LAPORAN TEKNIK KOMPILASI

SUNDANASE LANGUAGE

"Laporan ini dikumpulkan untuk memenuhi salah satu tugas mata kuliah Teknik Kompilasi"

Oleh:

Rafli Hillan Yufandani (4611419069)

Wisnu Afifuddin (4611419052)

Muhammad Ammar Nabil (4611419055)

TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS NEGERI SEMARANG

2021

PRAKATA

Puji syukur penulis panjatkan kepada Allah SWT yang telah melimpahkan rahmat, hidayah dan inayah-Nya kepada kita sehingga penulis dapat menyelesaikan laporam ini tanpa suatu hambatan apapun. Tidak lupa sholawat serta salam kami panjatkan kepada Nabi Muhammad SAW yang telah membimbing kita dari zaman jahiliyah ke zaman yang penuh cahaya.

Pembuatan laporan ini merupakan bentuk dari pemenuhan Tugas mata kuliah Teknik Kompilasi diampu oleh Bapak Aji Purwinarko S.Si, M.Cs. Laporan ini berisi informasi tentang tahapan pembuatan pemrograman bahasa sunda atau *Sundanase Language* dengan dasar bahasa pemrograman python yang dibuat oleh kelompok 5

Penulis mengucapkan terimakasih kepada semua pihak yang telah membantu dalam proses pembuatan laporan ini. Penulis menyadari bahwa masih ada kekurangan dalam laporan ini, oleh karena itu dimohon dengan kerendahan hati supaya pembaca memberikan kritik dan saran yang membangun supaya makalah ini menjadi lebih baik.

Semarang, 23 Juni 2021

Kelompok 5

1. Pembuatan

A. Analisis Leksikal (Lexer)

Analisis leksikal atau Lexer adalah suatu komponen yang berfungsi untuk mendeteksi bagian terkecil (token) dari suatu bahasa, ia menerima masukan serangkaian karakter dan menghasilkan deretan simbol yang masing-masing dinamakan token. Proses pendeteksian ini disebut *lexing*. Biasanya *regular expressions* digunakan untuk melakukan proses lexing karena bentuk token biasanya sangat sederhana.

```
from sly import Lexer
```

sly (Sly Lex Yacc) adalah salah satu libraries dari bahasa pemrograman python yang befungsi untuk menulis parser dan compiler.

“SLY menyediakan dua kelas Lexer dan Parser yang terpisah. Kelas Lexer digunakan untuk memecah teks input menjadi sebuah koleksi token yang ditentukan oleh kumpulan aturan ekspresi reguler. Kelas Parser digunakan untuk mengenali sintaks bahasa yang telah ditentukan dalam bentuk tata bahasa bebas konteks.”

```
class SundanaseLexer(Lexer):
    tokens = { NAME, NUMBER, STRING, IF, THEN, ELSE, FOR, TO, ARROW, EQEQ, FUN, P
RINT, ERROR }

    # A string containing ignored characters (spaces and tabs)
    ignore = ' \t '
    literals = { '=', '+', '-', '/', '*', '(', ')', ',', ';'}
```

Dibagian ini kita membuat sebuah class Lexer dari SLY, dan membuat compiler yang membuat operasi aritmatika, di dalam class ini terdapat tokens,literals, dan ignore. Lexers harus menentukan set token yang mendefinisikan semua kemungkinan nama tipe token yang dapat diproduksi oleh lexer. Ini selalu diperlukan dan digunakan untuk melakukan berbagai pemeriksaan validasi.

Ignore berfungsi pada saat membutuhkan beberapa token, misalnya nama, number, string pasti ada jarak atau spasi diantara keduanya atau ada spasi, untuk itu ignore berfungsi untuk mengabaikan spasinya.

Literals adalah notasi untuk mewakili nilai tetap dalam kode sumber atau tidak terjadi perubahan atau konstan.

```
# Tokens
    IF = r'UPAMI'
    THEN = r'TERAS'
    ELSE = r'HENTEU'
    FOR = r'KAHATUR'
    TO = r'KANGGO'
    FUN = r'FUN'
    ARROW = r'->'
    EQEQ = r'=='
    PRINT= r'PRINT'
    ERROR =r'LEPAT'
```

Pada Token di atas ini befungsi untuk menentukan token sebagai ekspresi regular dan menyimpannya sebagai string atau deretan symbol, pada bagian tokens ini kita bebas menamakan perumpamaan nama sesuai dengan bahasa apa yang kalian ingin gunakan, seperti saya yang ingin membuat bahasa sundanase maka tokens yang saya gunakan layaknya bahasa sunda pada umumnya.

r (*Python raw string*) dibuat dengan mengawali string literal dengan 'r', Python raw string memperlakukan garis miring terbalik (\) sebagai karakter literal. Ini berguna saat kita ingin memiliki string yang berisi garis miring terbalik dan tidak ingin diperlakukan sebagai karakter escape.

```
        NAME = r'[a-zA-Z_][a-zA-Z0-9_]*'
```

Token name menerima inputan dari lowercase sampai dengan uppercase serta number dan underscroce.

```
#Define Number Tokens
    @_(r'\d+')
    def NUMBER(self, t):
        t.value = int(t.value)
        return t
```

Pada bagain tokens diatas terdapat tokens number, untuk itu terdapat fungsi number dalam bentuk integer. ***"def"*** (Fungsi dalam Python didefinisikan menggunakan kata kunci def), ***"(self,t)"*** salah satu sintaks untuk memanggil dirinya sendiri dan t adalah nama token yang dipanggilnya.

```
        t.value = int(t.value)
```

```
        return t
```

Mengkonversi kedalam bentuk integer, dan t.value adalah sebuah fuction

```
        #Define Comment Token

    @_(r'#.*')
    def COMMENT(self, t):
        pass

    # Define a rule so we can track line numbers
    @_(r'\n+')
    def newline(self,t ):
        self.lineno = t.value.count('\n')
```

Pada new line token terdapat fungsi *Self.lineno* berfungsi untuk menampilkan line yang error pada code yang dibuat.

```
 #Define Error
    def error(self, t):
        print("Illegal character '%s'" % t.value[0])
        self.index += 1
```

Contoh dari illegal karakter yang membuat error token yaitu huruf \, |

```
if __name__ == '__main__':
    lexer = SundanaseLexer()
    env = {}
    while True:
        try:
            text = input('Sundanase > ')
        except EOFError:
            break
        if text:
            lex = lexer.tokenize(text)
            for token in lex:
                print(token)
```

Pada code diatas berfungsi untuk membuat terminal yang nantinya code tersebut adalah tempat untuk nge run/menjalankan code yang telah kita buat. **Tokenize** adalah proses untuk membagi teks yang dapat berupa kalimat, paragraf atau dokumen, menjadi token-token/bagian-bagian tertentu. Lexer hanya memiliki satu metode publik tokenize (). Ini adalah fungsi generator yang menghasilkan aliran Token contoh. Atribut tipe dan nilai Token masing-masing berisi nama dan nilai tipe token.

```
env = {}
```

Virtual environment adalah sebuah kakas yang digunakan untuk membuat sebuah environment yang terisolasi dari dunia luar. Sehingga, modul python apapun yang diinstall tidak mempengaruhi environment global, dan yang terpenting, project lainnya.

B. Analisis Sintaksis (Parser)

Analisis sintaksis terdiri dari dua kata yaitu analisis (Analisa) dan sintaksis (aturan-aturan dalam membentuk kalimat), Penganalisis intak dalam bidang kompilasi sering disebut dengan *Parse*. Analisis Sintaksis lebih sering disebut penguraian (parsing), tujuan utama dari analisis sintak adalah memeriksa apakah urutan token-token yang dihasilkan sesuai dengan tata bahasa dari bahasa yang bersangkutan. Dalam komputasi linguistik istilah yang digunakan untuk merujuk pada analisis formal oleh komputer dari kalimat atau lainnya serangkaian kata-kata menjadi konstitutennya, sehingga pohon parsing menunjukkan hubungan sintaksis mereka satu sama lain, yang juga mungkin informasi semantik dan lainnya. Dalam ilmu komputer, isitilah yang digunakan dalam analisis bahasa komputer, mengacu pada ada analisis sintaksis kode masukan menjadi beberapa bagian untuk memfasilitasi penulisan *compiler* dan *interpreter*.

Dalam komputasi, parser atau parsing adalah suatu cara memecah-mecah suatu rangkaian masukan (misalnya dari berkas atau keyboard) yang akan menghasilkan suatu pohon uraian (parse tree) yang akan digunakan pada tahap kompilasi berikutnya yaitu analisis semantik. Parser sering menggunakan penganalisis leksikal terpisah untuk membuat token dari urutan karakter masukan. Penggunaan yang paling umum dari parser adalah sebagai komponen kompilator atau interpreter. Ini mengurai kode sumber dari bahasa pemrograman komputer untuk membuat beberapa bentuk representasi internal.

Pohon (tree) adalah suatu graph terhubung yang tidak sirkuler, memiliki satu buah simpul (atau vertex / node) yaitu akar (root) dan dari akar ini memiliki lintasan (atau edge) ke setiap simpul yang lain. pohon penurunan (atau derivation tree / syntax tree / parse tree) berguna untuk menggambarkan bagaimana cara memperoleh suatu untai (string) dengan cara menurunkan atau mengganti simbol-simbol variabel menjadi terminal. Setiap simbol variabel akan diturunkan atau diganti menjadi terminal, adapun tugas analisis sintaksis yaitu:

1. Analisis Sintaksis menghasilakan pohon sintaks program sumber yang didefinisi grammar
2. Menurunkan simbol non – terminal.
3. Mengelompokkan token – token.

```
from sly import Parser

import sundanase_lexer

class SundanaseParser(Parser):
    tokens = sundanase_lexer.SundanaseLexer.tokens

    precedence = (
        ('left', '+', '-'),
        ('left', '*', '/'),
        ('right', 'UMINUS'),
        )
```

Kelompok kami menggunakan library SLY (SLY LEX YACC) dan mengimport parser. Lalu mengimport sundanase_lexer dari file lexer yang sebelumnya dibuat. Dengan membuat class Sundanase Parser ini untuk memperluar Lexer, dengan mengalirkan Token dari Basic Lexer ke Token Variabel. Di dalam class Basic Parser terdapat token yang diteruskan dari lexer ke parser dengan memanggil class Basic Lexer pada file sundanase_lexer. Setelah itu ada precedence, precedence itu sendiri merupakan tingkat prioritas aritmatika, dimana urutannya dari tingkat prioritas terendah hingga tingkat prioritas tertinggi, untuk kata ‘left’, itu dimaksudkan sebagai asosiatif. Untuk mengatasi ambiguitas, terutama dalam tata bahasa ekspresi, SLY memungkinkan token individu diberi prioritas tingkat dan asosiatif. Untuk UMINUS itu unary minus, yang berfungsi untuk menghitung atau menangkap memory dari operand yang dikirimkan.

```
  def __init__(self):
        self.env = { }
```

__init__ berfunsi untuk mengalihkan objek baru sedangkan untuk fungsi statement yang kosong, fungsinya jika dibutuhkan atau tidak saja, bersifat flexibel.

```
  @_('FOR var_assign TO  expr THEN statement')
    def statement(self,p):
        return ('for_loop', ('for_loop_setup', p.var_assign, p.expr), p.statement
)
```

```
    @_('IF condition THEN statement ELSE statement')
    def statement(self,p):
        return('if_stmt', p.condition,('branch', p.statement0, p.statement1))
```

Pada fungsi selanjutnya terdapat fungsi looping, dimana fungsi tersebut mempunyai syntax (FOR var assign To expr THEN statement) lalu mengembalikan nilai For lalu diikuti dengan var_assign, expr dan statement. Begitu juga dengan fungsi percabangan dengan menggunakan IF ELSE yang mempunyai syntax seperti gambar di atas dengan mengembalikan nilai yang telah ditentukan.

```
@_('FUN NAME "(" ")" ARROW statement')
    def statement(self,p):
        return ('fun_def', p.NAME, p.statement)

    @_('NAME "(" ")"')
    def statement(self,p):
        return('fun_call', p.NAME)

    @_('expr EQEQ expr')
    def condition(self, p):
        return ('condition_eqeq', p.expr0, p.expr1)
```

Lalu dilanjutkan dengan fungsi-fungsi yang tertera pada gambar di atas, dengan mengembalikan nilai sesuai dengan syntax yang diberikan.

```
@_('var_assign')
    def statement(self, p):
        return p.var_assign

    @_('NAME "=" expr')
    def var_assign(self,p):
        return('var_assign', p.NAME, p.expr)

    @_('NAME "=" STRING')
    def var_assign(self,p):
        return('var_assign', p.NAME, p.STRING)
```

Ada ***var_assign*** untuk membuat objek baru, *var_assign* ini juga menghandle fungsi-fungsi berikutnya.

```
@_('expr')
    def statement(self,p):
        return (p.expr)

    @_('expr "+" expr')
    def expr(self,p):
        return('add', p.expr0, p.expr1)
```

```
    @_('expr "-" expr')
    def expr(self,p):
        return('sub', p.expr0, p.expr1)

    @_('expr "*" expr')
    def expr(self,p):
        return('mul', p.expr0, p.expr1)

    @_('expr "/" expr')
    def expr(self,p):
        return('div', p.expr0, p.expr1)
```

EXPR di atas ini untuk meng-handling fungsi-fungsi di bawahnya. Pernyataan prioritas ini menetapkan bahwa PLUS atau MINUS memiliki tingkat prioritas yang sama dan asosiatif kiri dan bahwa TIMES atu DIVIDE memiliki prioritas yang sama dan bersifat kiri-asosiatif. Dalam deklarasi prioritas, token diurutkan dari prioritas terendah hingga tertinggi. Jadi, pernyataan ini menetapkan bahwa TIMES atu DIVIDE memiliki prioritas lebih tinggi daripada PLUS atau MINUS (karena muncul kemudian dalam spesifikasi prioritas). Spesifikasi prioritas berfungsi dengan mengaitkan nilai tingkat prioritas numerik dan arah asosiatif ketoken yang terdaftar. Untuk mengatasi ambiguitas, terutama dalam tata bahasa ekspresi, SLY memungkinkan token individu diberi prioritas tingkat dan asosiatif.

```
@_('"-" expr %prec UMINUS')
    def expr(self,p):
        return p.expr
```

Dalam kasus ini, % prec UMINUS menimpa aturan default yang diutamakan - menyetelnya ke UMINUS di prioritas penentu. Pada awalnya, penggunaan UMINUS dalam contoh ini mungkin tampak sangat membingungkan. UMINUS bukanlah token input atau tata bahasa aturan. Sebaliknya, Anda harus menganggapnya sebagai nama penanda khusus di tabel prioritas. Saat

Anda menggunakan % prec qualifier, Anda memberi tahu SLY bahwa Anda ingin prioritas ekspresi sama seperti untuk penanda khusus ini bukannya diutamakan seperti biasa. Dimungkinkan juga untuk menentukan non-asosiatif dalam tabel prioritas. Ini digunakan saat Anda tidak menginginkan operasi untuk dirantai bersama. Misalnya, Anda ingin mendukung operator perbandingan seperti <and>.

```
@_('NAME')
    def expr(self,p):
        return('var', p.NAME)

    @_('NUMBER')
    def expr(self,p):
        return('num', p.NUMBER)

    @_('PRINT STRING')
    def statement(self,p):
        return('print', p.STRING)

    @_(' PRINT expr')
    def statement(self,p):
        return('print', p.expr)

    @_('PRINT ERROR')
    def statement(self,p):
        print("Syntax error in print statement. Bad expression")
```

Fungsi name dan number untuk meng-handling keluaran , misal name yaitu var, dan number yaitu num, jika pada terminal mengetikan kata “saya” maka fungsi name akan jalan dengan mengeluarkan kata var pada terminal begitu juga dengan fungsi number. Untuk print string, expr berfungsi untuk menampilkan saja apa yang diketik oleh pengguna.

Fungsi name dan number untuk meng-handling keluaran , misal name yaitu var, dan number yaitu num, jika pada terminal mengetikan kata “saya” maka fungsi name akan jalan dengan mengeluarkan kata var pada terminal begitu juga dengan fungsi number. Untuk print string, expr berfungsi untuk menampilkan saja apa yang diketik oleh pengguna.

Sedangkan print error terjadi karena token buruk, pertama yang ditemukan akan menyebabkan aturan dikurangi yang mungkin akan mempersulitnya pulih jika lebih banyak token buruk segera menyusul. Lebih baik memiliki semacam tengara seperti titik

koma, penutup, tanda kurung, atau token lain yang dapat digunakan sebagai titik sinkronisasi.

Jika Anda membuat parser untuk penggunaan produksi, penanganan kesalahan sintaks sangatlah penting. Sebagai aturan umum, Anda tidak melakukannya ingin parser hanya mengangkat tangannya dan berhenti pada tanda masalah pertama. Sebaliknya, Anda ingin itu. melaporkan kesalahan tersebut, pulihkan jika memungkinkan, dan lanjutkan penguraian sehingga semua kesalahan dalam masukan dapat dilaporkan ke pengguna sekaligus. Ini adalah perilaku standar yang ditemukan di compiler untuk bahasa seperti C, C ++, dan Java.

```
if __name__ == '__main__':
    lexer = sundanase_lexer.SundanaseLexer()
    parser = SundanaseParser()
    env = {}
    while True:
        try:
            text = input('Sundanase > ')
        except EOFError:
            break
        if text:
            tree = parser.parse(lexer.tokenize(text))
            print(tree)
```

Gambar diatas berfungsi untuk menampilkan terminal. Setiap kali penerjemah Python membaca file sumber, itu melakukan dua hal:

1. Itu menetapkan beberapa variabel khusus seperti __name__, dan kemudian.
2. Itu mengeksekusi semua kode yang ditemukan dalam file.

'__main__' adalah nama ruang lingkup di mana kode tingkat atas dijalankan. Sebuah modul __name__ ditetapkan setara dengan '__main__' ketika dibaca dari standar input, skrip, atau dari prompt interaktif.

Sebuah modul dapat mengetahui apakah ia berjalan di ruang utama dengan memeriksa __name__ miliknya sendiri, yang memungkinkan idiom umum untuk mengeksekusi kode secara kondisional dalam sebuah modul ketika dijalankan sebagai skrip. Ketika true maka terminal sundanase akan memberikan baris input, kecuali jika EOFError maka akan berhenti. Maksude dari EOFError ini adalah End of File, dimana

ketika kita me-trigger sebuah EOF, maka program kita membuat program tersebut sadar bahwa tidak adala lagi data yang akan di-inputkan, dan EOF ini akan mengevaluasi nilai negative. Lalu ada variabel tree, yaitu untuk memproses data yang diinputkan pada text, setelah itu di print.

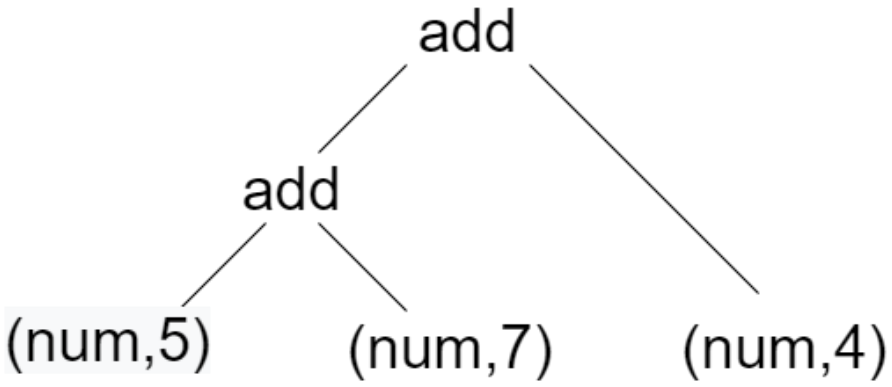
C. Interpreter

Dalam ilmu komputer, penerjemah atau lebih dikenal dengan interpreter merupakan perangkat lunak yang berfungsi melakukan eksekusi sejumlah instruksi yang ditulis dalam suatu bahasa pemrograman, sebuah penerjemah dapat berarti:

1. Mengeksekusi kode sumber secara langsung, atau
2. Menerjemahkannya ke dalam serangkaian p-code kemudian mengeksekusinya, atau
3. Mengeksekusi kode yang telah dikompilasi sebelumnya oleh kompiler yang merupakan bagian dari sistem penerjemahan. (Wikipedia)

Interpreter tidak sama dengan compiler. Pada compiler, kita akan menemukan bahwa hasil yang dihasilkan sudah berbentuk satu kesatuan perintah yang sudah ada dan berbentuk bahasa mesin. Selain itu, proses penerjemahannya dilakukan sebelum program tersebut akan dieksekusi. (Efendi 2021).

Interpreter adalah prosedur sederhana. Ide dasarnya adalah untuk mengambil pohon dan berjalan melaluinya dan mengevaluasi operasi aritmatika secara hierarkis. Proses ini dipanggil secara rekursif berulang-ulang sampai seluruh pohon dievaluasi dan jawabannya diambil. Katakanlah, misalnya, 5 + 7 + 4. Aliran karakter ini pertama kali di-token ke aliran token di lexer. Aliran token kemudian diurai untuk membentuk pohon parse. Pohon parse pada dasarnya mengembalikan ('tambah', ('tambah', ('bil', 5), ('num', 7)), ('num', 4)). (lihat gambar di bawah).



Interpreter akan menjumlahkan 5 dan 7 terlebih dahulu lalu memanggil walkTree secara rekursif dan menambahkan 4 ke hasil penjumlahan 5 dan 7. Jadi, kita akan mendapatkan 16. Pada kali ini interpreter digunakan pada tahap selanjutnya dalam pembuatan bahasa pemrograman Sundanase. Setelah melalui tahap lexer dan parser, program kemudian akan di terjemahkan kedalam bahasa yang diinginkan oleh programmer.

Tahap pertama dalam pembuatan lexer adalah mengimport class-class yang ada di file sundanase_lexer.py dan sundanase_parser.py, dengan cara:

```
import sundanase_lexer
import sundanase_parser
```

Kemudian membuat class interpreter dengan nama class SundanaseExecute dan binary tree, Pohon biner (binary tree) adalah struktur data di mana setiap node memiliki paling banyak dua anak (anak kiri dan kanan). Akar pohon ada di atas. Setiap simpul di bawah memiliki simpul di atas yang dikenal sebagai simpul induk. Kami mendefinisikan kelas yang memiliki atribut kiri dan kanan. Dari pohon biner ini, kami dapat mendefinisikan akar dan simpul kiri dan kanan. Fungsi ***__init__*** mempunyai fungsi yaitu memanggil objek atau methode yang telah dibuat sebelumnya dan ***Fungsi isinstance()*** Python memeriksa apakah objek atau variabel adalah turunan dari tipe kelas atau tipe data yang ditentukan.

```
class SundanaseExecute:

def __init__(self, tree, env):
        self.env = env
        result = self.walkTree(tree)
```

```
        if result is not None and isinstance(result, int):
            print(result)
        if isinstance(result, str) and result[0] == '"':
            print(result)
```

Langkah selanjutnya adalah membuat walktree:

```
def walkTree(self, node):

        if isinstance(node, int):
            return node
        if isinstance(node, str):
            return node

        if node is None:
            return None

        if node[0] == 'program':
            if node[1] == None:
                self.walkTree(node[2])
            else:
                self.walkTree(node[1])
                self.walkTree(node[2])
```

Node pasti memiliki class atau variabel pada intinya jika ada node dengan index 1 kosong maka memanggil methode walkTree pada node index 2, tetapi jika ada isinya memanggil methode di node index 1 dan node index 2.

```
        if node[0] == 'num':
            return node[1]

        if node[0] == 'str':
            return node[1]

        if node[0] == 'print':
            if node[1][0] == '"':
                print(node[1][1:len(node[1])-1])
            else:
                return self.walkTree(node[1])
```

Fungsi len() berfungsi untuk mengembalikan panjang (jumlah anggota) dari suatu objek.

```
        if node[0] == 'if_stmt':
            result = self.walkTree(node[1])
            if result:
                return self.walkTree(node[2][1])
            return self.walkTree(node[2][2])
```

```
        if node[0] == 'condition_eqeq':
            return self.walkTree(node[1]) == self.walkTree(node[2])
```

#Mendefinisikan Python

```
        if node[0] == 'fun_def':
            self.env[node[1]] = node[2]

        if node[0] == 'fun_call':
            try:
                return self.walkTree(self.env[node[1]])
            except LookupError:
                print("Undefined function '%s'" % node[1])
                return 0
```

Kode di atas menjelaskan cara mendefinisikan fungsi dengan Python. Ia memiliki kata kunci def untuk memberi tahu Python bahwa nama berikutnya adalah nama fungsi. Fungsi ini merupakan sintaksis atau perintah untuk membuat suatu fungsi dengan parameter di dalamnya.

```
        if node[0] == 'add':
            return self.walkTree(node[1]) + self.walkTree(node[2])
        elif node[0] == 'sub':
            return self.walkTree(node[1]) - self.walkTree(node[2])
        elif node[0] == 'mul':
            return self.walkTree(node[1]) * self.walkTree(node[2])
        elif node[0] == 'div':
            return int(self.walkTree(node[1]) / self.walkTree(node[2]))
```

Kode di atas menjelaskan cara mengoperasikan bilangan aritmatika mulai dari add hingga div.

```
    if node[0] == 'var_assign':
            self.env[node[1]] = self.walkTree(node[2])
            return node[1]

        if node[0] == 'var':
            try:
                return self.env[node[1]]
            except LookupError:
                print("Undefined variable '"+node[1]+"' found!")
                return 0
```

Variabel adalah wadah untuk menyimpan nilai data. Python tidak memiliki perintah untuk mendeklarasikan variabel. Variabel dibuat saat Anda pertama kali menetapkan nilai padanya. Variabel tidak perlu dideklarasikan dengan tipe tertentu, dan bahkan dapat diubah tipenya setelah ditetapkan.

```
        if node[0] == 'for_loop':
```

```
            if node[1][0] == 'for_loop_setup':
                loop_setup = self.walkTree(node[1])

                loop_count = self.env[loop_setup[0]]
                loop_limit = loop_setup[1]

                for i in range(loop_count+1, loop_limit+1):
                    res = self.walkTree(node[2])
                    if res is not None:
                        print(res)
                    self.env[loop_setup[0]] = i
                del self.env[loop_setup[0]]

        if node[0] == 'for_loop_setup':
            return (self.walkTree(node[1]), self.walkTree(node[2]))
```

Setelah selesai membuat walktree, langkah terakhir adalah membuat kode untuk menampilkan hasil dari eksekusi. Kode pertama-tama harus memanggil lexer, lalu parser, dan kemudian interpreter, dan akhirnya mengambil hasilnya. Outputnya kemudian ditampilkan ke shell.

```
if __name__ == '__main__':
    lexer = sundanase_lexer.SundanaseLexer()
    parser = sundanase_parser.SundanaseParser()
    env = {}
    while True:
        try:
            text = input('sundanase > ')
        except EOFError:
            break
        if text:
            tree = parser.parse(lexer.tokenize(text))
            SundanaseExecute(tree, env)
```

Perlu diketahui bahwa kami belum membuat handling eror apa pun. Jadi, SLY akan menampilkan pesan kesalahannya setiap kali kami melakukan sesuatu yang tidak ditentukan oleh aturan yang telah ditulis.

D. Main.py

Sama halnya dengan main2.py , kode pada main.py digunakan sebagai shell untuk menjalankan program. Tetapi, berbeda dengan main.py yang mana kita terlebih dahulu menjalankan shell kemudian baru mengetikkan kode yang kita inginkan. Untuk main.py digunakan untuk menjalankan file sundanase_languange dari sebuah file berekstensi ‘.sdn’ yang sudah diisi dengan kode yang kita ingin jalankan.

```
import sundanase_lexer
import sundanase_parser
import sundanase_interpreter

from sys import *

#MASUKKAN DENGAN SUNDANASE.SDN
lexer = sundanase_lexer.BasicLexer()
parser = sundanase_parser.BasicParser()
env = {}

file = open(argv[1])
text = file.readlines()
for line in text:
    tree = parser.parse(lexer.tokenize(line))
    sundanase_interpreter.BasicExecute(tree, env)
```

E. Main2.py

Kode di bawah berfungsi sebagai shell untuk menjalankan program yang akan di eksekusi setelah melalui proses dari lexer parser hingga interpreter

```
import sundanase_lexer
import sundanase_parser
import sundanase_interpreter

from sys import *

#MASUKAN LANGSUNG
if __name__ == '__main__':
    lexer = sundanase_lexer.SundanaseLexer()
    parser = sundanase_parser.SundanaseParser()
    env = {}
    while True:
        try:
            text = input('sundanase > ')
        except EOFError:
            break
        if text:
            tree = parser.parse(lexer.tokenize(text))
            sundanase_interpreter.SundanaseExecute(tree, env)
```

Modul sys di Python menyediakan berbagai fungsi dan variabel yang digunakan untuk memanipulasi berbagai bagian lingkungan runtime Python. Ini memungkinkan

pengoperasian pada interpreter karena menyediakan akses ke variabel dan fungsi yang berinteraksi kuat dengan interpreter.

2. Cara Pemakaian

2.1 Menggunakan CMD

Cara penggunaan melalui CMD sebagai berikut:

1. Buka CMD (windows + R → CMD), kemudian masuk ke directory dimana file sundanase-lang disimpan (...\Sundanase-lang).

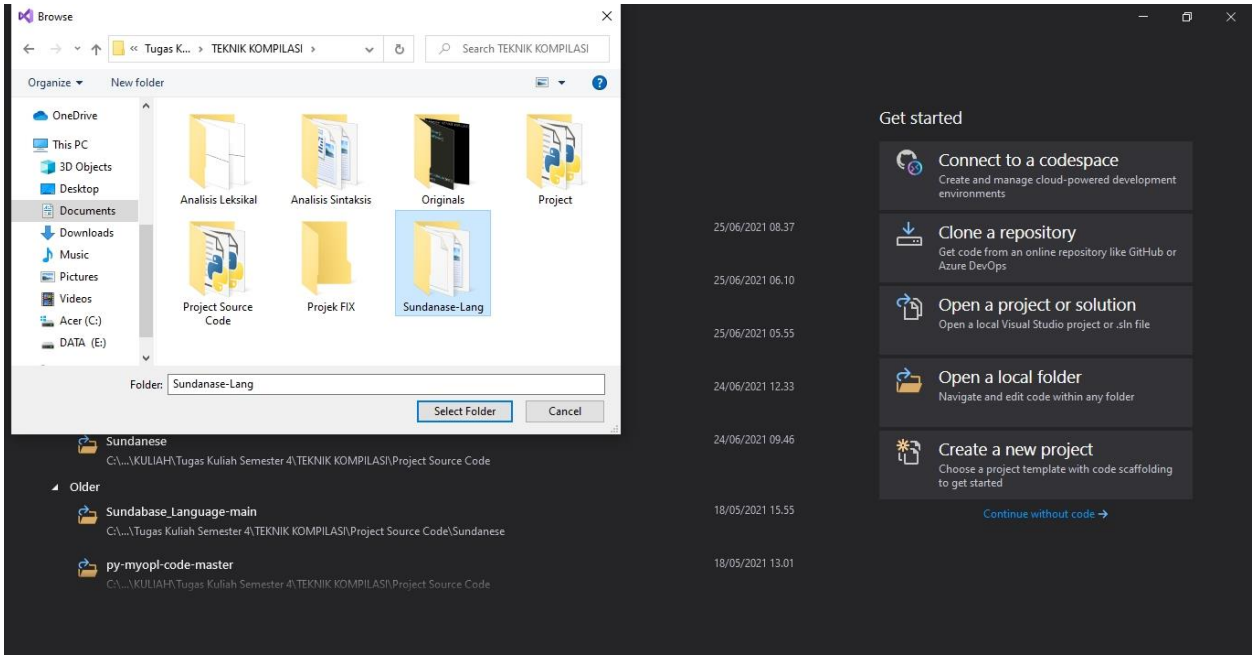
2. Kemudain jalankan file main.py nya (>python (spasi) main.py (spasi) nama file.sundanase/sdn

C:\Users\ACER 10\Documents\KULIAH\Tugas Kuliah Semester 4\TEKNIK KOMPILASI\Sundanase-Lang> pyhon main.py bahasaku.sundanase

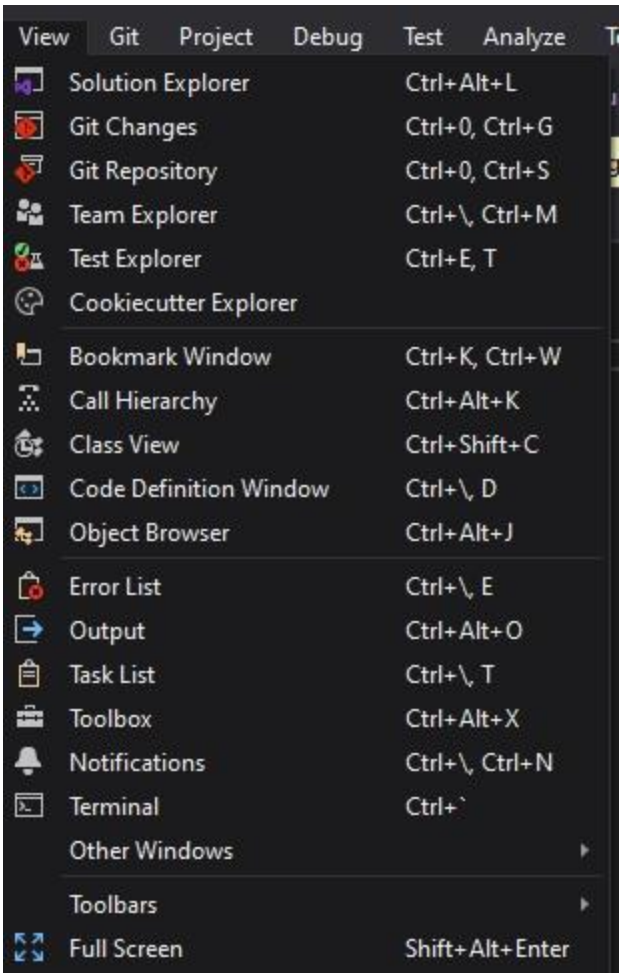
2.2 Menggunakan Microsoft Visual Studio

Cara penggunaan melalui MVS adalah sebagai berikut:

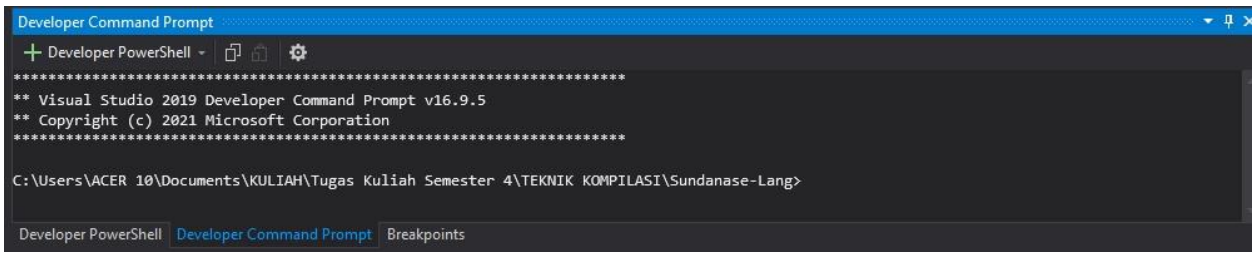
1. Buka Microsoft Visual Studio, kemudian masuk kedalam folder Sundanase-Lang.



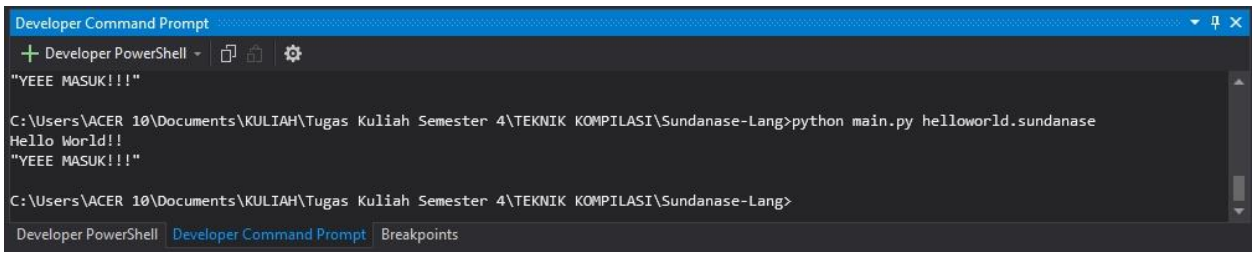
2. Buka terminal pada MVS, untuk tiap versi bisa berbeda, untuk kali ini di penulis melalui view lalu pilih terminal



Akan keluar tampilan seperti ini



3. Jalankan file dengan cara menulis >>Python (spasi) main.py (spasi) nama_file.sundanase/sdn



2.3 Bahasa Perintah Sundanase-Lang

2.3.1 Perintah PRINT

Perintah PRINT berfungsi untuk menampilkan atau mencetak hasil/result dari suatu perintah yang sebelumnya telah kita input. Ketika kita mengetik PRINT maka hasil akan tampak pada layar interface.

```
Sundanase > PRINT "HELLO WORLD!!!"
HELLO WORLD!!!
```

2.3.2 Komentar

Komentar pada Bahasa Sundanase dituliskan dengan tanda '#'. Komentar ini digunakan untuk menambahkan semacam note pada file program tanpa mengganggu program itu sendiri.

```
Sundanase > PRINT "HELLO WORLD!!!" #ini hanya komentar
HELLO WORLD!!!
```

2.3.3 Operasi Aritmatika

Dalam bahasa program Sundanase, terdapat fungsi aritmatika yang sudah mendukung tipe data integer dan juga float. Untuk operasinya sendiri ada penjumlahan, pengurangan, perkalian, pembagian, pemangkatan, dan juga Hasil sisa bagi. Pada operasi aritmatika program tidak harus menggunakan token sebagai identifier, tetapi bisa langsung digunakan dengan menulis angka dan jenis operasinya.

```
Sundanase > ((7+2)*3/9)^ 2
9.0
Sundanase > ((7+2)*3/9)
3.0
```

```
Sundanase > 4/8
0.5
Sundanase > 0.5*5
2.5
```

2.3.4 Perintah "UPAMI expr TERAS stmt1 HENTEU stmt2"

Pada perimtah UPAMI *expr* TERAS *stmt1* HENTEU *stmt2* dibutuhkan *expression* dan 2 *statement.*

- UPAMI adalah sintaksis atau perintah untuk menyatakan logika IF
- TERAS adalah sintaksis atau perintah untuk menyatakan logika THEN
- HENTEU adalah sintaksis atau perintah untuk menyatakan logika ELSE

```
Sundanase > a=5
Sundanase > b=10
Sundanase > UPAMI a>b TERAS PRINT "BENAR" HENTEU PRINT "SALAH"
SALAH
Sundanase >
```

2.3.5 Perintah “KAHATUR expr KANGGO stmt1 TERAS stmt2”

Pada perimtah KAHATUR *expr* KANGGO *stmt1* TERAS *stmt2* dibutuhkan *expression* dan 2 *statement.*

- KHATUR adalah sintaksis atau perintah untuk menyatakan logika FOR
- KANGGO adalah sintaksis atau perintah untuk menyatakan logika TO
- TERAS adalah sintaksis atau perintah untuk menyatakan logika THEN

```
Sundanase > KAHATUR a=1 KANGGO 5 TERAS PRINT a
1
2
3
4
```

2.3.6 Perintah “FUN functionName() → kode”

Sintaksis FUN merupakan perintah untuk fungsi. ‘Fungsi’ adalah sintaksis atau perintah untuk membuat suatu fungsi dengan parameter di dalamnya.

- FUN adalah sintaksis atau perintah untuk menyatakan logika FUN
- functionName() merupakan nama fungsi yang akan dibuat.

```
Sundanase > FUN ayomulai()-> PRINT "Jalan"
Sundanase > ayomulai()
Jalan
Sundanase >
```

DAFTAR PUSTAKA

GeeksforGeeks. 2020. “How to Create a Programming Language using Python?”. https://www.geeksforgeeks.org/how-to-create-a-programming-language-using-python/, diakses pada 5 Mei 2021.

GeeksforGeeks. 2020. “What does the if __name__ == “__main__”: do?”. https://www.geeksforgeeks.org/what-does-the-if-__name__-__main__-do/, diakses pada 5 Mei 2021.

GeeksforGeeks. 2021. “Binary Tree Data Structure”. https://www.geeksforgeeks.org/binary-tree-data-structure/ , diakses pada 5 Mei 2021.