1. Syntax analysis is the step after lexical analysis where the tokens provided by the lexer are turned into a parse tree. Basically the syntax analysis checks that the structure of the code is correct according to its rules how different things are defined. Meaning the code "makes sense". For example lexer is fine with two "while" tokens next to each other but syntax analyser is not (probably) because there is no rule match for something like that.

2. In the PLY tool there is different symbols (like program, atom, formals) that have the rules inside them. E.g. in my code "definitions" symbol can be a "function_definition", "procedure_definition" or "variable_definition". And these three are also defined as a symbol and construct from something. Eventually the "lowest" parts are the tokens from lexer, these are "terminal" symbols. So for example lexer gives a terminal symbol "INT_LITERAL" that can be a non-terminal symbol "atom" that can be "factor" that can be used in "term" and so on. And these things are implementing the EBNF rules defined in our PostHaste syntax.

3.a do-unless statement executes the statement(s) if the expression in unless is not true. And if the expression is true, then it executes the other statement(s) IF they exist. The latter statement(s) are optional. So the whole statement might not do anything.

3.b Procedure call is calling a function with possible argument(s). It has the function name, parenthesis and possible arguments. I understood that procedure call might return something but function call always returns something, according to EBNF rules.

4.

- Unless_expression is basically if-else. It does something if something is true(false technically this time), else it does something else.

- loop_statement on the other hand has only one set of statements and it does them until some condition is met.

- unless_statement has the "else" part as optional in comparison to unless_expression.

I think unless_expression is something like the ternary operator in many languages. (x < 5 ? something : something else)

5.

a. No because function or procedure definitions don't have possibility of having any definitions inside them. But you can *call* another procedure inside a procedure.

b. Maybe by making a new "complete_statement" that is like this: "statement SEMICOLON" and then a statement_list would be many of those "complete_statement"s.

c. I think no but I didn't fully understand the question unfortunately.

d. 3---2 seems to be allowed but xx---yy not. I tried to investigate this and the first point is that in the lexer the ---2 will produce token(-), token(-) and then int (-2) BUT with yy there will be three token(-) and then IDENT(y). And the syntax will then allow two minuses because there can be a minus in front of atom to make a factor, and ALSO a minus in front of a factor to make a term. And these are backed up by the fact that "xx---2" is also allowed and

“3----2” (four minuses) is not allowed. So the main root cause is that the number 2 is “consuming” the one minus character in lexer but “yy” is not.

e. No because function definition has rvalue inside it which is basically an expression and procedure call can’t be an expression. But as said earlier, inside procedure definition this can happen.

f. unless expression is rvalue and rvalue can be inside a function definition or assignment.

g. By checking what is in the left and right side of the equals sing. With initialization there is IDENT = expression, with assignment there is lvalue = rvalue and with comparing there is expression = simple_expr. Meaning no ambiguity possible.