CD PROJECT

# Implementation of Dwarf Java in Python

Team Members
1. Padmavathi Kadium
2. Sri Vasavi Chandu
3. Tiruveedhula Roshitha
4. Thota Pavan Kalyan
5. Tottempudi Sai Saran
6. Grandhi Manogna
7. Bhavana Vasana

## ABSTRACT

Programming languages are notations used to describe computations to humans and machines. Because all of the software running on all of the computers was written in some programming language, the world as we know it is dependent on programming languages. However, before a programme can be run, it must first be translated into a form that a computer can understand. Compilers are the software systems that perform this translation. A compiler is a piece of software that converts a high-level language (Source Language) programme to a low-level language (Object/Target/Machine Language). We all know that a computer is a logical combination of software and hardware. Because hardware understands a language that is difficult for us to understand, we tend to write programmes in high-level language, which is much easier for us to comprehend and remember. Hardware understands instructions in the form of electronic charge, which is the software programming equivalent of binary language. Binary has only two alphabets: 0 and 1. The hardware codes must be written in binary format, which is simply a series of 1s and 0s, in order to instruct. Writing such codes would be a difficult and time-consuming task for computer programmers, which is why we have compilers. These programmes are then fed into a series of tools and OS components to produce the desired code that the machine can use. This is referred to as a Language Processing System.

## INTRODUCTION

A compiler is a programme that can read a programme written in one language (the source language) and convert it to an equivalent programme written in another language (the target language). The compiler's important role is to report any errors in the source programme that it finds during the translation process. If the target programme is an executable machine-language programme, the user can invoke it to process inputs and generate outputs. Another type of language processor is an interpreter. An interpreter appears to directly execute the operations specified in the source programme on user inputs rather than producing a target programme as a translation. A compiler's machine-language target programme is usually much faster than an interpreter at mapping inputs to outputs. Because it executes the source programme statement by statement, an interpreter can usually provide better error diagnostics than a compiler. We all know that a computer is a logical combination of software and hardware. Because hardware understands a language that is difficult for us to understand, we tend to write programmes in high-level language, which is much easier for us to comprehend and remember. These programmes are now undergoing a series of transformations in order to be easily used machines. Language processing systems come in handy here. When we look at the compilation process more closely, we can see that it is a series of phases, each of which

transforms one representation of the source programme to another. The figure below depicts a typical decomposition of a compiler into phases. The analysis section divides the source programme into constituent parts and applies a grammatical structure to them. This structure is then used to generate an intermediate representation of the source programme. If the analysis part determines that the source programme is either syntactically or semantically incorrect, it must provide informative messages to the user so that corrective action can be taken. The analysis part also collects information about the source programme and stores it in a data structure called a symbol table, which is passed to the synthesis part along with the intermediate representation. The synthesis section creates the desired target programme from the intermediate representation and the symbol table information. The analysis part of the compiler is often referred to as the front end, while the synthesis part is referred to as the back end.

1. LEXICAL ANALYSIS

Lexical analysis or scanning is the first phase of a compiler. The lexical analyzer reads the source program's stream of characters and groups them into meaningful sequences known as lexemes. The lexical analyzer generates a token of the form for each lexeme as output (token-name, attribute-value).

2. SYNTAX ANALYSIS

The compiler's second phase is syntax analysis, also known as parsing. The parser constructs a tree-like intermediate representation of the token stream using the first components of the tokens produced by the lexical analyzer. A syntax tree is a common representation in which each interior node represents an operation and the children of the node represent the operation's arguments.

3. SEMANTIC ANALYSIS

The semantic analyzer checks the source programme for semantic consistency with the language definition using the syntax tree and the information in the symbol table. It also collects type information and stores it in the syntax tree or symbol table for later use during intermediate-code generation.

4. INTERMEDIATE CODE GENERATION

A compiler may create one or more intermediate representations, which can take various forms, while translating a source programme into target code. Syntax trees are a type of intermediate representation that is frequently used in syntax and semantic analysis.

5. CODE OPTIMIZATION

The machine-independent code-optimization phase attempts to improve the intermediate

code so that better target code results. Better usually means faster, but other goals may be desired, such as shorter code or target code that consumes less power.

6. CODE GENERATION
The code generator takes an intermediate representation of the source programme as input and converts it to the target language. If machine code is the target language, registers or memory locations are chosen for each variable used by the programme. The intermediate instructions are then translated into machine instruction sequences that perform the same task.

SYMBOL TABLE
A compiler's primary function is to record the variable names used in the source programme and collect information about the various attributes of each name.

## APPROACH

Dwarf Java is simply to define it as mini Java where we have 2 types of Data Types. They are Primitive Data Types and Non-Primitive Data Types.
1. PRIMITIVE DATA TYPES
a) Integers - Declared as unit
b) Decimals - Declared as denary
c) Characters - Declared as symbol, $ used for declaring it
d) Boolean declared as boolean, and boolean values are Yes and No

2. NON-PRIMITIVE DATA TYPES
a) Strings are declared as string, # is used to declare it.
b) Arrays are declared as []

## DECLARATIONS

1. Print statement - return()
2. If Loop - if <condition>
endif
3. For Loop - for <statement1 to statement2> do
endfor
4. While Loop - while <condition>
endwhile
5. Arrays - unit/denary variable[size]; (After array declaration use **;**)
6. Operators - <+, -, *, /, <=, >=, ==, >, <, ++, - ->\
7. Logical Operators - and & or

**PROGRAM SYNTAX**

```
unit/denary main()
begin
…….
……..
…….
end
```

**SOURCE CODE CONSIDERED**

```
unit main()
begin
unit L[10];
unit maxval=L[0];
for i=1 to n-1 do
if L[i]>maxval
maxval=L[i];
endif
endfor
return(maxval)
end
```

## 1. IMPLEMENTATION OF LEXICAL ANALYSIS

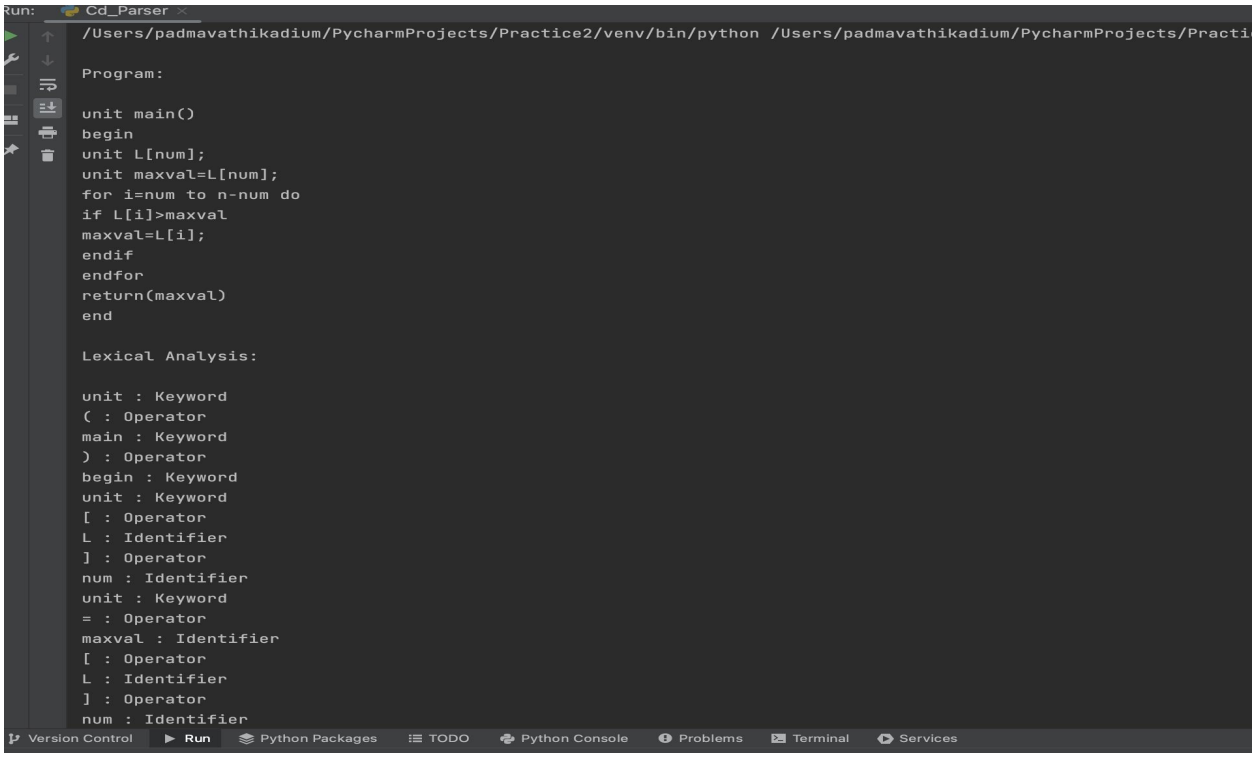
## 1.1) CODE

```
def lexical_analyser():
    keywords = ['unit', 'main', 'begin', 'for', 'to', 'do', 'if', 'endif', 'endfor', 'return', 'End']
    operators = ['(', ')', '[', ']', '=', '-', '>']
```

```
w = ""

    if os.stat("/Users/padmavathikadium/Desktop/question").st_size == 0:
        print("File is empty")
    else:
        with open('/Users/padmavathikadium/Desktop/question', 'r') as f:
            for line in f:
                for word in line:
                    for character in word:
                        if character.isspace() or character == ';':
                            if w == "":
                                continue
                            elif w in keywords:
                                print("%s : Keyword" % w)
                            else:
                                print("%s : Identifier" % w)
                            w = ""
                        elif character in operators:
                            print("%s : Operator" % character)
                            if w == "":
                                continue
                            elif w in keywords:
                                print("%s : Keyword" % w)
                            else:
                                print("%s : Identifier" % w)
                            w = ""
                        else:
                            w = w + character
```

## 1.2) OUTPUT


Run: Cd_Parser
/Users/padmavathikadium/PycharmProjects/Practice2/venv/bin/python /Users/padmavathikadium/PycharmProjects/Practi
Program:
unit main()
begin
unit L[num];
unit maxval=L[num];
for i=num to n-num do
if L[i]>maxval
maxval=L[i];
endif
endfor
return(maxval)
end
Lexical Analysis:
unit : Keyword
( : Operator
main : Keyword
) : Operator
begin : Keyword
unit : Keyword
[ : Operator
L : Identifier
] : Operator
num : Identifier
unit : Keyword
= : Operator
maxval : Identifier
[ : Operator
L : Identifier
] : Operator
num : Identifier
Version Control
Run
Python Packages
TODO
Python Console
Problems
Terminal
Services


Run: Cd_Parser
= : Operator
maxval : Identifier
[ : Operator
L : Identifier
] : Operator
num : Identifier
for : Keyword
= : Operator
i : Identifier
num : Identifier
to : Keyword
- : Operator
n : Identifier
num : Identifier
do : Keyword
if : Keyword
[ : Operator
L : Identifier
] : Operator
i : Identifier
> : Operator
maxval : Identifier
= : Operator
maxval : Identifier
[ : Operator
L : Identifier
] : Operator
i : Identifier
endif : Keyword
endfor : Keyword
( : Operator
return : Keyword
) : Operator
maxval : Identifier
Version Control
Run
Python Packages
TODO
Python Console
Problems
Terminal
Services

### 1.3) LEXICAL ANALYSIS TABLE

| Keyword | Operator | Identifier |
|---|---|---|
| unit | ( | L |
| main | ) | num |
| begin | [ | maxval |
| for | ] | i |
| to | > | n |
| do | = | |
| if | - | |
| endif | | |
| endfor | | |
| return | | |
| end | | |

## 2. IMPLEMENTATION OF SYNTAX ANALYSIS

### 2.1) TOKENIZATION

Tokenization is the process of separating a string sequence into tokens such as words, keywords, phrases, symbols, and other elements. Individual words, phrases, or even entire sentences can be used as tokens. Some characters, such as punctuation marks, are discarded during the tokenization process.

### 2.3) TOKENS USED

| | |
|---|---|
| i -> int | t -> to |
| s -> ‘ ‘ (space) | k -> n |
| m -> main | p -> - |
| a -> ( | d -> do |
| b -> ) | r -> if |
| n -> ‘\n’ (newline) | u -> > |
| e -> begin | x -> endif |
| l -> L | y -> endfor |
| g -> [ | z -> return |
| h -> ] | w -> end |
| o -> num | |
| c -> ; | |
| v -> maxval | |
| q -> = | |
| f -> for | |
| j -> i | |

## 2.2) CONTEXT FREE GRAMMAR


## 2.2) CODE

```
from ourrules import Consts
import sys
import os
# Lexical Analyser
def lexical_analyser():
        keywords = ['unit', 'main', 'begin', 'for', 'to', 'do', 'if', 'endif',   'endfor', 'return', 'End']
operators = ['(', ')', '[', ']', '=', '-', '>']
        w = ""
```

```
  if os.stat("/Users/padmavathikadium/Desktop/question").st_size == 0:
     print("File is empty")
  else:
     with open('/Users/padmavathikadium/Desktop/question', 'r') as f:
        for line in f:
           for word in line:
              for character in word:
                 if character.isspace() or character == ';':
                    if w == "":
                       continue
                    elif w in keywords:
                       print("%s : Keyword" % w)
                    else:
                       print("%s : Identifier" % w)
                    w = ""
                 elif character in operators:
                    print("%s : Operator" % character)
                    if w == "":
                       continue
                    elif w in keywords:
                       print("%s : Keyword" % w)
                    else:
                       print("%s : Identifier" % w)
                    w = ""
                 else:
                    w = w + character


# Tokenizer
def tokenizer(program_string):
   tokens = Consts.tokens
   ip = 0
   program_string += '$'
   token_string = ""
   while program_string[ip] != '$':
      current_token = ""
      if program_string[ip].isalpha() or program_string[ip] == '_':
         current_token += program_string[ip]
         ip += 1
         while program_string[ip].isalnum() or program_string[ip] == '_':
            current_token += program_string[ip]
            ip += 1
         if current_token in tokens.keys():
            token_string += tokens[current_token]
         else:
            token_string += tokens['var']
      elif program_string[ip].isnumeric():
```

```
            current_token += program_string[ip]
            ip += 1
            while program_string[ip].isnumeric() or program_string[ip] == '.':
                current_token += program_string[ip]
                ip += 1
            token_string += tokens['num']
        else:
            if program_string[ip] in tokens:
                token_string += tokens[program_string[ip]]
                ip += 1
            else:
                nl_count = 1
                pointer_count = 0
                for _ in range(ip):
                    if program_string[_] == '\n':
                        nl_count += 1
                        pointer_count = 0
                    else:
                        pointer_count += 1
                print("Tokenizer: Error on line " + str(nl_count) + " on column " + str(pointer_count))

                exit()
    return token_string


# # Main function
def main():
    file = open("/Users/padmavathikadium/Desktop/question", "r")
    program_string = file.read()
    print("\nProgram:\n")
    print(program_string)
    print("\nLexical Analysis:\n")
    lexical_analyser()
    print("\nTokens Generated:")
    # Tokenizing
    token_string = tokenizer(program_string)
    print(token_string)


main()

class Consts(object):
    # Tokens

    # 'var' is for a variable and 'num' is for a number
    tokens = {
        'unit': 'i',
```

```
    ' ': 's',
    'main': 'm',
    '(': 'a',
    ')': 'b',
    '\n': 'n',
    'begin': 'e',
    'L': 'l',
    '[': 'g',
    ']': 'h',
    'end': 'w',
    ';': 'c',
    'for': 'f',
    'maxval': 'v',
    '=': 'q',
    '-': 'p',
    '>': 'u',
    'i': 'j',
    'to': 't',
    'num': 'o',
    'n': 'k',
    'do': 'd',
    'if': 'r',
    'endif': 'x',
    'endfor': 'y',
    'return': 'z'

}

# Rules
rules = [
    ['P', 'S'],
    ['S', 'ismabnT'],
    ['T', 'enislgohcnU'],
    ['U', 'isvqlgohcnV'],
    ['V', 'fsjqostskposdnW'],
    ['W', 'rslgjhuvnX'],
    ['X', 'vqlgjhcnxnY'],
    ['Y', 'ynzavbnwn'],


]
```

## 2.3) OUTPUT

```
Tokens Generated:
ismabnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw


Process finished with exit code 0
```

## 2.4) PARSING FOR THE GRAMMAR

We used LL(1) parser for this,

FIRST -
1. P = {i}
2. S = {i}
3. T = {e}
4. U = {l}
5. V = {f}
6. W = {r}
7. X = {v}
8. Y = {y}

FOLLOW-
1. P = {$}
2. S = {$}
3. T = {$}
4. U = {$}
5. V = {$}
6. W = {$}
7. X = {$}
8. Y = {$}

## PARSING TABLE

| | i | s | m | a | b | n | e | l | g | o | h | c | v | q | f | j | t | k | p | d | r | u | x | y | z | w | $ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | P->S | | | | | | | | | | | | | | | | | | | | | | | | | | Sync |
| S | S->ismabnT | | | | | | | | | | | | | | | | | | | | | | | | | | Sync |
| T | | | | | | | T->enislgoshnU | | | | | | | | | | | | | | | | | | | | Sync |
| U | | | | | | | | U->lsvqllgohcnV | | | | | | | | | | | | | | | | | | | Sync |
| V | | | | | | | | | | | | | | | V->fsjqostskposdnW | | | | | | | | | | | | Sync |
| W | | | | | | | | | | | | | | | | | | | | | W->rslgjhuvnX | | | | | | Sync |
| X | | | | | | | | | | | | | X->vqlgjhcnxnY | | | | | | | | | | | | | | Sync |
| Y | | | | | | | | | | | | | | | | | | | | | | | | Y->ynzavbnwn | | | Sync |

## CONSIDERING A STRING

For Valid string considered is

**ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw**

| Matched | Stack | Input | Action |
|---|---|---|---|
| - | P$ | ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | - |
| - | S$ | ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | P~S |
| - | ismabnT$ | ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | S~ismabnT |
| i | smabnT$ | smabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched i |
| is | mabnT$ | mabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched s |
| ism | abnT$ | abnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched m |
| isma | bnT$ | bnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched a |
| ismab | nT$ | nenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched b |
| ismabn | T$ | enislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched n |
| ismabn | enislgohcnU$ | enislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | T~enislgohcnU |
| ismabne | nislgohcnU$ | nislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched e |
| ismabnen | islgohcnU$ | islgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched n |
| ismabneni | slgohcnU$ | slgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched i |
| ismabnenis | lgohcnU$ | lgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched s |
| ismabnenisl | gohcnU$ | gohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched l |
| ismabnenislg | ohcnU$ | ohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched g |
| ismabnenislgo | hcnU$ | hcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched o |
| ismabnenislgoh | cnU$ | cnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched h |
| ismabnenislgohc | nU$ | nlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched c |
| ismabnenislgohcn | U$ | lsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched n |
| ismabnenislgohcn | lsvqlgohcnV$ | lsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | U~lsvqlgohcnV |
| ismabnenislgohcnl | svqlgohcnV$ | svqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched l |
| ismabnenislgohcnls | vqlgohcnV$ | vqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched s |
| ismabnenislgohcnlsv | qlgohcnV$ | qlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched v |
| ismabnenislgohcnlsvq | lgohcnV$ | lgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched q |
| ismabnenislgohcnlsvql | gohcnV$ | gohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched l |
| ismabnenislgohcnlsvqlg | ohcnV$ | ohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched g |
| ismabnenislgohcnlsvqlgo | hcnV$ | hcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched o |
| ismabnenislgohcnlsvqlgoh | cnV$ | cnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched h |
| ismabnenislgohcnlsvqlgohc | nV$ | nfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched c |
| ismabnenislgohcnlsvqlgohcn | V$ | fsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched n |
| ismabnenislgohcnlsvqlgohcn | fsjqostskposdnW$ | fsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | V~fsjqostskposdnW |

| | | | |
|---|---|---|---|
| ismabnenislgohcnlsvqlgohcn | V$ | fsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched n |
| ismabnenislgohcnlsvqlgohcn | fsjqostskposdnW$ | fsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | V~fsjqostskposdnW |
| ismabnenislgohcnlsvqlgohcnf | sjqostskposdnW$ | sjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched f |
| ismabnenislgohcnlsvqlgohcnfs | jqostskposdnW$ | jqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched s |
| ismabnenislgohcnlsvqlgohcnfsj | qostskposdnW$ | qostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched j |
| ismabnenislgohcnlsvqlgohcnfsjq | ostskposdnW$ | ostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched q |
| ismabnenislgohcnlsvqlgohcnfsjqo | stskposdnW$ | stskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched o |
| ismabnenislgohcnlsvqlgohcnfsjqos | tskposdnW$ | tskposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched s |
| ismabnenislgohcnlsvqlgohcnfsjqost | skposdnW$ | skposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched t |
| ismabnenislgohcnlsvqlgohcnfsjqosts | kposdnW$ | kposdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched s |
| ismabnenislgohcnlsvqlgohcnfsjqostsk | posdnW$ | posdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched k |
| ismabnenislgohcnlsvqlgohcnfsjqostskp | osdnW$ | osdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched p |
| ismabnenislgohcnlsvqlgohcnfsjqostskpo | sdnW$ | sdnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched o |
| ismabnenislgohcnlsvqlgohcnfsjqostskpos | dnW$ | dnrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched s |
| ismabnenislgohcnlsvqlgohcnfsjqostskposd | nW$ | nrslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched d |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdn | W$ | rslgjhuvnvqlgjhcnxnynzavbnwn$ | Matched n |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdn | rslgjhuvnX$ | rslgjhuvnvqlgjhcnxnynzavbnwn$ | W~rslgjhuvnX |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnr | slgjhuvnX$ | slgjhuvnvqlgjhcnxnynzavbnwn$ | Matched r |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrs | lgjhuvnX$ | lgjhuvnvqlgjhcnxnynzavbnwn$ | Matched s |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrsl | gjhuvnX$ | gjhuvnvqlgjhcnxnynzavbnwn$ | Matched l |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslg | jhuvnX$ | jhuvnvqlgjhcnxnynzavbnwn$ | Matched g |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgj | huvnX$ | huvnvqlgjhcnxnynzavbnwn$ | Matched j |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjh | uvnX$ | uvnvqlgjhcnxnynzavbnwn$ | Matched h |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhu | vnX$ | vnvqlgjhcnxnynzavbnwn$ | Matched u |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuv | nX$ | nvqlgjhcnxnynzavbnwn$ | Matched v |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvn | X$ | vqlgjhcnxnynzavbnwn$ | Matched n |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvn | vqlgjhcnxnY$ | vqlgjhcnxnynzavbnwn$ | X~vqlgjhcnxnY |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnv | qlgjhcnxnY$ | qlgjhcnxnynzavbnwn$ | Matched v |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvq | lgjhcnxnY$ | lgjhcnxnynzavbnwn$ | Matched q |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvql | gjhcnxnY$ | gjhcnxnynzavbnwn$ | Matched l |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlg | jhcnxnY$ | jhcnxnynzavbnwn$ | Matched g |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgj | hcnxnY$ | hcnxnynzavbnwn$ | Matched j |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjh | cnxnY$ | cnxnynzavbnwn$ | Matched h |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhc | nxnY$ | nxnynzavbnwn$ | Matched c |
| ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcn | xnY$ | xnynzavbnwn$ | Matched n |

```
ismabnenislgohcnlsvqlgohcnfsjqostskposdn          W$          rslgjhuvnvqlgjhcnxnynzavbnwn$          Matched n
ismabnenislgohcnlsvqlgohcnfsjqostskposdn          rslgjhuvnX$         rslgjhuvnvqlgjhcnxnynzavbnwn$          W~rslgjhuvnX
ismabnenislgohcnlsvqlgohcnfsjqostskposdnr          slgjhuvnX$          slgjhuvnvqlgjhcnxnynzavbnwn$          Matched r
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrs          lgjhuvnX$           lgjhuvnvqlgjhcnxnynzavbnwn$          Matched s
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrsl          gjhuvnX$           gjhuvnvqlgjhcnxnynzavbnwn$          Matched l
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslg           jhuvnX$          jhuvnvqlgjhcnxnynzavbnwn$          Matched g
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgj          huvnX$           huvnvqlgjhcnxnynzavbnwn$          Matched j
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjh          uvnX$           uvnvqlgjhcnxnynzavbnwn$          Matched h
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhu          vnX$           vnvqlgjhcnxnynzavbnwn$          Matched u
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuv            nX$          nvqlgjhcnxnynzavbnwn$          Matched v
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvn           X$           vqlgjhcnxnynzavbnwn$          Matched n
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvn          vqlgjhcnxnY$           vqlgjhcnxnynzavbnwn$          X~vqlgjhcnxnY
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnv          qlgjhcnxnY$          qlgjhcnxnynzavbnwn$          Matched v
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvq          lgjhcnxnY$          lgjhcnxnynzavbnwn$          Matched q
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvql            gjhcnxnY$           gjhcnxnynzavbnwn$          Matched l
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlg           jhcnxnY$           jhcnxnynzavbnwn$          Matched g
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgj          hcnxnY$          hcnxnynzavbnwn$          Matched j
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjh          cnxnY$          cnxnynzavbnwn$          Matched h
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhc            nxnY$           nxnynzavbnwn$          Matched c
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcn           xnY$           xnynzavbnwn$          Matched n
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnx          nY$          nynzavbnwn$          Matched x
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxn          Y$          ynzavbnwn$          Matched n
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxn          ynzavbnwn$          ynzavbnwn$          Y~ynzavbnwn
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxny            nzavbnwn$           nzavbnwn$          Matched y
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnyn           zavbnwn$           zavbnwn$          Matched n
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynz          avbnwn$          avbnwn$          Matched z
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynza          vbnwn$          vbnwn$          Matched a
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzav            bnwn$           bnwn$          Matched v
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavb           nwn$           nwn$          Matched b
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbn          wn$          wn$          Matched n
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw          n$          n$          Matched w
ismabnenislgohcnlsvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnwn            $           $          Matched n
```

For Invalid string considered is
**ismabnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw**

| Matched | Stack | Input | Action |
|---|---|---|---|
| - | P$ | ismabnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | - |
| - | S$ | ismabnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | P -> S |
| - | ismabnT$ | ismabnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | S -> ismabnT |
| i | smabnT$ | smabnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched i |
| is | mabnT$ | mabnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched s |
| ism | abnT$ | abnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched m |
| Isma | bnT$ | bnenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched a |
| Ismab | nT$ | nenislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched b |
| ismabn | T$ | enislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched n |
| ismabn | enislgohcnU$ | enislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | T -> enislgohcnU |
| ismabne | nislgohcnU$ | nislgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched e |
| ismabnen | islgohcnU$ | islgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched n |
| ismabneni | slgohcnU$ | slgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched i |
| ismabnenis | lgohcnU$ | lgohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched s |
| ismabnenisl | gohcnU$ | gohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched l |
| ismabnenislg | ohcnU$ | ohcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched g |
| ismabnenislgo | hcnU$ | hcnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched o |
| ismabnenislgoh | cnU$ | cnisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched h |
| ismabnenislgohc | nU$ | nisvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched c |
| ismabnenislgohcn | U$ | isvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | Matched n |
| ismabnenislgohcn | $ | isvqlgohcnfsjqostskposdnrslgjhuvnvqlgjhcnxnynzavbnw$ | |

## INSIGHTS ON BUILDING DWARF JAVA

1. We have implemented dwarf java on an IOS environment using python language on pycharm interpreter.
3. We successfully implemented upto Parsing
4. The executed grammar and the parsing is mentioned
5. Code and the output are displayed in the report upto parsing.