1. What is the difference between Lexical and Syntax Analysis?

Lexical is a low-level, finite automaton, while Syntax is a high-level, push-down automaton.

2. What are the reasons to separate Lexical and Syntax Analysis?

Reasons include for simplicity, simplifying the parser, Portability, ensuring the parser is always portable while the lexical analyzer may not be, and Efficiency, allowing for optimization of the lexical analyzer.

3. What are three approaches to building a lexical analyzer?

Using software to construct a table-driven analyzer from a formal description of stated tokens.

Creating a program which implements a state diagram detailing the tokens.

Hand-constructing a table-driven implementation of the previous state diagram.

4. Explain the difference between the two types of Finite Automaton (link

Deterministic - Requires an input character to change one state.

Nondeterministic - Null (nonrequirement) of a character to change states, and can change multiple states.

5. Differentiate between a Finite Automaton and a Push-Down Automaton.

Finite does not store alphabetic input, only storing its current state, while pushdown has a stack for storing states, including longer sequenced alphabetic characters.

6. What is a Recursive-Descent parser?

A top-down coded implementation of a parsing algorithm.

7. We discussed earlier that grammar should not be left-recursive for top-down parsers. See here . Based on the video and the notes, answer how you would remove left recursion in this example.
    - E → E + E
    - E → y
    - —------------
    - E → EE'
    - E' → +EE' | ε
    - E → y

8. Remove left recursion in the following grammar.
    - E –> E + T | T
    - T –> T * F | F
    - F –> (E) | id
    - —-----------------
    - E → TE'
    - E' → +TE' | ε
    - T → FT'
    - T → *FT' | ε
    - F → (E) | id
9. Do left factoring for the following grammar:
    - A → aAB / aBc / aAc
    - —---------------
    - A → aA'
    - A' → AD / Bc
    - D → B / c
10. What issue does left-factoring resolve?

Left-factoring removes the infinitely recursive or forbidden normal forms found in top-down parsing by converting them to a non-left recursive form.