Q1

(i) What are the lengths of lexemes that match the patterns in your lexer design?

A.

Various lengths of lexemes match their pattern in our lexer design.
Length of the lexemes which match their pattern are:

Lexeme	Length of the lexeme
if	2
for	3
else	4
while	5
rotate_cw	9
rotate_acw	10

Some other lexemes which are defined as reserved words also match their pattern.

(ii) How many distinct patterns are there in your lexer design?

A.

There are 26 distinct patterns in our lexer design

(iii) How many distinct token types are there in your lexer design?

A.

There are 46 distinct token types in our lexer design

(iv) If the above two numbers (patterns and token types) are not the same, explain the difference.

A.

The above two numbers are not same.
The reserved words(reserved_keywords, reserved_tetrimino_movement, reserved_tetrimino) which we have defined using set are counted in tokens and not counted in pattern. This is because the lexemes corresponding to the reserved words are matched with the regex defined for an identifier and later assigned their specific token using conditional statement.

(v) How many of these token types are encoded into an enumerated type or a number?
A.
We have not encoded any token type into an enumerated type or number.

(vi) How many of these token types are just the lexemes themselves?
A.
There are is no such token type

(vii)Give a single, condensed regexp for all the (ASCII) characters that will be just ignored by your lexer.
A.
Condensed regexp for all the characters that will be just ignored by our lexer: (␣ represents white space)
((␣\t\r)|(//.*)|(\n+))

(viii) Which of these ignored characters are delimiters? Indicate clearly and separately. Whitespace on your white paper cannot be counted. So, e.g., use the symbol "␣" for one space and "␣␣␣", for three spaces. Also \t,\r,\n stand for tab, carriage return, newline, resp.
A.
Ignored characters which are delimiters:
(␣\t\n\r)

Q2

No, there is no lexeme which is transformed or truncated after token recognition.

Stepwise argument about lexer's pattern-action specifications:

Step 1: Defining the regular expression rules

The regular expression rules can be defined in two ways in PLY:

(i) Defining regular expression rule as a function
Eg:
```
def t_NUMBER(t):
      r'\d+'
      t.value = int(t.value)
      return t
```

(ii) Defining regular expression rule as pair of token class and pattern(represents the regex)

Eg:

```
PLUS    = r'\+'
MINUS   = r'-'
TIMES   = r'\*'
DIVIDE  = r'/'
```

Step 2: Deciding the precedence of the regular expression rules:

When building the master regular expression, rules are added in the following order:

1. All tokens defined by functions are added in the same order as they appear in the lexer file.
2. Tokens defined by strings are added next by sorting them in order of decreasing regular expression length (longer expressions are added first).

Step 3: Generating an instance of LexToken for the identified token:
The output of each call to token() function is an instance of LexToken. Each instance of LexToken contains fields that are initialized to attributes of identified lexeme. LexToken object has attributes of t.type which is the token type (as a string), t.value which is the lexeme (the actual text matched), t.lineno which is the current line number, and t.lexpos which is the position of the token relative to the beginning of the input text.

Q3

Yes, our lexer provides an external interface which is to be called by the parser for each next token. We designed our scanner by specifying the lexeme patterns to a lexical analyser generator, PLY. PLY is a pure-Python implementation of the popular compiler construction tools lex and yacc. lex.py provides an external interface in the form of a token() function that returns the next valid token on the input stream. Yacc.py, of the Python Package PLY, calls this repeatedly to retrieve tokens and invoke grammar rules.

The output of each call to token() function is an instance of LexToken. Each instance of LexToken contains fields that are initialized to attributes of identified lexeme. LexToken object has attributes of t.type which is the token type (as a string), t.value which is the lexeme (the actual text matched), t.lineno which is the current line number, and t.lexpos which is the position of the token relative to the beginning of the input text. The token() function return None if the end of the input file has been reached