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

0. All strings.

<table>
<thead>
<tr>
<th>Solution: ((0 + 1)^*)</th>
<th>Repeatedly write an arbitrary symbol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution: ((1 + 0)^*)</td>
<td>Union is symmetric.</td>
</tr>
<tr>
<td>Solution: ((0^<em>1^</em>)^*)</td>
<td>Repeatedly write any number of 0s followed by any number of 1s.</td>
</tr>
<tr>
<td>Solution: (1^*(00^<em>11^</em>)^<em>0^</em>)</td>
<td>Write any number of 1s, then repeatedly write any positive number of 0s followed by any positive number of 1s, and finally write any number of 0s.</td>
</tr>
<tr>
<td>Solution: ((1^*0)^<em>1^</em>)</td>
<td>Write any number of 1s before each 0, and again at the end.</td>
</tr>
<tr>
<td>Solution: ((00 + 01 + 1)^*(\epsilon + 0))</td>
<td>We can do this all day; every regular language is described by an infinite number of regular expressions!</td>
</tr>
</tbody>
</table>

1. All strings containing the substring 000.

| Solution: \((0 + 1)^* 000 (0 + 1)^*\) | Any string can appear before or after 000. |

2. All strings not containing the substring 000.

| Solution: \((1 + 01 + 001)^*(\epsilon + 0 + 00)\) | Every 1 is immediately preceded by zero, one, or two 0s. |
| Solution: \((\epsilon + 0 + 00)(1(\epsilon + 0 + 00))^*\) | Alternate between 1s and groups of at most two 0s. |
| Solution: \(1^*((0 + 00)11^*)(\epsilon + 0 + 00)\) | Alternate between runs of 1s and runs of at most two 0s. |
3. All strings in which every run of 0s has length at least 3.

Solution: \((1 + 000^*01)^*(\epsilon + 000^*)\)

Write either no 0s or at least three 0s before each 1, and again at the end.

Solution: \((1 + 000^*)^*\)

Whenever you write one 0, write at least three 0s.

Solution: \(1^*(000^*11^*)(\epsilon + 000^*)\)

Alternate between runs of at least three 0s and at least one 1, possibly with extra 1s at the beginning and (at least three) extra 0s at the end.

4. All strings in which every 1 appears before every substring 000.

Solution: \((1 + 01 + 001)^0^*\)

Every 1 is immediately preceded by at most two 0s, but there can be any number of 0s at the end.

5. All strings containing at least three 0s.

Solution: \((0 + 1)^*0(0 + 1)^*0(0 + 1)^*0(0 + 1)^*\)

Any string can appear before, between, or after the three 0s.

Solution (clever): \(1^*01^*01^*0(0 + 1)^*\)

Any number of 1s can appear before or between the first three 0s, and any string can appear after the first three 0s.

Solution (clever): \((0 + 1)^*01^*01^*01^*\)

Look at the last three 0s.

6. Every string except 000. [Hint: Don’t try to be clever.]

Solution: Every string \(w \neq 000\) satisfies one of three conditions: Either \(|w| < 3\), or \(|w| = 3\) and \(w \neq 000\), or \(|w| > 3\). The first two cases include only a finite number of strings, so we just list them explicitly, each case on one line. The expression on the last line includes all strings of length at least 4.

\[
\begin{align*}
\epsilon + 0 + 1 + 00 + 01 + 10 + 11 \\
+ 001 + 010 + 011 + 100 + 101 + 110 + 111 \\
+ (1 + 0)(1 + 0)(1 + 0)(1 + 0)(1 + 0)^*
\end{align*}
\]

Solution (clever): \(\epsilon + 0 + 00 + (1 + 01 + 001 + 000(1 + 0))(1 + 0)^*\)
Work on these later:

7. All strings \( w \) such that in every prefix of \( w \), the numbers of 0s and 1s differ by at most 1.

**Solution:** Equivalently, strings in which every even-length prefix has the same number of 0s and 1s:

\[
(01 + 10)^*(0 + 1 + \varepsilon)
\]

8. All strings containing at least two 0s and at least one 1.

**Solution:** There are three possibilities for how the three required symbols are ordered:

- Contains a 1 before two 0s: \((0 + 1)^* 1 (0 + 1)^* 0 (0 + 1)^* 0 (0 + 1)^*\)
- Contains a 1 between two 0s: \((0 + 1)^* 0 (0 + 1)^* 1 (0 + 1)^* 0 (0 + 1)^*\)
- Contains a 1 after two 0s: \((0 + 1)^* 0 (0 + 1)^* 0 (0 + 1)^* 1 (0 + 1)^*\)

So putting these cases together, we get the following:

\[
(0 + 1)^* 1 (0 + 1)^* 0 (0 + 1)^* 0 (0 + 1)^* + (0 + 1)^* 0 (0 + 1)^* 1 (0 + 1)^* 0 (0 + 1)^* + (0 + 1)^* 0 (0 + 1)^* 0 (0 + 1)^* 1 (0 + 1)^*\]

9. All strings \( w \) such that in every prefix of \( w \), the number of 0s and 1s differ by at most 2.

**Solution:** Call this language \( L \), and let \( w \) be an arbitrary string in \( L \). Call a non-empty substring of \( w \) balanced if it has the same number of 0s and 1s. Our strategy is to decompose \( w \in L \) into as many balanced substrings as possible, followed by at most one unbalanced suffix. We refer to these substrings as chunks. For example, The string

\[w = 00111101010011000101011110101010101\]
consists of four balanced chunks and one unbalanced chunk:

\[ w = 0011 \cdot 11010100 \cdot 110100 \cdot 00101011 \cdot 11010101 \]

Let \( B \) be the set of all possible balanced chunks, and let \( U \) be the set of all possible unbalanced. Let \( B_0 \) be the set of all balanced chunks that start with 0, and let \( B_1 \) be the set of all balanced chunks that start with 1, so that \( B = B_0 + B_1 \). Similarly split \( U \) into subsets \( U_0 \) and \( U_1 \) by first character.

Now we observe that a string \( x \) is in \( B_0 \) if and only if \( x \) satisfies the following conditions:

- \( x \) has the same number of 0s and 1s (so \( |x| \) must be even)
- Every non-empty proper prefix of \( x \) has either 1 or 2 more 0s than 1s.

Every string in \( B_0 \) starts with a 0. If the next symbol is 1, that’s also the last symbol; otherwise, the next symbol is 0, and that cannot be the last symbol (because the string must be balanced). The symbol after that must be a 0 (or we’d have a prefix with too many 0s), and that cannot be the last symbol. After the third symbol, we again have a choice between ending with 1 or continuing with 10. Eventually the string must end with 1. We conclude that

\[ B_0 = 0(01)^*1. \]

Similar reasoning implies

\[ B_1 = 1(10)^*0 \]
\[ U_0 = 0(01)^*(\epsilon + 0) \]
\[ U_1 = 1(10)^*(\epsilon + 1) \]

and therefore

\[ B = B_0 + B_1 = 0(01)^*1 + 1(10)^*0 \]
\[ U = U_0 + U_1 = 0(01)^*(\epsilon + 0) + 1(10)^*(\epsilon + 1) \]

Finally, every string in \( L \) consists of an arbitrary number of balanced chunks, followed by at most one unbalanced chunk, so

\[ L = B^*(\epsilon + U) \]
\[ = (B_0 + B_1)^*(\epsilon + U_0 + U_1) \]
\[ = (0(01)^*1 + 1(10)^*0)^*(\epsilon + 0(01)^*(0 + \epsilon) + 1(10)^*(1 + \epsilon)) \]

\(\square\)
10. All strings in which every run has odd length.

**Solution:** Let $L$ denote our target language. Let $A$ denote the set of all odd-length runs of $0$s, and let $B$ denote the set of all odd-length runs of $1$s. These languages have simple regular expressions

\[ A = (00)^*0 \quad B = (11)^*1 \]

Every binary string alternates between runs of $0$s and runs of $1$s; we are interested in strings that alternate between *odd* runs of $0$s and *odd* runs of $1$s. We can build a regular expression for $L$ by first considering all strings of alternating $A$s and $B$s, and then substituting the regular expressions for $A$ and $B$:

\[ L = (B + \varepsilon)(AB)^*(A + \varepsilon) = (((11)^*1 + \varepsilon)((00)^*0(11)^*1)^*((00)^*0 + \varepsilon) \]

**11.** All strings in which the substring $000$ appears an even number of times.

**Solution:** Let $L$ denote our target language.

Every string in $\{0, 1\}^*$ alternates between (possibly empty) runs of $0$s and individual $1$s; that is, $\{0, 1\}^* = (0^*1)^*0^*$. Trivially, every $000$ substring is contained in some run of $0$s. Our strategy is to consider which runs of $0$s contain an even or odd number of $000$ substrings.

- Let $X$ denote the set of all strings in $0^*$ with an *even* number of $000$ substrings. In particular, we have $\varepsilon \in X$. We easily observe that $X = \{0^n \mid n \leq 1 \text{ or } n \text{ is even}\}$ and thus

\[ X = 0 + (00)^* \]

Notice that the subexpression $(00)^*$ includes the empty string, so we don’t need to include it explicitly in our regular expression.

- Let $Y$ denote the set of all strings in $0^*$ with an *odd* number of $000$ substrings. We easily observe that $Y = \{0^n \mid n > 1 \text{ and } n \text{ is odd}\}$ and thus

\[ Y = 000(00)^* \]

By design, we have $0^* = X + Y$, and therefore

\[ \{0, 1\}^* = (0^*1)^*0^* = ((X + Y)1)^*(X + Y) \]

We are designing a regular expression for the set of binary strings with an *even* number of runs of $0$s in $Y$.

The design problem is easier if we treat $X$ and $Y$ as new symbols, and work over the alphabet $\{X, Y, 1\}$ instead of the original alphabet $\{0, 1\}$. So let $L'$ be the language...
of strings in \( \{X, Y, 1\}^* \) that match the regular expression \((X + Y)1^*(X + Y)\) and contain an even number of \(Y\)s.

- Let \(Z\) denote the set of all strings in \( \{X, Y, 1\}^* \) that start with \(Y\), end with \(Y\), and otherwise alternate between \(X\) and \(1\).

\[
Z = Y(1X)^*1Y
\]

Then we have

\[
L' = ((X + Z)1^*(X + Z) = \left( (X + Y(1X)^*1Y)1^* \right)(X + Y(1X)^*1Y)
\]

Substituting our earlier regular expressions for \(X\) and \(Z\), we conclude that

\[
L = \left( \left( \epsilon + (\epsilon 0)^* + 000(\epsilon 0)^*(1(\epsilon 0 + (\epsilon 0)^*))1000(\epsilon 0)^* \right)1^* \right.
\]

\[
\cdot \left( \left( \epsilon + (\epsilon 0)^* + 000(\epsilon 0)^*(1(\epsilon 0 + (\epsilon 0)^*))1000(\epsilon 0)^* \right) \right)
\]

Whew!