1:
The set consists of strings of blocks of 0’s and 1’s where each block is separated by a 2; each block can have any combination of 0’s and 1’s; the length of each successive block is one greater than the previous block; the first block is of length 1; the strings end in a 2; there are an even number of blocks; there are a minimum of 2 blocks.

2:
We assume that the alphabet of L and M are disjoint. If they are not disjoint, we can simply do a homomorphism to the alphabet of one of the languages and add hats for example. Then we just do the corresponding inverse homomorphism at the end to remove these hats.

We define the homomorphism $h_1$ as follows:
\[
\begin{align*}
    h_1(a) &= a \forall a \in \Sigma_L \\
    h_1(b) &= \in \forall b \in \Sigma_M
\end{align*}
\]

Then $h_1^{-1}(L)$ is the language containing strings in L with symbols from the alphabet of M arbitrarily inserted. Similarly define the homomorphism $h_2$ as:
\[
\begin{align*}
    h_2(b) &= b \forall b \in \Sigma_M \\
    h_2(a) &= \in \forall a \in \Sigma_L
\end{align*}
\]

Therefore the desired language $\text{alt}(L, M)$ is obtained as:
\[
\text{alt}(L, M) = h_1^{-1}(L) \cap h_2^{-1}(M) \cap (\Sigma_L \Sigma_M)^* 
\]

Since $\text{alt}(L, M)$ can be obtained from L and M by operations that preserve regularity, therefore if L and M are regular, so is $\text{alt}(L, M)$.

3:
We can construct an automaton for $\text{cycle}(L)$ given a DFA for L as follows. Let us assume that $xy$ is a string that may be in $\text{cycle}(L)$ such that $yx$ is in L. The automata guesses a start state from which to run the DFA for L – the DFA for L is run starting from the state the DFA would have been in after seeing the string y. Then a guess is made that we have seen the end of the string x and the DFA must be in a final state at this point. We now start from the start state of the DFA and run it on the string y. At the end of the string, a check is made to see if the
resulting state is the same state that was started from. If so, the string xy is in cycle(L).

Let Q be the set of states in the DFA for L. Let the states of the NFA be 3-tuples of the form (qstate, qstart, 0/1). qstate represents the state that the DFA for L is in as we run the input string on it. qstart is the guess of the start state from which the DFA for L is run. The 3rd field starts with 0 and only changes once to 1 when we transition from a final state of the DFA to the start state of the DFA (this occurs when we have seen the end of x). So the NFA for cycle(L) has epsilon-transitions from its start state to all states of the form (q1, qs, 0) where q1 ∈ Q and qs ∈ Q. It then contains all transitions of the form δ((q1, qs, 0), a) = (q2, qs, 0) and δ((q1, qs, 1), a) = (q2, qs, 1) for all transitions in DFA of the form δ(q1, a) = q2. Then there exists a transition δ((qf, qs, 0), a) = (q0, qs, 1) where qf is a final state in DFA(L) and q0 is the start state of DFA(L); this transition corresponds to the jump from the final to the start state of the DFA. Finally all states of the form (q, q, 1) are accepting states.

Since we can construct an NFA that accepts cycle(L) given a DFA for L, therefore regular languages are closed under the operation cycle(L).

4:

a) Note that \( \{0^i1^j \mid i \neq j \}^c \cap 0^*1^* = L_{0_{n^2}} \). We know that regular sets are closed under complementation and intersection. Therefore if \( \{0^i1^j \mid i \neq j \} \) is regular, \( L_{0_{n^2}} \) must be regular. But we know that \( L_{0_{n^2}} \) is not a regular set. Therefore, \( \{0^i1^j \mid i \neq j \} \) is not regular.

b) We transform this language to \( L_{0_{n^2}} \) by applying the homomorphism h defined as follows:

\[
\begin{align*}
  h(0) &= 0 \\
  h(1) &= 1 \\
  h(2) &= 1
\end{align*}
\]

5:

Consider a regular language L with a DFA having n states. It follows from the pumping lemma that if there exists a string in L whose length is between n and 2n, then there exist infinitely many strings in L. So, we may use the following algorithm:

1. Go through each string of length between n and 2n. For each string, check if the string is in L. If any such string is in L, declare that there are at least 100 strings and finish. If all strings of length between n and 2n have been checked and none are in L, then proceed to step 2.
2. Now we know that all strings in L are of length less than n. So, we simply enumerate all strings of length 0 to n in order and count the number of strings in
L. Once we have found 100 strings we can declare that L contains at least 100 strings. If we have checked all strings of length less than n and our count is less than 100, then declare that L does not contain at least 100 strings.