1. Abstract syntax tree is a data structure that represents the syntactic structure of the source code written in some particular programming language. Each node represents the concrete construct of the programming language (precisely that piece of source code). AST is a result of the second compilation phase (syntax analysis). Further AST is used for semantic analysis and code generation.
2. In the PLY tool syntax tree is generated with help of the same instruments that used in the syntax analysis phase - yacc.py. The ASTnode structure should be defined in the source code. Basically tree building process is done using the same objects as in phase 2: you can create a node at any rule that was described for parser. Then, each of them could be connected with any other rules (no matter is there an AST node created or not) with a reserved names p[i].children\_YOURNAME, p[i].child\_YOURNAME (basically it is true for our way of implementation as we used reserved names for tree\_print script). The name of the node doesn’t matter a lot, it is just how you want your tree to look like, the main thing is connections (edges).
3. a. For variables there is a list of variables definitions (their parent is program or function in most cases), for each variable its name is defined in the node. Besides that it contains the expression as a child, which contains the assigned value as a value.
   1. Pipe expressions: Pipe definition -> list [tuple expressions], list [pipe operations] for each tuple expression -> tuple atom(value).
   2. Function call: function call (value: funcIDENT) -> arguments-> list[simple\_expression(value)]
4. a. In some places where it is possible to have zero or more elements I have an expression which created a list of children nodes in any case, but if there is no elements it will be None in the tree.
   1. Basically in my implementation in mostly all p\_regexp\_... rules (which means all places where it is possible to have more then one object in a list (e.g variable definitions in program) some nodes are replaced using the list od childrens. Also I have several function called like function\_or\_variable (smth1\_or\_smth2) in these cases I don’t create a node just go further, so it also kind of tree optimization.
5. Functions are implemented.
6. This part seems to me easier then the previous ones (perhaps only the second phase), because it is mostly just editing previous code and rearranging some parts, while there are a lot of things to think about and to spend time on creating an appropriate structure. I think, this is a really good illustration of how real-world problems differs from the ‘school tasks’ as there are no really strict rules, but you need to think really deep about the possible cases.