



Report on

Mini-compiler for the "if-else" and "for" constructs of Python

*Submitted in partial fulfillment of the requirements for **Sem VI***

Compiler Design

Bachelor of Technology in Computer Science & Engineering

Submitted by:

Sneha Hegde	PES1201801157
Srishti Sachan	PES1201802126
Ojashvi Saxena	PES1201801254
Abhilash V	PES1201800238

Under the guidance of

Mahesh H. B.
Assistant Professor
PES University, Bengaluru

January – May 2021

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

TABLE OF CONTENTS

Chapter No.	Title	Page No.
1.	INTRODUCTION Sample Input Sample Output	03
2.	ARCHITECTURE OF LANGUAGE	05
3.	LITERATURE SURVEY	05
4.	CONTEXT FREE GRAMMAR	06
5.	DESIGN STRATEGY	09
6.	IMPLEMENTATION DETAILS	10
7.	SHORTCOMINGS AND FUTURE SCOPE	12
8.	SNAPSHOTS	13
9.	CONCLUSION	22

INTRODUCTION

This is the report for our compiler design project, a mini-compiler designed for the “if-else” and “for” constructs of Python, using the Lex and Yacc tools. Provided with an input file, the compiler generates tokens and builds a symbol table, based on the context free grammar in the Yacc file. An abstract syntax tree is also constructed by the parser, which is then used to create the three address code and quadruples in the intermediate representation phase. The intermediate representation is then optimised using several optimisation techniques.

Sample Input

```
1 a=1
2 for i in range(0,5):
3     if a==1:
4         a=a+1
5         print("a value set")
6     elif a==2:
7         b=15
8         print("b value set")
9     else:
10        print("nothing happened")
11
12
```

Sample Output

Tokens-

```
1 |
2 | -----Tokens-----
3 | 1 T_a T_Assign T_1 T_NL
4 | 2 T_For T_i T_In T_Range T_OP T_0 T_Comma T_5 T_CP T_Colon T_NL
5 | 3 T_INDENT T_If T_a T_EQ T_1 T_Colon T_NL
6 | 4 T_INDENT T_a T_Assign T_a T_Plus T_1 T_NL
7 | 5 T_IND T_Print T_OP T_"a value set" T_CP T_NL
8 | 6 T_DEDENT T_Elif T_a T_EQ T_2 T_Colon T_NL
9 | 7 T_INDENT T_b T_Assign T_15 T_NL
10 | 8 T_IND T_Print T_OP T_"b value set" T_CP T_NL
11 | 9 T_DEDENT T_Else T_Colon T_NL
12 | 10 T_INDENT T_Print T_OP T_"nothing happened" T_CP T_NL
13 | 11 T_DEDENT T_NL
14 | 12 T_NL
15 | 13 T_EOF
```

Symbol Table-

1					
2					
3					
4	Scope	Name	Data Type	Type	Declaration
5					Last Used Line
6	1		int	Constant	4
7	1	a	int	Identifier	4
8	1	0	int	Constant	2
9	1	5	int	Constant	2
10	1	2	int	Constant	6
11	1	15	int	Constant	7
12	1	b	int	Identifier	7
13	1	i	int	Identifier	2
14	1	T0	TempVarType	TempVar	-1
15	1	T3	TempVarType	TempVar	-1
16	1	L0	N/A	TempLabel	-1
17	1	T40	bool	TempVar	-1
18	1	L1	N/A	TempLabel	-1
19	1	T4	TempVarType	TempVar	-1
20	1	T41	bool	TempVar	-1
21	1	T6	TempVarType	TempVar	-1
22	1	T7	bool	TempVar	-1
23	1	T9	TempVarType	TempVar	-1
24	1	T10	int	TempVar	-1
25	1	T19	TempVarType	TempVar	-1
26	1	T20	bool	TempVar	-1
27	1	T21	TempVarType	TempVar	-1

TAC-

1					
2	T0 = 1				
3	a = T0				
4	T3 = 0				
5	i = T3				
6					
7	L0: T3 = i				
8	T3 = 0				
9	T40 = T3 >= T3				
10	If False T40 goto L1				
11	T3 = i				
12	T4 = 5				
13	T41 = T3 < T4				
14	If False T41 goto L1				
15	T0 = a				
16	T6 = 1				
17	T7 = T0 == T6				
18	If False T7 goto L0				
19	T0 = a				
20	T9 = 1				
21	T10 = T0 + T9				
22	a = T10				
23	goto L1				
24	L0: T0 = a				
25	T19 = 2				
26	T20 = T0 == T19				
27	If False T20 goto L0				
28	T21 = 15				
29	b = T21				
30	goto L1				
31	L0: L1: L1: goto L0				
32	L1:				

Quads (after all the optimisations)-

	Oper.	Arg1	Arg2	Res
168				
169 Lno.				
170				
171 2	=	0	-	T3
172 3	=	T3	-	i
173 4	Label	-	-	L0
174 5	=	i	-	T3
175 6	=	0	-	T3
176 7	>=	T3	T3	T40
177 8	If False	T40	-	L1
178 9	=	i	-	T3
179 15	==	1	1	T7
180 16	If False	T7	-	L0
181 21	goto	-	-	L1
182 22	Label	-	-	L0
183 25	==	1	2	T20
184 26	If False	T20	-	L0
185 29	goto	-	-	L1
186 30	Label	-	-	L0
187 31	Label	-	-	L1
188 32	Label	-	-	L1
189 33	goto	-	-	L0
190 34	Label	-	-	L1
191	-----			

ARCHITECTURE OF LANGUAGE

Python is a vast language. Some of the features we have implemented-

- “if-else” and “for” constructs (allows nesting)
- function definitions (allows nesting) and function calls
- import statements
- string, integer, boolean, list data types
- single and multiline comments
- print, break, pass, void return statements
- arithmetic, relational, logical, membership operations
- scoping

LITERATURE SURVEY

For the most part, we've used lex and yacc learning material provided to us by our professors, along with documentation found online, and from sites such as [GeeksforGeeks](#) and [Stack Overflow](#).

[A Guide to Lex & Yacc \(ncsu.edu\)](http://ncsu.edu/~cs311/lexyacc/)

CONTEXT FREE GRAMMAR

EXTENDED CONTEXT-FREE GRAMMAR

Key:

Terminals

Non-terminals

digit \longrightarrow [0-9]
number \longrightarrow digit+
identifier \longrightarrow [_a-zA-Z][_a-zA-Z0-9]* |
string \longrightarrow \"([^\n])*\" | '([^\n])*'
item \longrightarrow number | string
rhs \longrightarrow string | identifier | list | number | boolean_expression | arith_expr
list \longrightarrow [l2
l2 \longrightarrow item, l2 | item] |]
range_func \longrightarrow range(digit,digit)
relop \longrightarrow > | < | >= | <= | == | !=
arith_expr \longrightarrow arith_expr + term | arith_expr - term | term
term \longrightarrow term * factor | term / factor | factor
factor \longrightarrow identifier | (arith_expr) | number
boolean_expression \longrightarrow True | False | identifier | arith_expr relop arith_expr |
not boolean_expression | boolean_expression and boolean_expression |
boolean_expression or boolean_expression | (boolean_expression)
assign_stmt \longrightarrow identifier = rhs
print_stmt \longrightarrow print(p2
p2 \longrightarrow number, p2 | number) | identifier, p2 | identifier) | string, p2 | string)
import_stmt \longrightarrow import <MODULE_NAME>
basic_stmt \longrightarrow assign_stmt | pass | break | print_stmt
stmt_list \longrightarrow basic_stmt \n stmt_list | basic_stmt \n
block \longrightarrow \n <INDENT> stmts <DEDENT>
stmts \longrightarrow stmt_list | stmt_list compound_stmt | compound_stmt
compound_stmt \longrightarrow if_stmt | for_stmt

iterable \longrightarrow list | range_func
for_stmt \longrightarrow for identifier in iterable : block
if_stmt \longrightarrow if boolean_expression : block (elif boolean_expression : block)* [else : block]

CFG used in yacc-

StartDebugger : StartParse T_EndOfFile ;

StartParse : T_NL StartParse | finalStatements T_NL StartParse | finalStatements | ;

constant : T_Number
| T_String ;

term : T_ID
| constant
| list_index ;

list_index : T_ID T_OSB T_Number T_CSB ;

basic_stmt : pass_stmt
| break_stmt
| import_stmt
| assign_stmt
| arith_exp
| bool_exp
| print_stmt
| return_stmt;

pass_stmt : T_Pass ;

break_stmt : T_Break ;

import_stmt : T_Import T_ID ;

assign_stmt : T_ID T_Assign arith_exp
| T_ID T_Assign bool_exp
| T_ID T_Assign func_call
| T_ID T_Assign T_OSB call_args T_CSB ;

arith_exp : term
| arith_exp T_Plus arith_exp
| arith_exp T_Minus arith_exp
| arith_exp T_Mult arith_exp
| arith_exp T_Div arith_exp
| T_Minus arith_exp
| T_OP arith_exp T_CP ;

bool_exp : bool_term T_Or bool_term
| bool_term T_And bool_term
| arith_exp T_EQ arith_exp
| arith_exp T_GT arith_exp
| arith_exp T_LT arith_exp
| arith_exp T_GE arith_exp
| arith_exp T_LE arith_exp

```

    | arith_exp T_NE arith_exp
    | arith_exp T_In T_ID
    | bool_term ;

bool_term : bool_factor
    | T_True
    | T_False ;

bool_factor : T_Not bool_factor
    | T_OP bool_exp T_CP;

print_stmt : T_Print T_OP call_args T_CP ;

return_stmt : T_Return ;


finalStatements : basic_stmt
    | compd_stmt
    | func_def
    | func_call
    | error T_NL ;

compd_stmt : if_stmt
    | for_stmt ;

if_stmt : T_If bool_exp T_Colon start_block
    | T_If bool_exp T_Colon start_block elif_stmts ;

elif_stmts : else_stmt
    | T_Elif bool_exp T_Colon start_block elif_stmts ;

else_stmt : T_Else T_Colon start_block ;


iterable: T_ID
    | T_Range T_OP T_Number T_Comma T_Number T_CP ;

for_stmt : T_For T_ID T_In iterable T_Colon start_block ;


start_block : basic_stmt
    | T_NL T_INDENT finalStatements block ;

block : T_NL T_ND finalStatements block
    | T_NL end_block ;

end_block : T_DEDENT finalStatements
    | T_DEDENT
    | ;

```



```

args : T_ID args_list
      | ;

args_list : T_Comma T_ID args_list | ;

call_list : T_Comma term call_list | ;

call_args : T_ID call_list
           | T_Number call_list
           | T_String call_list
           | ;

func_def : T_Def T_ID T_OP args T_CP T_Colon start_block ;

func_call : T_ID T_OP call_args T_CP ;

```

DESIGN STRATEGY

SYMBOL TABLE CREATION

The symbol table uses two structures, STable and record, for storage.

Each STable variable corresponds to one scope (one symbol table per scope).

The record variables are individual symbol table entries.

STable has members which hold information related to the number of entries, pointers to these entries, the scope of the table, and the parent table.

The members of record have information about the name of the entry, the data type, the line number of declaration, and the line it was last used.

Scope is implemented for Python functions, using a hashing technique to maintain unique scopes (when indentation level is the same, but scope is different).

The information stored in the symbol table can then be used by subsequent phases.

INTERMEDIATE CODE GENERATION

In the parsing phase, an abstract syntax tree is created. In the CFG, the rules specify when and how to push symbols as nodes into the tree.

The compiler uses this syntax tree while creating the intermediate code in the intermediate representation phase. The three address code is generated, and stored in quadruple form.

CODE OPTIMIZATION

This phase implements four optimisation techniques on the quads created in the previous phase. They are-

- Strength Reduction (replaces expensive operations with cheaper ones)
- Constant Propagation (a constant assigned to a variable is substituted when the variable is encountered during compile time instead of runtime)

- Constant Folding (recognises and evaluates constant expressions at compile time rather than computing them at runtime)
- Dead Code Elimination (removes useless or unreachable code)

IMPLEMENTATION DETAILS

SYMBOL TABLE CREATION

The structures used-

//structure of a single record in a symbol table

```
typedef struct record
{
    char *type;
    char *name;
    char *datatype;
    int lineno_declared;
    int lastUseLine;
} record;
```

//structure of a symbol table

//there is one ST per scope

```
typedef struct STable
{
    int no; //index
    int noOfElements;
    int STscope;
    record *Elements;
    int Parent;    //gives position index of parent ST (in array of STs)
} STable;
```

INTERMEDIATE CODE GENERATION

The structures used-

//structure for Abstract Syntax Tree node

```
typedef struct ASTNode
{
    int nodeNo;
    char *NType;    //if the Node is an operator (specifies the operator. ex- +, -, etc)
    int opCount;    //number of operands (children) of the operator node
    struct ASTNode** NextLevel;
    record *id;    //if the Node is an identifier or a constant, this node needs to point to a record in a ST
} Node;
```

//structure for Quad

```
typedef struct Quad
{
    char *Op;    //operator
    char *A1;    //argument 1
    char *A2;    //argument 2
    char *R;     //result

    //can be used to mark a quad as redundant during dead code elimination
    int I;       //(initially holds index no. of the quad in the array)
} Quad;
```

CODE OPTIMIZATION

- **Strength Reduction-**
 - It iterates through all the quads. If a quad with two arguments is found, and the operator is multiplication (“*”), it converts the expression into a left shift operation. If the operator is division (“/”), it converts the expression into a right shift operation.
 - This only works when the second argument is a power of two.
- **Constant Propagation-**
 - It iterates through all the quads. If it finds a quad with only one argument, and the argument is of integer type, and the operator is “=”, this means that the expression is of a simple assignment type (ex- a=20). Call this quad1.
 - It captures the index of quad1, and iterates through the quads after it. If it finds that any of these quads have their arguments equal to the result of quad1 (ex- b=a+15), it replaces that argument with the argument of quad1 (ex- b=20+15)
- **Constant Folding-**
 - It iterates through all the quads. If a quad is found with two arguments and both the arguments are of integer type, it checks the operation of that quad.
 - Based on the operation (ex- “+”, “*”, “/”), it computes the expression, and the result is stored as one of the arguments of the quad, and the operation of the quad is changed to “=”.
 - ex- a=5+3 (quad’s arg1=5, arg2=3, operation=”+”) becomes a=8 (quad’s arg1=8, operation=”=”)
- **Dead Code Elimination-**
 - It iterates through all the quads. For each quad (quad1), it iterates through the remaining quads (quad2). If the result of quad1 is not encountered as either of the arguments of any quad2, this means that the result of quad1 is not used anywhere else, for any computation. This renders the result of quad1 as dead, and it marks quad1 as such.
 - ex- if we have s=30, and s is not being used anywhere else in the code, we can mark the quad for this assignment as dead, thus rejecting that quad.

ERROR HANDLING

The compiler checks to see if a referenced identifier has been declared in the scope. If not, it throws an error, saying that the identifier is being used before declaration.

It also checks to see if the iterable identifier in the for loop is of list type. If it's not, it says that the identifier is not indexable.

Any other syntax error is handled by displaying the line and column number of the point of error.

BUILDING AND RUNNING THE PROGRAM

- The lex file is called project_lex.l, and the yacc file is called project_yacc.y
- The commands to build the program are in a makefile-

```
a.out : lex.yy.c y.tab.c y.tab.h
      gcc lex.yy.c y.tab.c -g -ll -lm
```

```
y.tab.c : project_yacc.y
      yacc -Wno-yacc -dv project_yacc.y
```

```
lex.yy.c : project_lex.l
      lex project_lex.l
```

clean :

```
      rm -rf lex.yy.c y.tab.c y.tab.h a.out y.output Tokens.txt TAC.txt Quads.txt SymbolTable.txt
```

- Upon execution, the resulting token sequence, symbol table, three address code, and quads get stored in Tokens.txt, SymbolTable.txt, TAC.txt, and Quads.txt respectively.
- The executable file takes as input the testing file with the python code (if it's called sample.py)
- Commands to build and run-

```
make -f makefile.mk
./a.out < sample.py
make -f makefile.mk clean
```

SHORTCOMINGS AND FUTURE SCOPE

Future scope/Shortcomings-

- Extend range function to take 1 or 3 arguments (currently, it takes exactly 2)
- Extend for loop to include other iterables, for example- strings, tuples (currently, it takes only range and lists)
- Allow for indentation to be done using spaces (currently, it can only be done using tabs)
- Implement operator overloading, so that string operations may be done, too (currently, "+" and "*" only work with integer type arguments)
- Allow for returning values in functions (currently, only void returns can be done)
- Implement loop optimisation techniques

SNAPSHOTS

Test file 1-

- Input-

```
1 import mymodule
2
3 #to demo scope
4 def foo():
5     def bar():
6         q=False
7         r="stringify"
8         def baz():
9             print(p,"function")
10
11
12
13
14 foo()
15
16 #this is a comment
17
18 #demo dead code elim
19 x=True
20 z="this is a string"
21
22 #demo strength reduction
23 c=3*4
24
25 #demo constant prop
26 s=18
27 t=s+s
28
29 #demo const folding
30 u=100+15
31
32
33
34 a=5
35 b=10
36
37 if (a==5) and (b>8):
38     a=9
39     b=20
40     print("inside if")
41 elif a<4:
42     print("inside elif")
43 else:
44     pass
45
46
47 for item1 in range(0,2):
48     for item2 in range(0,4):
49         for item3 in range(0,6):
50             e=9
51             for item4 in range(0,8):
52                 f=40
53                 for item5 in range(0,5):
54                     print("nested loop")
55
```

- Output-

```
sneha@sneha-VirtualBox:~/Desktop/cd_proj/phase2$ ./a.out < input1.py
```

```
-----
Valid Python Syntax!
-----
```

```
sneha@sneha-VirtualBox:~/Desktop/cd_proj/phase2$ █
```

Tokens-

```
1 |
2 |-----Tokens-----
3 | 1 T_Import T_mymodule T_NL
4 | 2 T_NL
5 | 3 T_NL
6 | 4 T_Def T_foo T_OP T_CP T_Colon T_NL
7 | 5 T_INDENT T_Def T_bar T_OP T_CP T_Colon T_NL
8 | 6 T_INDENT T_q T_Assign T_False T_NL
9 | 7 T_ND T_r T_Assign T_"stringify" T_NL
10 | 8 T_ND T_Def T_baz T_OP T_CP T_Colon T_NL
11 | 9 T_INDENT T_Print T_OP T_p T_Comma T_"function" T_CP T_NL
12 | 10 T_DEDENT T_NL
13 | 11 T_DEDENT T_NL
14 | 12 T_NL
15 | 13 T_NL
16 | 14 T_foo T_OP T_CP T_NL
17 | 15 T_NL
18 | 16 T_NL
19 | 17 T_NL
20 | 18 T_NL
21 | 19 T_x T_Assign T_True T_NL
22 | 20 T_z T_Assign T_"this is a string" T_NL
23 | 21 T_NL
24 | 22 T_NL
25 | 23 T_c T_Assign T_3 T_Mult T_4 T_NL
26 | 24 T_NL
27 | 25 T_NL
28 | 26 T_s T_Assign T_18 T_NL
29 | 27 T_t T_Assign T_s T_Plus T_s T_NL
30 | 28 T_NL
31 | 29 T_NL
32 | 30 T_u T_Assign T_100 T_Plus T_15 T_NL
33 | 31 T_NL
34 | 32 T_NL
35 | 33 T_NL
36 | 34 T_a T_Assign T_5 T_NL
37 | 35 T_b T_Assign T_10 T_NL
38 | 36 T_NL
39 | 37 T_If T_OP T_a T_EQ T_5 T_CP T_And T_OP T_b T_GT T_8 T_CP T_Colon T_NL
40 | 38 T_INDENT T_a T_Assign T_9 T_NL
41 | 39 T_ND T_b T_Assign T_20 T_NL
42 | 40 T_ND T_Print T_OP T_"inside if" T_CP T_NL
43 | 41 T_Elif T_a T_LT T_4 T_Colon T_NL
44 | 42 T_INDENT T_Print T_OP T_"inside elif" T_CP T_NL
45 | 43 T_Else T_Colon T_NL
46 | 44 T_INDENT T_Pass T_NL
47 | 45 T_NL
48 | 46 T_NL
49 | 47 T_For T_item1 T_In T_Range T_OP T_0 T_Comma T_2 T_CP T_Colon T_NL
50 | 48 T_INDENT T_For T_item2 T_In T_Range T_OP T_0 T_Comma T_4 T_CP T_Colon T_NL
51 | 49 T_INDENT T_For T_item3 T_In T_Range T_OP T_0 T_Comma T_6 T_CP T_Colon T_NL
52 | 50 T_INDENT T_e T_Assign T_9 T_NL
53 | 51 T_ND T_For T_item4 T_In T_Range T_OP T_0 T_Comma T_8 T_CP T_Colon T_NL
54 | 52 T_INDENT T_f T_Assign T_40 T_NL
55 | 53 T_ND T_For T_item5 T_In T_Range T_OP T_0 T_Comma T_5 T_CP T_Colon T_NL
56 | 54 T_INDENT T_Print T_OP T_"nested loop" T_CP T_NL
57 | 55 T_DEDENT T_NL
58 | 56 T_DEDENT T_NL
59 | 57 T_DEDENT T_NL
60 | 58 T_DEDENT T_NL
61 | 59 T_NL
62 | 60 T_EOF
```

Symbol Table-

		Symbol Table				
Scope	Name	Data Type	Type	Declaration	Last Used Line	
6 1	mymodule	N/A	ModuleName	1	1	
7 1	foo	func	Func. Name	4	4	
8 1	True	bool	Constant	19	19	
9 1	x	bool	Identifier	19	19	
10 1	"this is a string"	str	Constant	20	20	
11 1	z	str	Identifier	20	20	
12 1	3	int	Constant	23	23	
13 1	4	int	Constant	48	48	
14 1	c	int	Identifier	23	23	
15 1	18	int	Constant	26	26	
16 1	s	int	Identifier	26	27	
17 1	t	int	Identifier	27	27	
18 1	100	int	Constant	30	30	
19 1	15	int	Constant	30	30	
20 1	u	int	Identifier	30	30	
21 1	5	int	Constant	53	53	
22 1	a	int	Identifier	38	41	
23 1	10	int	Constant	35	35	
24 1	b	int	Identifier	39	39	
25 1	8	int	Constant	51	51	
26 1	9	int	Constant	50	50	
27 1	20	int	Constant	39	39	
28 1	0	int	Constant	53	53	
29 1	2	int	Constant	47	47	
30 1	6	int	Constant	49	49	
31 1	e	int	Identifier	50	50	
32 1	40	int	Constant	52	52	
33 1	f	int	Identifier	52	52	
34 1	item5	int	Identifier	53	53	
35 1	item4	int	Identifier	51	51	
36 1	item3	int	Identifier	49	49	
37 1	item2	int	Identifier	48	48	
38 1	item1	int	Identifier	47	47	
39 1	T4	Temp/VarType	Temp/Var	-1	-1	
40 1	T7	Temp/VarType	Temp/Var	-1	-1	
41 1	T38	func	Temp/Var	-1	-1	
42 1	T31	Temp/VarType	Temp/Var	-1	-1	
43 1	T34	Temp/VarType	Temp/Var	-1	-1	
44 1	T37	Temp/VarType	Temp/Var	-1	-1	
45 1	T38	Temp/VarType	Temp/Var	-1	-1	
46 1	T39	int	Temp/Var	-1	-1	
47 1	T42	Temp/VarType	Temp/Var	-1	-1	
48 1	T47	int	Temp/Var	-1	-1	
49 1	T50	Temp/VarType	Temp/Var	-1	-1	
50 1	T51	Temp/VarType	Temp/Var	-1	-1	
51 1	T52	int	Temp/Var	-1	-1	
52 1	T55	Temp/VarType	Temp/Var	-1	-1	
53 1	T58	Temp/VarType	Temp/Var	-1	-1	
54 1	T62	Temp/VarType	Temp/Var	-1	-1	
55 1	T63	bool	Temp/Var	-1	-1	
56 1	T65	Temp/VarType	Temp/Var	-1	-1	
57 1	T66	bool	Temp/Var	-1	-1	
58 1	T67	bool	Temp/Var	-1	-1	
59 1	L0	N/A	TempLabel	-1	-1	
60 1	T68	Temp/VarType	Temp/Var	-1	-1	
61 1	T71	Temp/VarType	Temp/Var	-1	-1	
62 1	L1	N/A	TempLabel	-1	-1	
63 1	T81	Temp/VarType	Temp/Var	-1	-1	
64 1	T82	bool	Temp/Var	-1	-1	
65 1	T93	Temp/VarType	Temp/Var	-1	-1	
66 1	L4	N/A	TempLabel	-1	-1	
67 1	T145	bool	Temp/Var	-1	-1	
68 1	L5	N/A	TempLabel	-1	-1	
69 1	T94	Temp/VarType	Temp/Var	-1	-1	
70 1	T146	bool	Temp/Var	-1	-1	
71 1	T95	Temp/VarType	Temp/Var	-1	-1	
72 1	T138	bool	Temp/Var	-1	-1	
73 1	T96	Temp/VarType	Temp/Var	-1	-1	
74 1	T139	bool	Temp/Var	-1	-1	
75 1	T97	Temp/VarType	Temp/Var	-1	-1	
76 1	T131	bool	Temp/Var	-1	-1	
77 1	T98	Temp/VarType	Temp/Var	-1	-1	
78 1	T132	bool	Temp/Var	-1	-1	
79 1	T99	Temp/VarType	Temp/Var	-1	-1	
80 1	T102	Temp/VarType	Temp/Var	-1	-1	
81 1	T123	bool	Temp/Var	-1	-1	
82 1	T103	Temp/VarType	Temp/Var	-1	-1	
83 1	T124	bool	Temp/Var	-1	-1	
84 1	T104	Temp/VarType	Temp/Var	-1	-1	
85 1	T107	Temp/VarType	Temp/Var	-1	-1	
86 1	T115	bool	Temp/Var	-1	-1	
87 1	T108	Temp/VarType	Temp/Var	-1	-1	
88 1	T116	bool	Temp/Var	-1	-1	
89 4	bar	func	Func. Name	5	5	
90 9	False	bool	Constant	6	6	
91 9	q	bool	Identifier	6	6	
92 9	"stringify"	str	Constant	7	7	
93 9	r	str	Identifier	7	7	
94 9	baz	func	Func. Name	8	8	
95 16	"function"	str	Constant	9	9	

TAC-

```
1 |-----Three Address Code-----
2 import mymodule
3 Begin Function foo
4 Begin Function bar
5 T4 = False
6 q = T4
7 T7 = "stringify"
8 r = T7
9 Begin Function baz
10 End Function baz
11 End Function bar
12 End Function foo
13 (T30)Call Function foo
14 T31 = True
15 x = T31
16 T34 = "this is a string"
17 z = T34
18 T37 = 3
19 T38 = 4
20 T39 = T37 * T38
21 c = T39
22 T42 = 18
23 s = T42
24 T42 = s
25 T42 = s
26 T47 = T42 + T42
27 t = T47
28 T50 = 100
29 T51 = 15
30 T52 = T50 + T51
31 u = T52
32 T55 = 5
33 a = T55
34 T58 = 10
35 b = T58
36 T55 = a
37 T62 = 5
38 T63 = T55 == T62
39 T58 = b
40 T65 = 8
41 T66 = T58 > T65
42 T67 = T63 and T66
43 If False T67 goto L0
44 T68 = 9
45 a = T68
46 T71 = 20
47 b = T71
48 goto L1
49 L0: T55 = a
50 T81 = 4
51 T82 = T55 < T81
52 If False T82 goto L0
53 goto L1
54 L0: L1: L1: T93 = 0
55 item1 = T93
56
57 L4: T93 = item1
58 T93 = 0
59 T145 = T93 >= T93
60 If False T145 goto L5
61 T93 = item1
62 T94 = 2
63 T146 = T93 < T94
64 If False T146 goto L5
65 T95 = 0
66 item2 = T95
67
68 L4: T95 = item2
69 T95 = 0
70 T138 = T95 >= T95
71 If False T138 goto L5
72 T95 = item2
73 T96 = 4
74 T139 = T95 < T96
75 If False T139 goto L5
76 T97 = 0
77 item3 = T97
78
79 L4: T97 = item3
80 T97 = 0
81 T131 = T97 >= T97
82 If False T131 goto L5
83 T97 = item3
84 T98 = 6
85 T132 = T97 < T98
86 If False T132 goto L5
87 T99 = 9
88 e = T99
89 T102 = 0
90 item4 = T102
91
92 L4: T102 = item4
93 T102 = 0
94 T123 = T102 >= T102
95 If False T123 goto L5
96 T102 = item4
97 T103 = 8
98 T124 = T102 < T103
99 If False T124 goto L5
100 T104 = 40
101 f = T104
102 T107 = 0
103 item5 = T107
104
105 L4: T107 = item5
106 T107 = 0
107 T115 = T107 >= T107
108 If False T115 goto L5
109 T107 = item5
110 T108 = 5
111 T116 = T107 < T108
112 If False T116 goto L5
113 goto L4
114 L5: goto L4
115 L5: goto L4
116 L5: goto L4
117 L5: goto L4
118 L5:
```


Quads (before optimisation)-

-----Quadruples-----				
Ln.	Oper.	Arg1	Arg2	Res
7 0	import	mymodule	-	-
8 1	BeginF	foo	-	-
9 2	BeginF	bar	-	-
10 3	=	False	-	T4
11 4	=	T4	-	q
12 5	=	"stringify"	-	T7
13 6	=	T7	-	r
14 7	BeginF	baz	-	-
15 8	EndF	baz	-	-
16 9	EndF	bar	-	-
17 10	EndF	foo	-	-
18 11	Call	foo	-	T30
19 12	=	True	-	T31
20 13	=	T31	-	x
21 14	=	"this is a string"	-	T34
22 15	=	T34	-	z
23 16	=	3	-	T37
24 17	=	4	-	T38
25 18	*	T37	T38	T39
26 19	=	T39	-	c
27 20	=	10	-	T42
28 21	=	T42	-	s
29 22	=	s	-	T42
30 23	=	s	-	T42
31 24	+	T42	T42	T47
32 25	=	T47	-	t
33 26	=	100	-	T50
34 27	=	15	-	T51
35 28	+	T50	T51	T52
36 29	=	T52	-	u
37 30	=	5	-	T55
38 31	=	T55	-	a
39 32	=	10	-	T58
40 33	=	T58	-	b
41 34	=	a	-	T55
42 35	=	5	-	T62
43 36	==	T55	T62	T63
44 37	=	b	-	T58
45 38	=	8	-	T65
46 39	>	T58	T65	T66
47 40	and	T63	T66	T67
48 41	If False	T67	-	L0
49 42	=	9	-	T68
50 43	=	T68	-	a
51 44	=	20	-	T71
52 45	=	T71	-	b
53 46	goto	-	-	L1
54 47	Label	-	-	L0
55 48	=	a	-	T55
56 49	=	4	-	T81
57 50	<	T55	T81	T82
58 51	If False	T82	-	L0
59 52	goto	-	-	L1
60 53	Label	-	-	L0
61 54	Label	-	-	L1
62 55	Label	-	-	L1
63 56	=	0	-	T93
64 57	=	T93	-	item1
65 58	Label	-	-	L4
66 59	=	item1	-	T93
67 60	=	0	-	T93
68 61	>=	T93	T93	T145
69 62	If False	T145	-	L5
70 63	=	item1	-	T93
71 64	=	2	-	T94
72 65	<	T93	T94	T146
73 66	if false	T146	-	L5
74 67	=	0	-	T95
75 68	=	T95	-	item2
76 69	Label	-	-	L4
77 70	=	item2	-	T95
78 71	=	0	-	T95
79 72	>=	T95	T95	T138
80 73	If False	T138	-	L5
81 74	=	item2	-	T95
82 75	=	4	-	T96
83 76	<	T95	T96	T139
84 77	if false	T139	-	L5
85 78	=	0	-	T97
86 79	=	T97	-	item3
87 80	Label	-	-	L4
88 81	=	item3	-	T97
89 82	=	0	-	T97
90 83	>=	T97	T97	T131
91 84	If False	T131	-	L5
92 85	=	item3	-	T97
93 86	=	0	-	T98
94 87	<	T97	T98	T132
95 88	if false	T132	-	L5
96 89	=	9	-	T99
97 90	=	T99	-	e
98 91	=	0	-	T102
99 92	=	T102	-	item4
100 93	Label	-	-	L4
101 94	=	item4	-	T102
102 95	=	0	-	T102
103 96	>=	T102	T102	T123
104 97	If False	T123	-	L5
105 98	=	item4	-	T102
106 99	=	8	-	T103
107 100	<	T102	T103	T124
108 101	if false	T124	-	L5
109 102	=	40	-	T104
110 103	=	T104	-	f
111 104	=	0	-	T107
112 105	=	T107	-	item5
113 106	Label	-	-	L4
114 107	=	item5	-	T107
115 108	=	0	-	T107
116 109	>=	T107	T107	T115
117 110	If False	T115	-	L5
118 111	=	item5	-	T107
119 112	=	5	-	T108
120 113	<	T107	T108	T116
121 114	if false	T116	-	L5
122 115	goto	-	-	L4
123 116	Label	-	-	L5
124 117	goto	-	-	L4
125 118	Label	-	-	L5
126 119	goto	-	-	L4
127 120	Label	-	-	L5
128 121	goto	-	-	L4
129 122	Label	-	-	L5
130 123	goto	-	-	L4
131 124	Label	-	-	L5
132				
133				

Quads (after strength reduction)-

Quadruples after Strength Reduction				
Lno.	Oper.	Arg1	Arg2	Res
133				
134				
135				
136				
137				
138	import	mymodule	-	-
139	BeginF	foo	-	-
140	BeginF	bar	-	-
141	=	False	-	T4
142	=	T4	-	q
143	=	"stringify"	-	T7
144	=	T7	-	r
145	BeginF	baz	-	-
146	EndF	baz	-	-
147	EndF	bar	-	-
148	EndF	foo	-	-
149	Call	foo	-	T30
150	=	True	-	T31
151	=	T31	-	x
152	=	"this is a string"	-	T34
153	=	T34	-	z
154	=	3	-	T37
155	=	4	-	T38
156	<<	T37	2	T39
157	=	T39	-	c
158	=	18	-	T42
159	=	T42	-	s
160	=	s	-	T42
161	=	5	-	T42
162	+	T42	T42	T47
163	=	T47	-	t
164	=	100	-	T50
165	=	15	-	T51
166	+	T50	T51	T52
167	=	T52	-	u
168	=	5	-	T55
169	=	T55	-	a
170	=	10	-	T58
171	=	T58	-	b
172	=	a	-	T55
173	=	5	-	T62
174	==	T55	T62	T63
175	=	b	-	T58
176	=	8	-	T65
177	>	T58	T65	T66
178	and	T63	T66	T67
179	If False	T67	-	L0
180	=	9	-	T68
181	=	T68	-	a
182	=	20	-	T71
183	=	T71	-	b
184	goto	-	-	L1
185	Label	-	-	L0
186	=	a	-	T55
187	<	4	-	T81
188	=	T55	T81	T82
189	If False	T82	-	L0
190	goto	-	-	L1
191	Label	-	-	L0
192	Label	-	-	L1
193	Label	-	-	L1
194	=	0	-	T93
195	=	T93	-	item1
196	Label	-	-	L4
197	=	item1	-	T93
198	=	0	-	T93
199	>=	T93	T93	T145
200	If False	T145	-	L5
201	=	item1	-	T93
202	=	2	-	T94
203	<	T93	T94	T146
204	if false	T146	-	L5
205	=	0	-	T95
206	=	T95	-	item2
207	Label	-	-	L4
208	=	item2	-	T95
209	=	0	-	T95
210	>=	T95	T95	T138
211	If False	T138	-	L5
212	=	item2	-	T95
213	=	4	-	T96
214	<	T95	T96	T139
215	if false	T139	-	L5
216	=	0	-	T97
217	=	T97	-	item3
218	Label	-	-	L4
219	=	item3	-	T97
220	=	0	-	T97
221	>=	T97	T97	T131
222	If False	T131	-	L5
223	=	item3	-	T97
224	=	6	-	T98
225	<	T97	T98	T132
226	if false	T132	-	L5
227	=	9	-	T99
228	=	T99	-	e
229	=	0	-	T102
230	=	T102	-	item4
231	Label	-	-	L4
232	=	item4	-	T102
233	=	0	-	T102
234	>=	T102	T102	T123
235	If False	T123	-	L5
236	=	item4	-	T102
237	=	8	-	T103
238	<	T102	T103	T124
239	if false	T124	-	L5
240	=	40	-	T104
241	=	T104	-	f
242	=	0	-	T107
243	=	T107	-	item5
244	Label	-	-	L4
245	=	item5	-	T107
246	=	0	-	T107
247	>=	T107	T107	T115
248	If False	T115	-	L5
249	=	item5	-	T107
250	=	5	-	T108
251	<	T107	T108	T116
252	if false	T116	-	L5
253	goto	-	-	L4
254	Label	-	-	L5
255	goto	-	-	L4
256	Label	-	-	L5
257	goto	-	-	L4
258	Label	-	-	L5
259	goto	-	-	L4
260	Label	-	-	L5
261	goto	-	-	L4
262	Label	-	-	L5
263				

We can see that the * operation has been changed to <<

Quads (after constant propagation)-

-----Quadruples after Constant Propagation-----				
Lno.	Oper.	Arg1	Arg2	Res
265				
266				
267	import	mymodule	-	-
268	BeginF	foo	-	-
269	BeginF	bar	-	-
270	=	False	-	T4
271	=	T4	-	q
272	=	"stringify"	-	T7
273	=	T7	-	r
274	BeginF	baz	-	-
275	EndF	baz	-	-
276	EndF	bar	-	-
277	EndF	foo	-	-
278	Call	foo	-	T38
279	=	True	-	T31
280	=	T31	-	x
281	=	"this is a string"	-	T34
282	=	T34	-	z
283	=	3	-	T37
284	=	4	-	T38
285	<<	3	2	T39
286	=	T39	-	c
287	=	18	-	T42
288	=	18	-	s
289	=	18	-	T42
290	=	18	-	T42
291	+	18	18	T47
292	=	T47	-	t
293	=	100	-	T50
294	=	15	-	T51
295	+	100	15	T52
296	=	T52	-	u
297	=	5	-	T55
298	=	5	-	a
299	=	10	-	T58
300	=	10	-	b
301	=	5	-	T55
302	=	5	-	T62
303	==	5	5	T63
304	=	10	-	T58
305	=	8	-	T65
306	>	10	8	T66
307	and	T63	T66	T67
308	If False	T67	-	L0
309	=	9	-	T68
310	=	9	-	a
311	=	20	-	T71
312	=	20	-	b
313	goto	-	-	L1
314	Label	-	-	L0
315	=	5	-	T55
316	=	4	-	T81
317	<	5	4	T82
318	If False	T82	-	L0
319	goto	-	-	L1
320	Label	-	-	L0
321	Label	-	-	L1
322	Label	-	-	L1
323	Label	-	-	L1
324	=	0	-	T93
325	=	T93	-	item1
326	Label	-	-	L4
327	=	item1	-	T93
328	=	0	-	T93
329	>=	T93	T93	T145
330	If False	T145	-	L5
331	=	item1	-	T93
332	=	2	-	T94
333	=	T93	2	T146
334	<	T146	-	L5
335	if false	0	-	T95
336	=	T95	-	item2
337	Label	-	-	L4
338	=	item2	-	T95
339	=	0	-	T95
340	>=	T95	T95	T138
341	If False	T138	-	L5
342	=	item2	-	T95
343	=	4	-	T96
344	<	T95	4	T139
345	if false	T139	-	L5
346	=	0	-	T97
347	=	T97	-	item3
348	Label	-	-	L4
349	=	item3	-	T97
350	=	0	-	T97
351	>=	T97	T97	T131
352	If False	T131	-	L5
353	=	item3	-	T97
354	=	6	-	T98
355	<	T97	6	T132
356	if false	T132	-	L5
357	=	9	-	T99
358	=	0	-	e
359	=	0	-	T102
360	=	T102	-	item4
361	Label	-	-	L4
362	=	item4	-	T102
363	=	0	-	T102
364	>=	T102	T102	T123
365	If False	T123	-	L5
366	=	item4	-	T102
367	=	8	-	T103
368	<	T102	8	T124
369	if false	T124	-	L5
370	=	40	-	T104
371	=	40	-	f
372	=	0	-	T107
373	=	T107	-	item5
374	Label	-	-	L4
375	=	item5	-	T107
376	=	0	-	T107
377	>=	T107	T107	T115
378	If False	T115	-	L5
379	=	item5	-	T107
380	=	5	-	T108
381	<	T107	5	T116
382	if false	T116	-	L5
383	goto	-	-	L4
384	Label	-	-	L5
385	goto	-	-	L4
386	Label	-	-	L5
387	goto	-	-	L4
388	Label	-	-	L5
389	goto	-	-	L4
390	Label	-	-	L5
391	goto	-	-	L4
392	Label	-	-	L5
393	goto	-	-	L4
394	Label	-	-	L5

Quads (after constant folding)-

395	-----Quadruples after Constant Folding-----			
396				
397				
398 Lno.	Oper.	Arg1	Arg2	Res
399				
400 0	import	mymodule	-	-
401 1	BeginF	foo	-	-
402 2	BeginF	bar	-	-
403 3	=	False	-	T4
404 4	=	T4	-	q
405 5	=	"stringify"	-	T7
406 6	=	T7	-	r
407 7	BeginF	baz	-	-
408 8	EndF	baz	-	-
409 9	EndF	bar	-	-
410 10	EndF	foo	-	-
411 11	Call	foo	-	T30
412 12	=	True	-	T31
413 13	=	T31	-	x
414 14	=	"this is a string"	-	T34
415 15	=	T34	-	z
416 16	=	3	-	T37
417 17	=	4	-	T38
418 18	=	12	-	T39
419 19	=	T39	-	c
420 20	=	18	-	T42
421 21	=	18	-	s
422 22	=	18	-	T42
423 23	=	18	-	T42
424 24	=	36	-	T47
425 25	=	T47	-	t
426 26	=	100	-	T50
427 27	=	15	-	T51
428 28	=	115	-	T52
429 29	=	T52	-	u
430 30	=	5	-	T55
431 31	=	5	-	a
432 32	=	10	-	T58
433 33	=	10	-	b
434 34	=	5	-	T55
435 35	=	5	-	T62
436 36	==	5	5	T63
437 37	=	10	-	T58
438 38	=	8	-	T65
439 39	>	10	8	T66
440 40	and	T63	T66	T67
441 41	If False	T67	-	L0
442 42	=	9	-	T68
443 43	=	9	-	a
444 44	=	20	-	T71
445 45	=	20	-	b
446 46	goto	-	-	L1
447 47	Label	-	-	L0
448 48	=	5	-	T55
449 49	=	4	-	T81
450 50	<	5	4	T82
451 51	If False	T82	-	L0
452 52	goto	-	-	L1
453 53	Label	-	-	L0
454 54	Label	-	-	L1
455 55	Label	-	-	L1
456 56	=	0	-	T93
457 57	=	T93	-	item1
458 58	Label	-	-	L4
459 59	=	item1	-	T93
460 60	=	0	-	T93
461 61	>=	T93	T93	T145
462 62	If False	T145	-	L5
463 63	=	item1	-	T93
464 64	=	2	-	T94
465 65	<	T93	2	T146
466 66	if false	T146	-	L5
467 67	=	0	-	T95
468 68	=	T95	-	item2
469 69	Label	-	-	L4
470 70	=	item2	-	T95
471 71	=	0	-	T95
472 72	>=	T95	T95	T138
473 73	If False	T138	-	L5
474 74	=	item2	-	T95
475 75	=	4	-	T96
476 76	<	T95	4	T139
477 77	if false	T139	-	L5
478 78	=	0	-	T97
479 79	=	T97	-	item3
480 80	Label	-	-	L4
481 81	=	item3	-	T97
482 82	=	0	-	T97
483 83	>=	T97	T97	T131
484 84	If False	T131	-	L5
485 85	=	item3	-	T97
486 86	=	6	-	T98
487 87	<	T97	6	T132
488 88	if false	T132	-	L5
489 89	=	9	-	T99
490 90	=	9	-	e
491 91	=	0	-	T102
492 92	=	T102	-	item4
493 93	Label	-	-	L4
494 94	=	item4	-	T102
495 95	=	0	-	T102
496 96	>=	T102	T102	T123
497 97	If False	T123	-	L5
498 98	=	item4	-	T102
499 99	=	8	-	T103
500 100	<	T102	8	T124
501 101	if false	T124	-	L5
502 102	=	40	-	T104
503 103	=	40	-	f
504 104	=	0	-	T107
505 105	=	T107	-	item5
506 106	Label	-	-	L4
507 107	=	item5	-	T107
508 108	=	0	-	T107
509 109	>=	T107	T107	T115
510 110	If False	T115	-	L5
511 111	=	item5	-	T107
512 112	=	5	-	T108
513 113	<	T107	5	T116
514 114	if false	T116	-	L5
515 115	goto	-	-	L4
516 116	Label	-	-	L5
517 117	goto	-	-	L4
518 118	Label	-	-	L5
519 119	goto	-	-	L4
520 120	Label	-	-	L5
521 121	goto	-	-	L4
522 122	Label	-	-	L5
523 123	goto	-	-	L4
524 124	Label	-	-	L5
525				
526				

Quads (after dead code elimination)-

-----Quadruples after Dead Code Elimination-----				
Lno.	Oper.	Arg1	Arg2	Res
529	import	mymodule	-	-
530	BeginF	foo	-	-
531	BeginF	bar	-	-
532	BeginF	baz	-	-
533	EndF	baz	-	-
534	EndF	bar	-	-
535	EndF	foo	-	-
536	Call	foo	-	T30
537	=	5	5	T63
538	>	10	8	T66
539	and	T63	T66	T67
540	If False	T67	-	L0
541	goto	-	-	L1
542	Label	-	-	L0
543	<	5	4	T82
544	If False	T82	-	L0
545	goto	-	-	L1
546	Label	-	-	L0
547	Label	-	-	L1
548	Label	-	-	L1
549	=	0	-	T93
550	=	T93	-	item1
551	Label	-	-	L4
552	=	item1	-	T93
553	=	0	-	T93
554	>=	T93	T93	T145
555	If False	T145	-	L5
556	=	item1	-	T93
557	=	0	-	T95
558	=	T95	-	item2
559	Label	-	-	L4
560	=	item2	-	T95
561	=	0	-	T95
562	>=	T95	T95	T138
563	If False	T138	-	L5
564	=	item2	-	T95
565	=	0	-	T97
566	=	T97	-	item3
567	Label	-	-	L4
568	=	item3	-	T97
569	=	0	-	T97
570	>=	T97	T97	T131
571	If False	T131	-	L5
572	=	item3	-	T97
573	=	0	-	T102
574	=	T102	-	item4
575	Label	-	-	L4
576	=	item4	-	T102
577	=	0	-	T102
578	>=	T102	T102	T123
579	If False	T123	-	L5
580	=	item4	-	T102
581	=	0	-	T107
582	Label	-	-	item5
583	=	item5	-	L4
584	=	0	-	T107
585	=	T107	-	T107
586	>=	T107	T107	T115
587	If False	T115	-	L5
588	=	item5	-	T107
589	goto	-	-	L4
590	Label	-	-	L5
591	goto	-	-	L4
592	Label	-	-	L5
593	goto	-	-	L4
594	Label	-	-	L5
595	goto	-	-	L4
596	Label	-	-	L5
597	goto	-	-	L4
598	Label	-	-	L5
599	goto	-	-	L4
600	Label	-	-	L5
601				

Test file 2-

- Input-

```
1 #comment
2
3 a=15
4 for i in a:
5     print("nope")
```

- Output-

```
sneha@sneha-VirtualBox:~/Desktop/cd_proj/phase2$ ./a.out < input2.py
ERROR: Identifier 'a' at line 4 Not Indexable
sneha@sneha-VirtualBox:~/Desktop/cd_proj/phase2$
```

Test file 3-

- Input-

```
1 import random
2
3 if a==10:
4     print("equal")
```

- Output-

```
sneha@sneha-VirtualBox:~/Desktop/cd_proj/phase2$ ./a.out < input3.py
ERROR: Identifier 'a' at line 3 Not Declared
sneha@sneha-VirtualBox:~/Desktop/cd_proj/phase2$
```

CONCLUSION

We were able to successfully design and implement a mini-compiler for the “if-else” and “for” constructs of Python. In the process, we gained a better insight on how compilers work, and their various phases.