti yang telah disebutkan pada bagian Deskripsi Masalah. Setiap algoritma yang ada tentu memiliki keunggulan dan kelemahannya masing-masing. Dengan demikian, pemilihan algoritma yang tepat dan sesuai juga menjadi salah satu faktor penting dalam meningkatkan efektivitas pengerjaan program. Kelompok kami sendiri memilih untuk menggunakan algoritma CYK sesuai saran dan rekomendasi yang disampaikan pada spesifikasi tugas pemrograman ini. Di samping itu, perlu dipertimbangkan pula waktu yang dimiliki untuk melakukan eksplorasi terhadap suatu bahasa pemrograman serta algoritma khusus tertentu sehingga tidak membebani *programmer* dalam pengerjaan proyek dengan jangka waktu yang singkat. Pilihlah algoritma yang paling *feasible* untuk dipelajari dan dieksekusi.
3. Penting bagi kelompok untuk memiliki strategi serta distribusi tugas yang baik. Ketika membuat program dalam sebuah tim, kesamaan cara menulis kode serta kemampuan untuk menulis komentar menjadi hal yang sangat penting. Hal ini diperlukan agar memudahkan anggota kelompok dalam menyatukan dan melanjutkan sebuah program. Kemampuan tersebut tentunya didukung juga dengan adanya *version control system* yang baik yang dapat digunakan oleh *programmer* dalam membuat sebuah program secara bersama-sama. Untuk itu, kami sangat menyarankan ‘GitHub’ untuk digunakan sebagai *version control system* dalam pengerjaan tugas-tugas besar pada mata kuliah IF2124 ini, maupun pada pembuatan program dan pengerjaan proyek yang lainnya.
4. Kelompok menyadari bahwa pada program *compiler* bahasa Python yang telah kami buat, masih banyak aspek yang dapat dikembangkan lebih lagi. Salah satunya ialah dengan mengoptimalkan algoritma agar program mampu melakukan proses *compiling* dan evaluasi *syntax* dengan waktu yang lebih singkat. Hal ini tentu menjadi ruang untuk *programmer* agar dapat melakukan improvisasi terhadap *compiler* yang dimilikinya, terutama dalam hal waktu eksekusi. Selain itu, kelompok juga merekomendasikan untuk menyediakan fitur yang dapat memberi tahu letak dan detail kesalahan/*error* pada *syntax* program apabila ada. Dengan demikian, pengguna akan lebih dipermudah dalam melakukan *debugging* terhadap program Python yang ditulisnya (meningkatkan nilai *user experience*).

LAMPIRAN

1. Pembagian Tugas

No.	Nama	Tugas
1.	Johannes Winson Sukiatmodjo (13520123)	Mencari referensi <i>converter</i> CFG to CNF Membuat <i>test cases</i> Melakukan testing pada program Menulis laporan
2.	Ignasius Ferry Priguna (13520126)	Membuat CFG Menyesuaikan <i>converter</i> CFG to CNF Membuat program algoritma CYK Membuat program FA Menulis laporan
3.	Nelsen Putra (13520130)	Membuat <i>repository GitHub</i> Inisialisasi pembuatan algoritma CYK Membuat README Melakukan testing pada program Menulis laporan

2. Link Repository

Link repository *GitHub* kelompok sipitBukanHalangan Tugas Pemrograman IF2124 Tata Bahasa Formal dan Otomata: <https://github.com/nelsenputra/IF2124-pythonCompiler>

REFERENSI

Codepolitan.com. (2016). Mengenal Statement Try Except di Python. Diakses pada 22 November 2021, dari <https://www.codepolitan.com/mengenal-statement-try-except-di-python>

Github.com. (2018). CFG2CNF. Diakses pada 17 November 2021, dari <https://github.com/adelmassimo/CFG2CNF>

Jagongoding.com. (2021). Python Dasar. Diakses pada 22 November 2021, dari <https://jagongoding.com/python/dasar/>

Niagahoster.co.id. (2019). Belajar Python Pemula: Pengenalan Dasar. Diakses pada 22 November 2021, dari <https://www.niagahoster.co.id/blog/belajar-python/#Sintaks>

Petanikode.com. (2021). Belajar Pemrograman Python: Mengenal 6 Jenis Operator dalam Python. Diakses pada 23 November 2021, dari <https://www.petanikode.com/python-operator/>

W3schools.com. (2021). Python Tutorial. Diakses pada 19 November 2021, dari <https://www.w3schools.com/python/>

Xarg.org. (2021). The CYK Algorithm. Diakses pada 16 November 2021, dari <https://www.xarg.org/tools/cyk-algorithm/>