Once he has enough data points, he intends to use statistics to find out which of these events may be related to the squirrelifications.

```
addEntry(["work", "touched tree", "pizza", "running", "television"], false);
addEntry(["work", "ice cream", "cauliflower", "lasagna", "touched tree", "brushed teeth"], false);

addEntry(["weekend", "cycling", "break", "peanuts","beer"], true);
```

Correlation is a measure of dependence between statistical variables. A statistical variable is not quite the same as a programming variable. In statistics you typically have a set of measurements, and each variable is measured for every measurement. Correlation between variables is usually expressed as a value that ranges from -1 to 1. Zero correlation means the variables are not related. A correlation of one indicates that the two are perfectly related—if you know one, you also know the other. Negative one also means that the variables are perfectly related but that they are opposites—when one is true, the other is false.

To compute the measure of correlation between two Boolean variables, we can use the *phi coefficient* (ϕ). This is a formula whose input is a frequency table containing the number of times the different combinations of the variables were observed. The output of the formula is a number between -1 and 1 that describes the correlation.

We could take the event of eating pizza and put that in a frequency table like this, where each number indicates the amount of times that combination occurred in our measurements:

No squirrel, no pizza 76	 No squirrel, pizza 9
 Squirrel, no pizza 4	 Squirrel, pizza 1

If we call that table n, we can compute ϕ using the following formula:

$$\phi = \frac{n11n00 - n10n01}{\sqrt{n1 \bullet n0 \bullet n \bullet 1n \bullet 0}}$$

The notation n_{01} indicates the number of measurements where the first variable (squirrelness) is false (0) and the second variable (pizza) is true (1). In the pizza table, n_{01} is 9.

The value $n_{1\bullet}$ refers to the sum of all measurements where the first variable is true, which is 5 in the example table. Likewise, $n_{\bullet 0}$ refers to the sum of the measurements where the second variable is false.

So for the pizza table, the part above the division line (the dividend) would be $1\times76-4\times9 = 40$, and the part below it (the divisor) would be the square root of $5\times85\times10\times80$, or $\sqrt{340000}$. This comes out to $\phi \approx 0.069$, which is tiny. Eating pizza does not appear to have influence on the transformations.

Computing correlation

We can represent a two-by-two table in JavaScript with a four-element array (`[76, 9, 4, 1]`). We could also use other representations, such as an array containing two two-element arrays (`[[76, 9], [4, 1]]`) or an object with property names like `"11"`and `"01"`, but the flat array is simple and makes the expressions that access the table pleasantly short. We'll interpret the indices to the array as two-bit binary numbers, where the leftmost (most significant) digit refers to the squirrel variable and the rightmost (least significant) digit refers to the event variable. For example, the binary number `10` refers to the case where Jacques did turn into a squirrel, but the event (say, "pizza") didn't occur. This happened four times. And since binary `10` is `2` in decimal notation, we will store this number at index 2 of the array.

This is the function that computes the ϕ coefficient from such an array:

```
2   return (table[3] * table[0] - table[2] * table[1]) /
```

```
      Math.sqrt((table[2] + table[3]) *
                (table[0] + table[1]) *
                (table[1] + table[3]) *
                (table[0] + table[2]));
}

console.log(phi([76, 9, 4, 1]));
// → 0.068599434
```

This is a direct translation of the ϕ formula into JavaScript. `Math.sqrt` is the square root function, as provided by the `Math` object in a standard JavaScript environment. We have to add two fields from the table to get fields like $n_{1\bullet}$ because the sums of rows or columns are not stored directly in our data structure.

Jacques kept his journal for three months. The resulting data set is available in the coding sandbox for this chapter, where it is stored in the `JOURNAL` binding and in a downloadable file.

To extract a two-by-two table for a specific event from the journal, we must loop over all the entries and tally how many times the event occurs in relation to squirrel transformations.

```
function tableFor(event, journal) {
  let table = [0, 0, 0, 0];
  for (let i = 0; i < journal.length; i++) {
    let entry = journal[i], index = 0;
    if (entry.events.includes(event)) index += 1;
    if (entry.squirrel) index += 2;
    table[index] += 1;
  }
  return table;
}

console.log(tableFor("pizza", JOURNAL));
// → [76, 9, 4, 1]
```

Arrays have an `includes` method that checks whether a given value exists in the array. The function uses that to determine

whether the event name it is interested in is part of the event list for a given day.

The body of the loop in `tableFor` figures out which box in the table each journal entry falls into by checking whether the entry contains the specific event it's interested in and whether the event happens alongside a squirrel incident. The loop then adds one to the correct box in the table.

We now have the tools we need to compute individual correlations. The only step remaining is to find a correlation for every type of event that was recorded and see whether anything stands out.