- Find students around you to form a ***small group***.
- Work on the exercise problems in sequence. You are allowed to use ***all resources*** (including textbook and search engines) to help you solve the problems.
- After solving each problem, ***discuss*** your solution with other group members and exchange your ideas. You should try to poke holes in each other's argument and defend your own; this is a good way to learn about how to communicate.
- In daily life we are used to *not argue* with people and overlook the gaps in the statements. Well, not for this course. Being critical and patient when listening to others, ask questions to see if you understand them correctly, and provide well-thoughtout counter-arguments when you disagree. Be civilized and argue about ideas but don't make it personal.
- As you are trying to solve the problems you might have questions. Ask the questions within the group and see if others know the answer. If no ones knows, or the whole group is stuck and don't know how to proceed, Raise your hand and pull one of the ***course staff*** to help.
- After reaching concensus, ***write down*** the solutions *in your own words*. You are not allowed to copy sentences from others or TAs; everything you deliver must be from *you*. Use the writing tips provided on Canvas to help with your presentation.
- ***Submit*** your writeup with the proofread signature through Gradescope. You have *24 hours* to do so. The TA will grade them and provide feedback.

Our topic for this working session is the *regular expression*.

Recall that a lanaguage L is ***regular*** if L can be recursively defined as:

- empty language $\varnothing$,
- singleton $\{\mathtt{x}\}$ for some character $\mathtt{x}$,
- union $A \cup B$ if both A and B are regular languages,
- concatenation $A \cdot B$ if both A and B are regular languages,
- Kleene-star A^* if A is a regular language.

We can express any regular language using ***regular expressions***: $\varnothing$, $\mathtt{x}$, $A + B$, AB, and A^*. By adding proper parentheses, the language corresponding to a regular expression must be unique.

However, the same regular language L might have multiple valid regular expressions! Recall the example in class: $(\mathtt{0} + \mathtt{1})^*$ and $(\mathtt{0}^*\mathtt{1}^*)^*$ both correspond to the set of all binary strings.

Tips: We can use shorthands like A^k to represent k-fold concatenation, A^+ to represent Kleene-star with at least one repetition (that is, $A^+ := A^*A$), and Σ to represent the union of all singletons, one for each character in the alphabet Σ.

Example. Design a regular expression for each of the following language over alphabet $\{\mathtt{0}, \mathtt{1}\}$:

- The set of all strings having $\mathtt{0}$ at the 39th position.

Solution: A regular expression $\Sigma^{38}\mathtt{0}\Sigma^* = \underbrace{(\mathtt{0}+\mathtt{1})\cdots(\mathtt{0}+\mathtt{1})}_{38 \text{ times}}\mathtt{0}(\mathtt{0}+\mathtt{1})^*$, where $\Sigma := \{\mathtt{0}, \mathtt{1}\}$.

We allow any character in the first 38 positions, but the 39th has to be $\mathtt{0}$. (In particular, the string has to have length at least 39.) After that, we can have whatever, including the empty string ε.

Design a regular expression for each of the following languages. No formal proofs required, but explanation of your construction in a few English sentences would be helpful. Throughout the exercise we assume the alphabet to be $\Sigma := \{0, 1\}$.

1. Set of all strings containing `000` or `111` as substrings
2. Set of all strings containing `000` or `111` as *subsequences*
3. Set of all strings *not* containing `000` *nor* `111` as substrings
4. Set of all strings *not* containing `000` *nor* `111` as subsequences

---

To think about later: (No submissions needed)

5. Set of all strings containing `000` *and* `111` as substrings
6. Set of all strings that contains at least three `0`s and two `1`s
7. Set of all strings except for `000` and `111`
8. Set of all strings with more `01`s than `10`s

Conceptual question: Is there a general rule to express the *complement* of a language? That is, how do we write a regular expression for the set of strings *not* in a given regular language L?