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.

Solution: $(0+1)^*$

Repeatedly write an arbitrary symbol. ■

Solution: $(1+0)^*$

Union is symmetric. ■

Solution: $(0^*1^*)^*$

Repeatedly write any number of 0s followed by any number of 1s. ■

Solution: $1^*(00^*11^*)^*0^*$

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. ■

Solution: $(1^*0)^*1^*$

Write any number of 1s before each 0, and again at the end. ■

Solution: $(00+01+1)^*(\varepsilon+0)$ ■

We can do this all day; every regular language is described by an ***infinite*** *number of regular expressions!*

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: $(1+01+001)^*(\varepsilon+0+00)$

Every 1 is immediately preceded by zero, one, or two 0s. ■

Solution: $(\varepsilon+0+00)(1(\varepsilon+0+00))^*$

Alternate between 1s and groups of at most two 0s. ■

Solution: $1^*((0+00)11^*)^*(\varepsilon+0+00)$

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: $(1 + 0000^*1)^*(\varepsilon + 0000^*)$

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

Solution: $(1 + 0000^*)^*$

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

Solution: $1^*(0000^*11^*)^*(\varepsilon + 0000^*)$

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

4. All strings in which the last 1 appears before the first substring 000.

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

Each 1 is immediately *preceded* by at most two 0s, but there can be any number of 0s after the last 1. ■

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.

$$\begin{gathered}\varepsilon + 0 + 1 + 00 + 01 + 10 + 11 \\ + 001 + 010 + 011 + 100 + 101 + 110 + 111 \\ + (1 + 0)(1 + 0)(1 + 0)(1 + 0)(1 + 0)^*\end{gathered}$$

■

Solution (clever): $\varepsilon + 0 + 00 + \big(1 + 01 + 001 + 000(1 + 0)\big)(1 + 0)^*$ ■

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:
>
> $$\begin{aligned}&(0+1)^*1(0+1)^*0(0+1)^*0(0+1)^*\\+&(0+1)^*0(0+1)^*1(0+1)^*0(0+1)^*\\+&(0+1)^*0(0+1)^*0(0+1)^*1(0+1)^*\end{aligned}$$
>
> ■

> **Solution:** There are three possibilities for how such a string can begin:
>
> - Start with 00, then any number of 0s, then 1, then anything.
> - Start with 01, then any number of 1s, then 0, then anything.
> - Start with 1, then a substring with exactly two 0s, then anything.
>
> All together: $000^*1(0+1)^* + 011^*0(0+1)^* + 11^*01^*0(0+1)^*$
>
> Or equivalently: $\big(000^*1 + 011^*0 + 11^*01^*0\big)(0+1)^*$ ■

> **Solution (clever):** $(0+1)^*\big(101^*0+011^*0+01^*01\big)(0+1)^*$ ■

9. 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 0s, and let B denote the set of all odd-length runs of 1s. These languages have simple regular expressions
>
> $$A=(00)^*0 \qquad B=(11)^*1$$
>
> *Every* binary string alternates between runs of 0s and runs of 1s; we are interested in strings that alternate between *odd* runs of 0s and *odd* runs of 1s. We can build a regular expression for L by first considering all strings of alternating As and Bs, and

then substituting the regular expressions for A and B:

$$\begin{aligned} L &= (B+\varepsilon)(AB)^*(A+\varepsilon) \\ &= ((11)^*1+\varepsilon)((00)^*0(11)^*1)^*((00)^*0+\varepsilon) \end{aligned}$$ ■

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

Solution (from the future): Build the six-state DFA that accepts this language, and then convert that DFA into an equivalent regular expression. Seriously, this is the right way to do it; we just don't have the tools yet. ■

Solution (the hard way): Call this language L, and let w be any string in L. Call a non-empty substring of w *balanced* if it has the same number of 0s and 1s. Our high-level 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

$$w = 00111101010011010000101011110101 01 \in L$$

consists of four balanced chunks and one unbalanced chunk:

$$w = 0011 \cdot 11010100 \cdot 110100 \cdot 00101011 \cdot 11010101$$

Each of the chunks 0011, 11010100, 110100, 00101011, and 11010101 is contained in L.

Let B be the set of all possible balanced chunks, and let U be the set of all possible unbalanced chunks. 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.

(If a strong x satisfies the first condition but not the second, then either x has a prefix that is too unbalanced, or x has a balanced prefix, which implies that we can split x into smaller balanced chunks.) Let's try to build a string $x \in B_0$ one symbol at a time:

- The first symbol in x must be 0, and that cannot be the last symbol of x.
- If the second symbol of x is 1, that's also the last symbol of x. Otherwise, the second symbol is 0, and that cannot be the last symbol of x (because x must be balanced).

- If x begins with 00, the third symbol must be a 1 (or we'd have a prefix with too many 0s), and that cannot be the last symbol of x.
- If x begins with 001, either the fourth and last symbol is 1, or the next two symbols are 01 and those cannot be the last symbols of x.
- Eventually x must end with 1.

In short, every string in B_0 consists of a single 0, followed by any number of 01s, followed by a single 1; equivalently,

$$B_0 = 0(01)^*1.$$

Similar reasoning implies

$$\begin{aligned} B_1 &= 1(10)^*0 \\ U_0 &= 0(01)^*(\varepsilon + 0) \\ U_1 &= 1(10)^*(\varepsilon + 1) \end{aligned}$$

and therefore

$$\begin{aligned} B &= B_0 + B_1 &= 0(01)^*1 + 1(10)^*0 \\ U &= U_0 + U_1 &= 0(01)^*(\varepsilon + 0) + 1(10)^*(\varepsilon + 1) \end{aligned}$$

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

$$\begin{aligned} L &= B^*(\varepsilon + U) \\ &= (B_0 + B_1)^*(\varepsilon + U_0 + U_1) \\ &= \big(0(01)^*1 + 1(10)^*0\big)^*\,\big(\varepsilon + 0(01)^*(0 + \varepsilon) + 1(10)^*(1 + \varepsilon)\big) \end{aligned}$$

Whew! ■

★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 0s and individual 1s; that is, $\{0,1\}^* = (0^*1)^*0^*$. Trivially, every 000 substring is contained in some run of 0s. Our strategy is to consider which runs of 0s 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 $X = \{0^n \mid n \le 1 \text{ or } n \text{ is even}\}$ 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 0s 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 Ys.

- 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
$$\begin{aligned} L' &= ((X+Z)1)^*(X+Z) \\ &= \Big(\big(X + Y(1X)^*1Y\big)1\Big)^*\big(X + Y(1X)^*1Y\big) \end{aligned}$$
Substituting our earlier regular expressions for X and Z, we conclude that
$$\begin{aligned} L = \Big(\big(0 + (00)^* + 000(00)^*(1(0+(00)^*))^*1000(00)^*\big)1\Big)^* \\ \cdot\big(0 + (00)^* + 000(00)^*(1(0+(00)^*))^*1000(00)^*\big) \end{aligned}$$

Whew! ■