Recall from lecture that a *regular expression* is compact notation for a language (that is, a set of strings). Formally, a regular language is one of the following:

- The symbol $\emptyset$ (representing the empty set)
- Any string (representing the set containing only that string)
- $R + S$ for some regular expressions R and S (representing alternation / union)
- $R \cdot S$ or RS for some regular expressions R and S (representing concatenation)
- R^* for some regular expression R (representing Kleene closure / unbounded repetition)

In the absence of parentheses, Kleene closure has highest precedence, followed by concatenation. For example:

$$0 + \mathtt{RE}^* = 0 + \mathtt{R}(\mathtt{E}^*) = \{\mathtt{0},\ \mathtt{R},\ \mathtt{RE},\ \mathtt{REE},\ \mathtt{REEE},\ \mathtt{REEEE},\ \mathtt{REEEEEE},\ \mathtt{REEEEEEE},\ \mathtt{REEEEEEEE}, \ldots\}$$

$$0 + (\mathtt{RE})^* = \{\varepsilon,\ \mathtt{0},\ \mathtt{RE},\ \mathtt{RERE},\ \mathtt{RERERE},\ \mathtt{RERERERE},\ \mathtt{RERERERERE},\ \mathtt{RERERERERERE}, \ldots\}$$

$$(0 + \mathtt{R})\mathtt{E}^* = \{\mathtt{0},\ \mathtt{R},\ \mathtt{0E},\ \mathtt{RE},\ \mathtt{0EE},\ \mathtt{REE},\ \mathtt{0EEE},\ \mathtt{REEE},\ \mathtt{0EEEE},\ \mathtt{REEEE},\ \mathtt{0EEEEE}, \ldots\}$$

$$(0 + \mathtt{RE})^* = \{\varepsilon,\ \mathtt{0},\ \mathtt{00},\ \mathtt{RE},\ \mathtt{000},\ \mathtt{0RE},\ \mathtt{RE0},\ \mathtt{0000},\ \mathtt{00RE},\ \mathtt{0RE0},\ \mathtt{RE00},\ \mathtt{RERE}, \ldots\}$$

Give regular expressions for each of the following languages over the binary alphabet $\{\mathtt{0}, \mathtt{1}\}$. (For extra practice, find multiple regular expressions for each language.)

0. All strings.
1. All strings containing the substring `000`.
2. All strings *not* containing the substring `000`.
3. All strings in which every run of `0`s has length at least 3.
4. All strings in which the *last* `1` appears before the *first* substring `000`.
5. All strings containing at least three `0`s.
6. Every string except `000`. *[Hint: Don't try to be clever.]*

More difficult problems to work on later:

7. All strings w such that *in every prefix of* w, the number of `0`s and `1`s differ by at most 1.
8. All strings containing at least two `0`s and at least one `1`.
9. All strings in which every run has odd length. (For example, `0001` and `100000111` and the empty string ε are in this language, but `000000` and `001000` are not.)

★10. All strings w such that *in every prefix of* w, the number of `0`s and `1`s differ by at most 2.

★11. All strings in which the substring `000` appears an even number of times. (For example, `01100` and `000000` and the empty string ε are in this language, but `00000` and `001000` are not.)