**Submission Guidelines**

Dr. Shasha started off class by acknowledging the python programs everyone had submitted online, adding a few more guidelines to make the submission process unified and efficient:

- Submit programs by midnight on wednesday to ensure Dr. Shasha sees them

- Make sure the program compiles and runs without any errors before sending it

- If you received assistance from a colleague, modify your code to have a comment "# " followed by the colleague's name

- Everybody is working on a solution to the exact same problem, so name your file something unique to you. E.g. JaneDoeFibonacciSequence.py

- Make sure the program is saved as plain text. This means the program is detailed in unicode, a universal code standard that should be understood and interpreted by all machines. (for TextEdit users, here is a tutorial for converting an .rtf to plain text: http:// support.apple.com/kb/TA20406)

**Chomsky's Lingustics Continued: Finite State Automata**

The class then segued into further discussion of Chomsky's *Three Models for the Description of Language,* beginning with an expansion of Finite State Automata as blueprints for repeated processes, especially with respect to the composition of syntactically-sound sentences or phrases. Finite State Automata may appear on an upcoming quiz and should be understood without confusion.

Dr. Shasha challenged the class (with the usual incentive of chocolates) to construct a Finite State Automaton that creates sentences with the structure n(ab).

E.g. ab, abab, ababab, abababab, and c. The result can be seen in the attached image 'form n(ab).png'. Proceeding this was another incentivized challenge: constructing a FSA that creates sentences structured anbm. E.g. aaaaaab, aabbb, aaabbb, aaaabbbb, and c. The result can be seen in the attached image 'form anbm.png'. Momentarily edging away from producing solely grammars, Dr. Shasha extended the FSA form to the real-life mechanical process of an elevator. The elevator reacts to different stimuli such as the up button being pressed, the down button being pressed, and the expectation of making a decision as to what to do when both are pressed. This was lightly juxtaposed to circuitry, an upcoming module.

**Chomsky's Linguistics Continued: Grammars and Parse Trees**

Next, we discussed the rigidity of grammars mostly in relation to how computers interpret data when they are preparing to commit mathematical operations. Grammar structure is responsible for why these mathematical processes differ from one programming language to another. Dr. Shasha explained that it is advantageous for some languages to read expressions from right to left, while others follow generally-acknowledged order of operations. In summary, grammars are parameters that define how an operation is done. Using a couple of axiomatic declarations to define the grammar, the class would compose a parse tree to interpret several different operations. The declarations would be broken into expression statements (E) and term statements (T), respectively. The first grammar we examined is as follows:

Grammar 1:

R1: E :: T+T | T-T | T\*T | T/T

R2: T :: any number | (E)

This grammar can be applied to countless equations, but does not necessarily accommodate all mathematical situations without equivocation. The first example we examined was the equation (5 \* 3) + 4. Since there are three active terms, we cannot simply use Rule 1's expression declaration; we must use R2 to represent an equation as a term, i.e. (5 \* 3) and then proceed to use R1. Terms get compartmentalized into other terms to shorten the length of the expression. See the detailed parse tree ('ParseTreeI.png') for further elucidation. When representing a process such as (T \* T) branching upwards to a single expression (E), it is helpful to note the state of each bit. For instance (T \* T) is derived from the (E) expression above it, making (E) the node, and T, \*, and T the children. Every application of the rule from the top of the tree down follows this same format.

Other grammars were represented by other parse trees, but the main idea is the same. Parse trees are expansions of axioms as interpretations of arithmetic, linguistic, or other systems. Dr. Shasha said constructing a parse tree for a particular instance of a grammar could be a quiz question, meaning parse trees should be understood without confusion.