1. Syntax analysis is a second part of compilation process which right after the lexical analysis. The purpose is to check whether the given input correct or not according to the rules of the particular programming language.
2. There is a special module implemented in ply which is called ply.yacc. The tokens map are taken from the lexer. Then we use functions with a reserved name p_CASENAME. Inside the function there is a docstring which contains some appropriate specification for this particular case (basically this syntactic rule) 3. a) Variable definition. There are 4 different cases how variables could be defined according to their type. It could be simple variable (number or literal), constant (not variable, actually), tuple variable or pipe variable. Each of them has their own implementation.
   b) Function call. Function call looks like: funcIDENT(r'[A-Z]{1}([a-z]|[0-9]|_)+') and then in squared parentheses ([]) arguments.
   c) Tuple expression. It is a tuple_atom. Which could be implemented in 4 different ways (tupleIDENT, [constant_expression ** constant_expression], [constant_expression..constant_expression]], function_call) + (possibly) ++(double plus).
   tupleIDENT is r'<[a-z]+>' constant_expression
   is literal or number
4. a) No, it's not possible. Function body (inside BEGIN - END) may contain only variable definitions and return. Hence, it's impossible to define a new function inside the other one.
   b) Yes, it is possible. Because, to perform such operation we use simple_expression which can be defined as STRING_LITERAL + STRING_LITERAL.
   c) Yes. It is possible. Initializing a variable we need to use simple_expression which uses atom inside it and which is allowed to be constIDENT.
   d) No, it's not possible. Because when we initialize a constant, we need to have consIDENT or a number. None of them matches variable name.
   e) No, it's not possible. In current syntax we have only double plus (++), not double minus (--).
   f) It is done with help of recursion order. PLUS and MINUS are connected with PLUS_MINUS which is allowed to be each of them. On the same time DIV and MULT are also connected into DIV_MULT which is allowed to be each of them also. Then, all such operations are arranged into several steps. simple_expression -> term -> factor -> atom.On the simple_expression step only PLUS and MINUS are allowed. Then on the term level MULT and DIV appear. And on the factor level MINUS is allowed (as a prefix, e.g. -5). In this way it is guaranteed that PLUS and MINUS operations will be done in the end (as they are on the higher levels of recursion as MULT and DIV.
5. Functions are implemented.
6. I had problems with yacc.parser as it is working in different ways on the same examples (particulary on the last template testcase) and some times it fails but sometimes it's not. It was somehow related to my regex condition (as it is not ok with some expression OR empty_string). But finally I fixed it.