Write a program that produces a bowling scorecard.

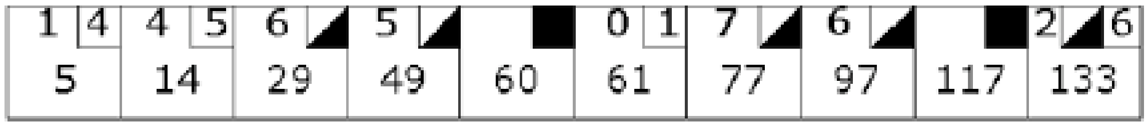
(adapted from Ron Jeffries)

The input is a sequence of throws showing the number of pins knocked down on that throw.

The output is the frame by frame score.

What if the input sequence is not legal?

Sample of what a real score card looks like. Note that frame 10 is different from the others in that you may throw up to three balls in the frame.



1 4	4 5	6 ◢	5 ◢	■	0 1	7 ◢	6 ◢	■	2 ◢ 6
5	14	29	49	60	61	77	97	117	133

input was
1, 4, 4, 5, 6, 4, 5, 5, 10, 0, 1, 7, 3, 6, 4, 10, 2, 8, 6
Output is
5, 14, 29, 49, 60, 61, 77, 97, 117, 133

Create a simple calculator.

The calculator displays a value. A new calculator should have a display of 0.

Give the calculator keys like 5 and 3. When you hit a single key the value of the key should "show" in the display.

If you press a sequence of number keys, the results should accumulate in the display.

Give the calculator a + key that adds the results of the previous two operands. Other operator keys will be added later (-, *, /).

Give the calculator an = key that performs any outstanding operation and clears for a new calculation.

Pressing an operator immediately after equals will continue the current calculation.
Add the rest of the keys: numeric and operator

Create a GUI for the calculator with a button for each key and a text field for the display.

Build a simulator for a deterministic finite state machine.

The DFA has a tape with symbols from a fixed finite alphabet occupying a finite left justified sequence of cells of the tape.

The machine starts in the unique start state reading cell 0.
The machine reads the current cell, advances the read head to the left and determines its current state. It then consults its program (see below) and moves to a new state.

If the machine moves to a final state after reading the last occupied cell on the tape it halts and accepts its input. If it has no program statement corresponding to its current read symbol and state it halts and rejects its input. If it moves to a non-final state after reading its last input symbol it rejects the input.

The tape can be arbitrarily long.

Build a Turing Machine simulator.

A TM has a potentially infinite tape in both directions. It starts reading cell 0 of the tape in its start state. Some finite segment of the tape contains symbols from some fixed finite alphabet that contains a special symbol.

The machine reads the symbol at the current read position and determines its current state. It then consults its program. Based on the program it writes (or not) some symbol from the alphabet on the tape, moves the tape head left, right, or not at all, and changes to another state.

If the machine ever moves into a final state it is said to have produced the current contents of the tape. If the machine has no program statement for the current situation it halts in error.

Create a water vending machine.

(adapted from Alistair Cockburn)

The machine sells Still water and Fizzy water. When the machine is created/initialized, the price of each and the quantity available of each is provided as a parameter.

The machine accepts money. Negative amounts of money are illegal.

The machine correctly makes change.

The machine can return coins inserted to cancel a transaction.

The machine will dispense product and make change when sufficient money has been inserted.

The machine should return the price of each product when asked.

The machine can display how much of each product is available

The machine can display how much money it has taken in for completed sales.

Permit the price of each product to be incremented. Negative increments are illegal.

A functional interface (NO GUI) is all that is required.

Change: The machine only accepts coins in U.S. Denominations.

Create a ticket machine for a transportation system.

(adapted from Barnes and Kölling)

A ticket machine can be created by specifying how much a ticket costs. It can tell you the cost of a ticket on demand.

Comment: The unit of money is the "cent". Partial units are not allowed. All tickets from a given machine cost the same.
Comment: A ticket machine does not require a GUI at this time. It will be manipulated functionally.

A user (i.e. other code) can insert money into a ticket machine and the machine will tally how much as been inserted.

A ticket machine should be able to tell you how much has been collected for the current ticket.

A ticket machine must be able to print tickets.

Comment: Tickets are written to standard output as are other messages, including error messages.

A ticket machine should be able to tell you how much money has been collected altogether.

Here are some more stories that should get you to a better version.

A ticket machine should be able to refund any balance before or after printing a ticket.

A ticket machine should not issue a ticket unless sufficient money has been inserted. It should instead complain.

A ticket machine should complain if you try to give it negative amounts of money.

A ticket machine should keep a running total of monies collected for tickets.

Future stories
Management requires a total money report that contains the number of tickets sold and the total monies collected.

Permit management to reset the total money to 0, presumably after emptying the money box.

Management should be able to increase the fares by incrementing the cost of a ticket.

Negative increments are not allowed.

First class fares have been implemented at twice the usual fare. Permit printing first class tickets.

Provide a GUI for the ticket machine with buttons for all functions except the total and get balance. It should use textfields for input and a text area in which to display the ticket and informational messages. It should show you the balance whenever you correctly insert money.

Error messages should also be shown in the text area.

Create a second GUI for admin functions such as getting the total money, resetting it to 0, etc.

Other challenges we have not tried with others:

Write a program that reads an integer and produces equivalent **Roman Numeral** form.

Write a program that reads a number in **Roman Numeral** form and outputs its value in decimal form.

Write a program that will play **Hangman** against the user.

Choose a word from a wordlist saved in the program. Show one dash character to the user for each letter of the word.

The program must be able to accept a letter from the user. Prompt for it.

When the letter is in the word, the user is shown the current state with the letter replacing the corresponding dash(es) in all positions in which it appears. When the letter is not in the word, the state is unchanged, but shown again to the user.

Make the choice of word random from the list.

Let the program load its wordlist from a file.

Play multiple games, one after the other.

Create a GUI for the program.