Contents

```
}
console.log(total);
// → 55
```

This version gives us a few more improvements. Most importantly, there is no need to specify the way we want the program to jump back and forth anymore. The `while` language construct takes care of that. It continues executing the block (wrapped in braces) below it as long as the condition it was given holds. That condition is `count <= 10`, which means "`count` is less than or equal to 10". We no longer have to create a temporary value and compare that to zero, which was an uninteresting detail. Part of the power of programming languages is that they take care of uninteresting details for us.

At the end of the program, after the `while` construct has finished, the `console.log` operation is applied to the result in order to write it as output.

Finally, here is what the program could look like if we happened to have the convenient operations `range` and `sum` available, which respectively create a collection of numbers within a range and compute the sum of a collection of numbers:

```
console.log(sum(range(1, 10)));
// → 55
```

The moral of this story is that the same program can be expressed in long and short, unreadable and readable ways. The first version of the program was extremely obscure, whereas this last one is almost English: `log` the `sum` of the `range` of numbers from 1 to 10. (We will see in later

accidentally *overflow* such small numbers—to end up with a number that did not fit into the given amount of bits. Today, even personal computers have plenty of memory, so you are free to use 64-bit chunks, which means you need to worry about overflow only when dealing with truly astronomical numbers.

Not all whole numbers below 18 quintillion fit in a JavaScript number, though. Those bits also store negative numbers, so one bit indicates the sign of the number. A bigger issue is that nonwhole numbers must also be represented. To do this, some of the bits are used to store the position of the decimal point. The actual maximum whole number that can be stored is more in the range of 9 quadrillion (15 zeros), which is still pleasantly huge.

Fractional numbers are written by using a dot.

```
9.81
```

For very big or very small numbers, you can also use scientific notation by adding an "e" (for "exponent"), followed by the exponent of the number:

```
2.998e8
```

That is $2.998 \times 10^8 = 299{,}800{,}000$.

Calculations with whole numbers (also called *integers*) smaller than the aforementioned 9 quadrillion are guaranteed to always be precise. Unfortunately, calculations with fractional numbers are generally not. Just as π (pi) cannot be precisely expressed by a finite number of decimal digits, many numbers lose some precision when only 64 bits are

that often aren't what you want or expect. This is called *type coercion*. So the `null` in the first expression becomes `0`, and the `"5"` in the second expression becomes `5` (from string to number). Yet in the third expression, `+` tries string concatenation before numeric addition, so the `1` is converted to `"1"` (from number to string).

When something that doesn't map to a number in an obvious way (such as `"five"` or `undefined`) is converted to a number, the value `NaN` is produced. Further arithmetic operations on `NaN` keep producing `NaN`, so if you find yourself getting one of those in an unexpected place, look for accidental type conversions.

When comparing values of the same type using `==`, the outcome is easy to predict: you should get true when both values are the same, except in the case of `NaN`. But when the types differ, JavaScript uses a complicated and confusing set of rules to determine what to do. In most cases, it just tries to convert one of the values to the other value's type. However, when `null` or `undefined` occurs on either side of the operator, it produces true only if both sides are one of `null` or `undefined`.

```
console.log(null == undefined);
// → true
console.log(null == 0);
// → false
```

That last piece of behavior is often useful. When you want to test whether a value has a real value instead of `null` or `undefined`, you can simply compare it to `null` with the `==` (or `!=`) operator.

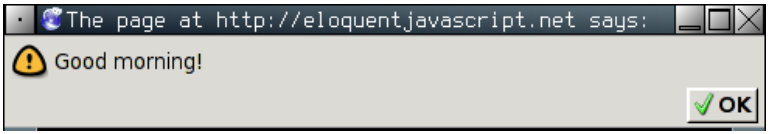
But what if you want to test whether something refers to the precise value `false`? The rules for converting strings and numbers to Boolean

interact with the surrounding system. For example, in a browser, there are variables and functions to inspect and influence the currently loaded website and to read mouse and keyboard input.

Functions

A lot of the values provided in the default environment have the type *function*. A function is a piece of program wrapped in a value. Such values can be *applied* in order to run the wrapped program. For example, in a browser environment, the variable `alert` holds a function that shows a little dialog box with a message. It is used like this:

```
alert("Good morning!");
```



Executing a function is called *invoking*, *calling*, or *applying* it. You can call a function by putting parentheses after an expression that produces a function value. Usually you'll directly use the name of the variable that holds the function. The values between the parentheses are given to the program inside the function. In the example, the `alert` function uses the string that we give it as the text to show in the dialog box. Values given to functions are called *arguments*. The `alert` function needs only one of them, but other functions might need a different number or different types of arguments.

Indenting Code

You've probably noticed the spaces I put in front of some statements. In JavaScript, these are not required—the computer will accept the program just fine without them. In fact, even the line breaks in programs are optional. You could write a program as a single long line if you felt like it. The role of the indentation inside blocks is to make the structure of the code stand out. In complex code, where new blocks are opened inside other blocks, it can become hard to see where one block ends and another begins. With proper indentation, the visual shape of a program corresponds to the shape of the blocks inside it. I like to use two spaces for every open block, but tastes differ—some people use four spaces, and some people use tab characters.

for loops

Many loops follow the pattern seen in the previous `while` examples. First, a "counter" variable is created to track the progress of the loop. Then comes a `while` loop, whose test expression usually checks whether the counter has reached some boundary yet. At the end of the loop body, the counter is updated to track progress.

Because this pattern is so common, JavaScript and similar languages provide a slightly shorter and more comprehensive form, the `for` loop.

```
for (var number = 0; number <= 12; number = number + 2)
  console.log(number);
// → 0
```

a grid of the given width and height.

`console.log` (a built-in browser function), which takes control, does its job, and then returns control to line 2. Then it reaches the end of the `greet` function, so it returns to the place that called it, at line 4. The line after that calls `console.log` again.

We could show the flow of control schematically like this:

```
top
   greet
        console.log
   greet
top
   console.log
top
```

Because a function has to jump back to the place of the call when it returns, the computer must remember the context from which the function was called. In one case, `console.log` has to jump back to the `greet` function. In the other case, it jumps back to the end of the program.

The place where the computer stores this context is the *call stack*. Every time a function is called, the current context is put on top of this "stack". When the function returns, it removes the top context from the stack and uses it to continue execution.

Storing this stack requires space in the computer's memory. When the stack grows too big, the computer will fail with a message like "out of stack space" or "too much recursion". The following code illustrates this by asking the computer a really hard question, which causes an infinite back-and-forth between two functions. Rather, it *would* be infinite, if the computer had an infinite stack. As it is, we will run out of

to understand the program. So we take the repeated functionality, find a good name for it, and put it into a function.

The second way is that you find you need some functionality that you haven't written yet and that sounds like it deserves its own function. You'll start by naming the function, and you'll then write its body. You might even start writing code that uses the function before you actually define the function itself.

How difficult it is to find a good name for a function is a good indication of how clear a concept it is that you're trying to wrap. Let's go through an example.

We want to write a program that prints two numbers, the numbers of cows and chickens on a farm, with the words `Cows` and `Chickens` after them, and zeros padded before both numbers so that they are always three digits long.

```
007 Cows
011 Chickens
```

That clearly asks for a function of two arguments. Let's get coding.

```
function printFarmInventory(cows, chickens) {
  var cowString = String(cows);
  while (cowString.length < 3)
    cowString = "0" + cowString;
  console.log(cowString + " Cows");
  var chickenString = String(chickens);
  while (chickenString.length < 3)
    chickenString = "0" + chickenString;
  console.log(chickenString + " Chickens");
```

```
console.log(listOfNumbers[2 - 1]);
// → 3
```

The notation for getting at the elements inside an array also uses square brackets. A pair of square brackets immediately after an expression, with another expression inside of them, will look up the element in the left-hand expression that corresponds to the *index* given by the expression in the brackets.

The first index of an array is zero, not one. So the first element can be read with `listOfNumbers[0]`. If you don't have a programming background, this convention might take some getting used to. But zero-based counting has a long tradition in technology, and as long as this convention is followed consistently (which it is, in JavaScript), it works well.

Properties

We've seen a few suspicious-looking expressions like `myString.length` (to get the length of a string) and `Math.max` (the maximum function) in past examples. These are expressions that access a *property* of some value. In the first case, we access the `length` property of the value in `myString`. In the second, we access the property named `max` in the `Math` object (which is a collection of mathematics-related values and functions).

Almost all JavaScript values have properties. The exceptions are `null` and `undefined`. If you try to access a property on one of these nonvalues, you get an error.

```
    squirrel: didITurnIntoASquirrel
  });
}
```

And then, every evening at ten—or sometimes the next morning, after climbing down from the top shelf of his bookcase—he records the day.

```
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);
```

Once he has enough data points, he intends to compute the correlation between his squirrelification and each of the day's events and ideally learn something useful from those correlations.

Correlation is a measure of dependence between variables ("variables" in the statistical sense, not the JavaScript sense). It is usually expressed as a coefficient that ranges from -1 to 1. Zero correlation means the variables are not related, whereas 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.

For binary (Boolean) variables, the *phi* coefficient (φ) provides a good measure of correlation and is relatively easy to compute. To compute φ, we need a table n that contains the number of times the various combinations of the two variables were observed. For example, we could take the event of eating pizza and put that in a table like this:

```
// → 1
console.log([1, 2, 3, 2, 1].lastIndexOf(2));
// → 3
```

Both `indexOf` and `lastIndexOf` take an optional second argument that indicates where to start searching from.

Another fundamental method is `slice`, which takes a start index and an end index and returns an array that has only the elements between those indices. The start index is inclusive, the end index exclusive.

```
console.log([0, 1, 2, 3, 4].slice(2, 4));
// → [2, 3]
console.log([0, 1, 2, 3, 4].slice(2));
// → [2, 3, 4]
```

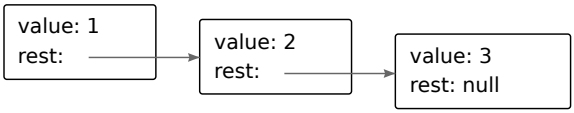
When the end index is not given, `slice` will take all of the elements after the start index. Strings also have a `slice` method, which has a similar effect.

The `concat` method can be used to glue arrays together, similar to what the `+` operator does for strings. The following example shows both `concat` and `slice` in action. It takes an array and an index, and it returns a new array that is a copy of the original array with the element at the given index removed.

```
function remove(array, index) {
  return array.slice(0, index)
    .concat(array.slice(index + 1));
}
console.log(remove(["a", "b", "c", "d", "e"], 2));
// → ["a", "b", "d", "e"]
```

```
      rest: null
    }
  }
};
```

The resulting objects form a chain, like this:



A nice thing about lists is that they can share parts of their structure. For example, if I create two new values `{value: 0, rest: list}` and `{value: -1, rest: list}` (with `list` referring to the variable defined earlier), they are both independent lists, but they share the structure that makes up their last three elements. In addition, the original list is also still a valid three-element list.

Write a function `arrayToList` that builds up a data structure like the previous one when given `[1, 2, 3]` as argument, and write a `listToArray` function that produces an array from a list. Also write the helper functions `prepend`, which takes an element and a list and creates a new list that adds the element to the front of the input list, and `nth`, which takes a list and a number and returns the element at the given position in the list, or `undefined` when there is no such element.

If you haven't already, also write a recursive version of `nth`.

Passing along arguments

The `noisy` function defined earlier, which wraps its argument in another function, has a rather serious deficit.

```
function noisy(f) {
  return function(arg) {
    console.log("calling with", arg);
    var val = f(arg);
    console.log("called with", arg, "- got", val);
    return val;
  };
}
```

If `f` takes more than one parameter, it gets only the first one. We could add a bunch of arguments to the inner function (`arg1`, `arg2`, and so on) and pass them all to `f`, but it is not clear how many would be enough. This solution would also deprive `f` of the information in `arguments.length`. Since we'd always pass the same number of arguments, it wouldn't know how many arguments were originally given.

For these kinds of situations, JavaScript functions have an `apply` method. You pass it an array (or array-like object) of arguments, and it will call the function with those arguments.

```
function transparentWrapping(f) {
  return function() {
    return f.apply(null, arguments);
  };
}
```

be traced to only a small part of the code, which sits inside an inner loop.

Great-great-great-great-...

My grandfather, Philibert Haverbeke, is included in the data file. By starting with him, I can trace my lineage to find out whether the most ancient person in the data, Pauwels van Haverbeke, is my direct ancestor. And if he is, I would like to know how much DNA I theoretically share with him.

To be able to go from a parent's name to the actual object that represents this person, we first build up an object that associates names with people.

```
var byName = {};
ancestry.forEach(function(person) {
  byName[person.name] = person;
});

console.log(byName["Philibert Haverbeke"]);
// → {name: "Philibert Haverbeke", ...}
```

Now, the problem is not entirely as simple as following the `father` properties and counting how many we need to reach Pauwels. There are several cases in the family tree where people married their second cousins (tiny villages and all that). This causes the branches of the family tree to rejoin in a few places, which means I share more than $1/2^G$ of my genes with this person, where G for the number of generations between

That kind of zealotry always produces a lot of impractical silliness, and there has been a sort of counter-revolution since then. In some circles, objects have a rather bad reputation nowadays.

I prefer to look at the issue from a practical, rather than ideological, angle. There are several useful concepts, most importantly that of *encapsulation* (distinguishing between internal complexity and external interface), that the object-oriented culture has popularized. These are worth studying.

This chapter describes JavaScript's rather eccentric take on objects and the way they relate to some classical object-oriented techniques.

Methods

Methods are simply properties that hold function values. This is a simple method:

```
var rabbit = {};
rabbit.speak = function(line) {
  console.log("The rabbit says '" + line + "'");
};

rabbit.speak("I'm alive.");
// → The rabbit says 'I'm alive.'
```

Usually a method needs to do something with the object it was called on. When a function is called as a method—looked up as a property and immediately called, as in `object.method()`—the special variable `this` in its body will point to the object that it was called on.

This method tells us whether the object *itself* has the property, without looking at its prototypes. This is often a more useful piece of information than what the `in` operator gives us.

When you are worried that someone (some other code you loaded into your program) might have messed with the base object prototype, I recommend you write your `for`/`in` loops like this:

```
for (var name in map) {
  if (map.hasOwnProperty(name)) {
    // ... this is an own property
  }
}
```

Prototype-less objects

But the rabbit hole doesn't end there. What if someone registered the name `hasOwnProperty` in our `map` object and set it to the value 42? Now the call to `map.hasOwnProperty` will try to call the local property, which holds a number, not a function.

In such a case, prototypes just get in the way, and we would actually prefer to have objects without prototypes. We saw the `Object.create` function, which allows us to create an object with a specific prototype. You are allowed to pass `null` as the prototype to create a fresh object with no prototype. For objects like `map`, where the properties could be anything, this is exactly what we want.

```
var map = Object.create(null);
```

The resulting table resembles the example shown before, except that it does not right-align the numbers in the `height` column. We will get to that in a moment.

Getters and setters

When specifying an interface, it is possible to include properties that are not methods. We could have defined `minHeight` and `minWidth` to simply hold numbers. But that'd have required us to compute them in the constructor, which adds code there that isn't strictly relevant to *constructing* the object. It would cause problems if, for example, the inner cell of an underlined cell was changed, at which point the size of the underlined cell should also change.

This has led some people to adopt a principle of never including non-method properties in interfaces. Rather than directly access a simple value property, they'd use `getSomething` and `setSomething` methods to read and write the property. This approach has the downside that you will end up writing—and reading—a lot of additional methods.

Fortunately, JavaScript provides a technique that gets us the best of both worlds. We can specify properties that, from the outside, look like normal properties but secretly have methods associated with them.

```
var pile = {
  elements: ["eggshell", "orange peel", "worm"],
  get height() {
    return this.elements.length;
  },
```

Representing space

The grid that models the world has a fixed width and height. Squares are identified by their x- and y-coordinates. We use a simple type, `Vector` (as seen in the exercises for the previous chapter), to represent these coordinate pairs.

```
function Vector(x, y) {
  this.x = x;
  this.y = y;
}
Vector.prototype.plus = function(other) {
  return new Vector(this.x + other.x, this.y + other.y);
};
```

Next, we need an object type that models the grid itself. A grid is part of a world, but we are making it a separate object (which will be a property of a world object) to keep the world object itself simple. The world should concern itself with world-related things, and the grid should concern itself with grid-related things.

To store a grid of values, we have several options. We can use an array of row arrays and use two property accesses to get to a specific square, like this:

```
var grid = [["top left",    "top middle",    "top right"],
            ["bottom left", "bottom middle", "bottom right"]];
console.log(grid[1][2]);
// → bottom right
```

Or we can use a single array, with size width × height, and decide that

```
    }
  }
};
```

Animating life

The next step is to write a `turn` method for the world object that gives the critters a chance to act. It will go over the grid using the `forEach` method we just defined, looking for objects with an `act` method. When it finds one, `turn` calls that method to get an action object and carries out the action when it is valid. For now, only `"move"` actions are understood.

There is one potential problem with this approach. Can you spot it? If we let critters move as we come across them, they may move to a square that we haven't looked at yet, and we'll allow them to move *again* when we reach that square. Thus, we have to keep an array of critters that have already had their turn and ignore them when we see them again.

```
World.prototype.turn = function() {
  var acted = [];
  this.grid.forEach(function(critter, vector) {
    if (critter.act && acted.indexOf(critter) == -1) {
      acted.push(critter);
      this.letAct(critter, vector);
    }
  }, this);
};
```

returns false to indicate no action was taken. Otherwise, it moves the critter and subtracts the energy cost.

In addition to moving, critters can eat.

```
actionTypes.eat = function(critter, vector, action) {
  var dest = this.checkDestination(action, vector);
  var atDest = dest != null && this.grid.get(dest);
  if (!atDest || atDest.energy == null)
    return false;
  critter.energy += atDest.energy;
  this.grid.set(dest, null);
  return true;
};
```

Eating another critter also involves providing a valid destination square. This time, the destination must not be empty and must contain something with energy, like a critter (but not a wall—walls are not edible). If so, the energy from the eaten is transferred to the eater, and the victim is removed from the grid.

And finally, we allow our critters to reproduce.

```
actionTypes.reproduce = function(critter, vector, action) {
  var baby = elementFromChar(this.legend,
                             critter.originChar);
  var dest = this.checkDestination(action, vector);
  if (dest == null ||
      critter.energy <= 2 * baby.energy ||
      this.grid.get(dest) != null)
    return false;
  critter.energy -= 2 * baby.energy;
```

For example, consider the following code, which calls a constructor without the `new` keyword so that its `this` will *not* refer to a newly constructed object:

```
function Person(name) { this.name = name; }
var ferdinand = Person("Ferdinand"); // oops
console.log(name);
// → Ferdinand
```

So the bogus call to `Person` succeeded but returned an undefined value and created the global variable `name`. In strict mode, the result is different.

```
"use strict";
function Person(name) { this.name = name; }
// Oops, forgot 'new'
var ferdinand = Person("Ferdinand");
// → TypeError: Cannot set property 'name' of undefined
```

We are immediately told that something is wrong. This is helpful.

Strict mode does a few more things. It disallows giving a function multiple parameters with the same name and removes certain problematic language features entirely (such as the `with` statement, which is so misguided it is not further discussed in this book).

In short, putting a `"use strict"` at the top of your program rarely hurts and might help you spot a problem.

block.

```
function withContext(newContext, body) {
  var oldContext = context;
  context = newContext;
  try {
    return body();
  } finally {
    context = oldContext;
  }
}
```

Note that we no longer have to store the result of `body` (which we want to return) in a variable. Even if we return directly from the `try` block, the `finally` block will be run. Now we can do this and be safe:

```
try {
  withContext(5, function() {
    if (context < 10)
      throw new Error("Not enough context!");
  });
} catch (e) {
  console.log("Ignoring: " + e);
}
// → Ignoring: Error: Not enough context!

console.log(context);
// → null
```

Even though the function called from `withContext` exploded, `withContext` itself still properly cleaned up the `context` variable.

Testing for matches

Regular expression objects have a number of methods. The simplest one is `test`. If you pass it a string, it will return a Boolean telling you whether the string contains a match of the pattern in the expression.

```
console.log(/abc/.test("abcde"));
// → true
console.log(/abc/.test("abxde"));
// → false
```

A regular expression consisting of only nonspecial characters simply represents that sequence of characters. If *abc* occurs anywhere in the string we are testing against (not just at the start), `test` will return `true`.

Matching a set of characters

Finding out whether a string contains *abc* could just as well be done with a call to `indexOf`. Regular expressions allow us to go beyond that and express more complicated patterns.

Say we want to match any number. In a regular expression, putting a set of characters between square brackets makes that part of the expression match any of the characters between the brackets.

Both of the following expressions match all strings that contain a digit:

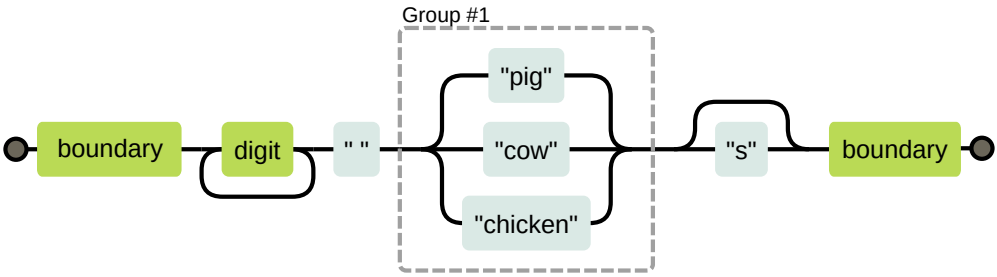
```
console.log(/[0123456789]/.test("in 1992"));
// → true
```

```
// → false
```

Parentheses can be used to limit the part of the pattern that the pipe operator applies to, and you can put multiple such operators next to each other to express a choice between more than two patterns.

The mechanics of matching

Regular expressions can be thought of as flow diagrams. This is the diagram for the livestock expression in the previous example:



Our expression matches a string if we can find a path from the left side of the diagram to the right side. We keep a current position in the string, and every time we move through a box, we verify that the part of the string after our current position matches that box.

So if we try to match `"the 3 pigs"` with our regular expression, our progress through the flow chart would look like this:

- At position 4, there is a word boundary, so we can move past the first box.

because we are writing them in a normal string, not a slash-enclosed regular expression. The second argument to the `RegExp` constructor contains the options for the regular expression—in this case `"gi"` for global and case-insensitive.

But what if the name is `"dea+hl[]rd"` because our user is a nerdy teenager? That would result in a nonsensical regular expression, which won't actually match the user's name.

To work around this, we can add backslashes before any character that we don't trust. Adding backslashes before alphabetic characters is a bad idea because things like `\b` and `\n` have a special meaning. But escaping everything that's not alphanumeric or whitespace is safe.

```
var name = "dea+hl[]rd";
var text = "This dea+hl[]rd guy is super annoying.";
var escaped = name.replace(/[^\w\s]/g, "\\$&");
var regexp = new RegExp("\\b(" + escaped + ")\\b", "gi");
console.log(text.replace(regexp, "_$1_"));
// → This _dea+hl[]rd_ guy is super annoying.
```

The search method

The `indexOf` method on strings cannot be called with a regular expression. But there is another method, `search`, which does expect a regular expression. Like `indexOf`, it returns the first index on which the expression was found, or -1 when it wasn't found.

```
console.log("  word".search(/\S/));
```

practice of writing as tiny a regular expression as possible to match a given pattern, and *only* that pattern.

For each of the following items, write a regular expression to test whether any of the given substrings occur in a string. The regular expression should match only strings containing one of the substrings described. Do not worry about word boundaries unless explicitly mentioned. When your expression works, see whether you can make it any smaller.

1. *car* and *cat*
2. *pop* and *prop*
3. *ferret*, *ferry*, and *ferrari*
4. Any word ending in *ious*
5. A whitespace character followed by a dot, comma, colon, or semicolon
6. A word longer than six letters
7. A word without the letter *e*

Refer to the table in the chapter summary for help. Test each solution with a few test strings.

```
console.log(weekDay.name(weekDay.number("Saturday")));
// → Saturday
```

Detaching from the global scope

The previous pattern is commonly used by JavaScript modules intended for the browser. The module will claim a single global variable and wrap its code in a function in order to have its own private namespace. But this pattern still causes problems if multiple modules happen to claim the same name or if you want to load two versions of a module alongside each other.

With a little plumbing, we can create a system that allows one module to directly ask for the interface object of another module, without going through the global scope. Our goal is a `require` function that, when given a module name, will load that module's file (from disk or the Web, depending on the platform we are running on) and return the appropriate interface value.

This approach solves the problems mentioned previously and has the added benefit of making your program's dependencies explicit, making it harder to accidentally make use of some module without stating that you need it.

For `require` we need two things. First, we want a function `readFile`, which returns the content of a given file as a string. (A single such function is not present in standard JavaScript, but different JavaScript environments, such as the browser and Node.js, provide their own ways

works and what doesn't. Never assume that a painful interface is "just the way it is". Fix it, or wrap it in a new interface that works better for you.

Predictability

If programmers can predict the way your interface works, they (or you) won't get sidetracked as often by the need to look up how to use it. Thus, try to follow conventions. When there is another module or part of the standard JavaScript environment that does something similar to what you are implementing, it might be a good idea to make your interface resemble the existing interface. That way, it'll feel familiar to people who know the existing interface.

Another area where predictability is important is the actual *behavior* of your code. It can be tempting to make an unnecessarily clever interface with the justification that it's more convenient to use. For example, you could accept all kinds of different types and combinations of arguments and do the "right thing" for all of them. Or you could provide dozens of specialized convenience functions that provide slightly different flavors of your module's functionality. These might make code that builds on your interface slightly shorter, but they will also make it much harder for people to build a clear mental model of the module's behavior.

We define a function `parseExpression`, which takes a string as input and returns an object containing the data structure for the expression at the start of the string, along with the part of the string left after parsing this expression. When parsing subexpressions (the argument to an application, for example), this function can be called again, yielding the argument expression as well as the text that remains. This text may in turn contain more arguments or may be the closing parenthesis that ends the list of arguments.

This is the first part of the parser:

```
function parseExpression(program) {
  program = skipSpace(program);
  var match, expr;
  if (match = /^"([^"]*)"/.exec(program))
    expr = {type: "value", value: match[1]};
  else if (match = /^\d+\b/.exec(program))
    expr = {type: "value", value: Number(match[0])};
  else if (match = /^[^\s(),"]+/.exec(program))
    expr = {type: "word", name: match[0]};
  else
    throw new SyntaxError("Unexpected syntax: " + program);

  return parseApply(expr, program.slice(match[0].length));
}

function skipSpace(string) {
  var first = string.search(/\S/);
  if (first == -1) return "";
  return string.slice(first);
```

implemented in less than 150 lines of code.

Functions

A programming language without functions is a poor programming language indeed.

Fortunately, it is not hard to add a `fun` construct, which treats its last argument as the function's body and treats all the arguments before that as the names of the function's arguments.

```
specialForms["fun"] = function(args, env) {
  if (!args.length)
    throw new SyntaxError("Functions need a body");
  function name(expr) {
    if (expr.type != "word")
      throw new SyntaxError("Arg names must be words");
    return expr.name;
  }
  var argNames = args.slice(0, args.length - 1).map(name);
  var body = args[args.length - 1];

  return function() {
    if (arguments.length != argNames.length)
      throw new TypeError("Wrong number of arguments");
    var localEnv = Object.create(env);
    for (var i = 0; i < arguments.length; i++)
      localEnv[argNames[i]] = arguments[i];
    return evaluate(body, localEnv);
  };
```

The Web

The *World Wide Web* (not to be confused with the Internet as a whole) is a set of protocols and formats that allow us to visit web pages in a browser. The "Web" part in the name refers to the fact that such pages can easily link to each other, thus connecting into a huge mesh that users can move through.

To add content to the Web, all you need to do is connect a machine to the Internet, and have it listen on port 80, using the *Hypertext Transfer Protocol* (HTTP). This protocol allows other computers to request documents over the network.

Each document on the Web is named by a *Uniform Resource Locator* (URL), which looks something like this:

```
 http://eloquentjavascript.net/12_browser.html
|      |                       |               |
protocol       server               path
```

The first part tells us that this URL uses the HTTP protocol (as opposed to, for example, encrypted HTTP, which would be *https://*). Then comes the part that identifies which server we are requesting the document from. Last is a path string that identifies the specific document (or *resource*) we are interested in.

Each machine connected to the Internet gets a unique *IP address*, which looks something like `37.187.37.82`. You can use these directly as the server part of a URL. But lists of more or less random numbers are hard to remember and awkward to type, so you can instead register a *domain name* to point toward a specific machine or set of machines. I

```
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```

This page has the following structure:

```
<p><button onclick="replaceImages()">Replace</button></p>

<script>
  function replaceImages() {
    var images = document.body.getElementsByTagName("img");
    for (var i = images.length - 1; i >= 0; i--) {
      var image = images[i];
      if (image.alt) {
        var text = document.createTextNode(image.alt);
        image.parentNode.replaceChild(text, image);
      }
    }
  }
</script>
```

Given a string, `createTextNode` gives us a type 3 DOM node (a text node), which we can insert into the document to make it show up on the screen.

The loop that goes over the images starts at the end of the list of nodes. This is necessary because the node list returned by a method like `getElementsByTagName` (or a property like `childNodes`) is *live*. That is, it is updated as the document changes. If we started from the front, removing the first image would cause the list to lose its first element so that the second time the loop repeats, where `i` is 1, it would stop because the length of the collection is now also 1.

If you want a *solid* collection of nodes, as opposed to a live one, you can convert the collection to a real array by calling the array `slice` method on it.

at all—`display: none` prevents an element from showing up on the screen. This is a way to hide elements. It is often preferable to removing them from the document entirely because it makes it easy to reveal them again at a later time.

This text is displayed **inline**,
as a block
, and .

JavaScript code can directly manipulate the style of an element through the node's `style` property. This property holds an object that has properties for all possible style properties. The values of these properties are strings, which we can write to in order to change a particular aspect of the element's style.

```
<p id="para" style="color: purple">
  Pretty text
</p>

<script>
  var para = document.getElementById("para");
  console.log(para.style.color);
  para.style.color = "magenta";
</script>
```

Some style property names contain dashes, such as `font-family`. Because such property names are awkward to work with in JavaScript (you'd have to say `style["font-family"]`), the property names in the `style` object for such properties have their dashes removed and the letters that follow

Elements by tag name

The `getElementsByTagName` method returns all child elements with a given tag name. Implement your own version of it as a regular nonmethod function that takes a node and a string (the tag name) as arguments and returns an array containing all descendant element nodes with the given tag name.

To find the tag name of an element, use its `tagName` property. But note that this will return the tag name in all uppercase. Use the `toLowerCase` or `toUpperCase` string method to compensate for this.

The cat's hat

Extend the cat animation defined earlier so that both the cat and his hat (`<img src="img/hat.png">`) orbit at opposite sides of the ellipse.

Or make the hat circle around the cat. Or alter the animation in some other interesting way.

To make positioning multiple objects easier, it is probably a good idea to switch to absolute positioning. This means that `top` and `left` are counted relative to the top left of the document. To avoid using negative coordinates, you can simply add a fixed number of pixels to the position values.

```
  });
</script>
```

The "keydown" and "keyup" events give you information about the physical key that is being pressed. But what if you are interested in the actual text being typed? Getting that text from key codes is awkward. Instead, there exists another event, "keypress", which fires right after "keydown" (and repeated along with "keydown" when the key is held) but only for keys that produce character input. The charCode property in the event object contains a code that can be interpreted as a Unicode character code. We can use the String.fromCharCode function to turn this code into an actual single-character string.

```
<p>Focus this page and type something.</p>
<script>
  addEventListener("keypress", function(event) {
    console.log(String.fromCharCode(event.charCode));
  });
</script>
```

The DOM node where a key event originates depends on the element that has focus when the key is pressed. Normal nodes cannot have focus (unless you give them a tabindex attribute), but things such as links, buttons, and form fields can. We'll come back to form fields in Chapter 18. When nothing in particular has focus, document.body acts as the target node of key events.

that the content of `<script>` tags is run immediately when the tag is encountered. This is often too soon, such as when the script needs to do something with parts of the document that appear after the `<script>` tag.

Elements such as images and script tags that load an external file also have a `"load"` event that indicates the files they reference were loaded. Like the focus-related events, loading events do not propagate.

When a page is closed or navigated away from (for example by following a link), a `"beforeunload"` event fires. The main use of this event is to prevent the user from accidentally losing work by closing a document. Preventing the page from unloading is not, as you might expect, done with the `preventDefault` method. Instead, it is done by returning a string from the handler. The string will be used in a dialog that asks the user if they want to stay on the page or leave it. This mechanism ensures that a user is able to leave the page, even if it is running a malicious script that would prefer to keep them there forever in order to force them to look at dodgy weight loss ads.

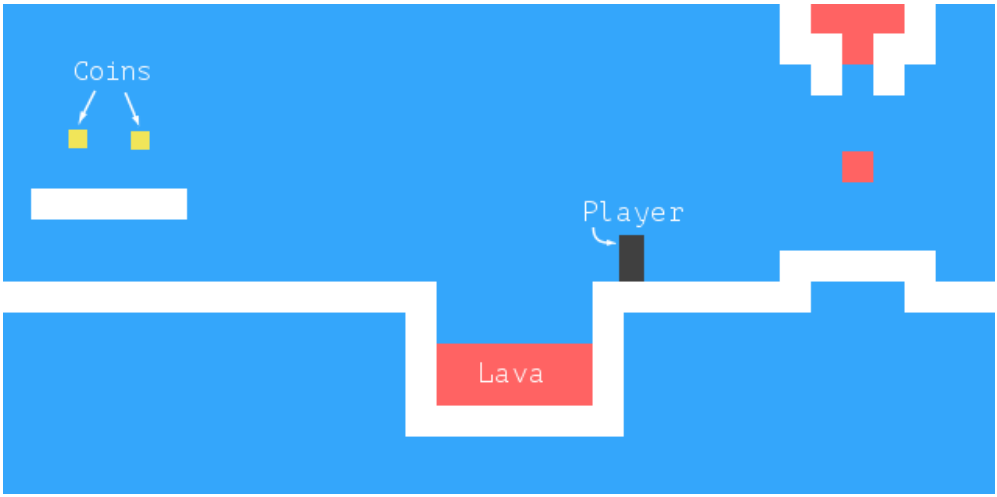
Script execution timeline

There are various things that can cause a script to start executing. Reading a `<script>` tag is one such thing. An event firing is another. Chapter 13 discussed the `requestAnimationFrame` function, which schedules a function to be called before the next page redraw. That is yet another way in which a script can start running.

It is important to understand that even though events can fire at

The game

Our game will be roughly based on Dark Blue (*www.lessmilk.com/games/1*
by Thomas Palef. I chose this game because it is both entertaining and minimalist, and because it can be built without too much code. It looks like this:



The dark box represents the player, whose task is to collect the yellow boxes (coins) while avoiding the red stuff (lava?). A level is completed when all coins have been collected.

The player can walk around with the left and right arrow keys and jump with the up arrow. Jumping is a specialty of this game character. It can reach several times its own height and is able to change direction in midair. This may not be entirely realistic, but it helps give the player the feeling of being in direct control of the onscreen avatar.

Drawing

The encapsulation of the drawing code is done by defining a *display* object, which displays a given level. The display type we define in this chapter is called `DOMDisplay` because it uses simple DOM elements to show the level.

We will be using a style sheet to set the actual colors and other fixed properties of the elements that make up the game. It would also be possible to directly assign to the elements' `style` property when we create them, but that would produce more verbose programs.

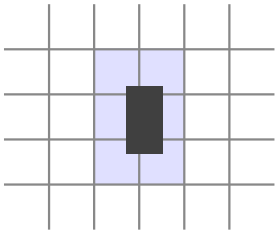
The following helper function provides a short way to create an element and give it a class:

```
function elt(name, className) {
  var elt = document.createElement(name);
  if (className) elt.className = className;
  return elt;
}
```

A display is created by giving it a parent element to which it should append itself and a level object.

```
function DOMDisplay(parent, level) {
  this.wrap = parent.appendChild(elt("div", "game"));
  this.level = level;

  this.wrap.appendChild(this.drawBackground());
  this.actorLayer = null;
  this.drawFrame();
```



If the body sticks out of the level, we always return `"wall"` for the sides and top and `"lava"` for the bottom. This ensures that the player dies when falling out of the world. When the body is fully inside the grid, we loop over the block of grid squares found by rounding the coordinates and return the content of the first nonempty square we find.

Collisions between the player and other dynamic actors (coins, moving lava) are handled *after* the player moved. When the motion has taken the player into another actor, the appropriate effect—collecting a coin or dying—is activated.

This method scans the array of actors, looking for an actor that overlaps the one given as an argument:

```
Level.prototype.actorAt = function(actor) {
  for (var i = 0; i < this.actors.length; i++) {
    var other = this.actors[i];
    if (other != actor &&
        actor.pos.x + actor.size.x > other.pos.x &&
        actor.pos.x < other.pos.x + other.size.x &&
        actor.pos.y + actor.size.y > other.pos.y &&
        actor.pos.y < other.pos.y + other.size.y)
      return other;
  }
```

```
      return false;
    }
  });
}
```

A game is a sequence of levels. Whenever the player dies, the current level is restarted. When a level is completed, we move on to the next level. This can be expressed by the following function, which takes an array of level plans (arrays of strings) and a display constructor:

```
function runGame(plans, Display) {
  function startLevel(n) {
    runLevel(new Level(plans[n]), Display, function(status) {
      if (status == "lost")
        startLevel(n);
      else if (n < plans.length - 1)
        startLevel(n + 1);
      else
        console.log("You win!");
    });
  }
  startLevel(0);
}
```

These functions show a peculiar style of programming. Both `runAnimation` and `runLevel` are higher-order functions but are not in the style we saw in Chapter 5. The function argument is used to arrange things to happen at some time in the future, and neither of the functions returns anything useful. Their task is, in a way, to schedule actions. Wrapping these actions in functions gives us a way to store them as a value so

```
<script>
  var cx = document.querySelector("canvas").getContext("2d");
  cx.beginPath();
  cx.moveTo(50, 10);
  cx.lineTo(10, 70);
  cx.lineTo(90, 70);
  cx.fill();
</script>
```

This example draws a filled triangle. Note that only two of the triangle's sides are explicitly drawn. The third, from the bottom-right corner back to the top, is implied and won't be there when you stroke the path.

You could also use the `closePath` method to explicitly close a path by adding an actual line segment back to the path's start. This segment *is* drawn when stroking the path.

Curves

A path may also contain curved lines. These are, unfortunately, a bit more involved to draw than straight lines.

The `quadraticCurveTo` method draws a curve to a given point. To determine the curvature of the line, the method is given a control point as

When `drawImage` is given *nine* arguments, it can be used to draw only a fragment of an image. The second through fifth arguments indicate the rectangle (x, y, width, and height) in the source image that should be copied, and the sixth to ninth arguments give the rectangle (on the canvas) into which it should be copied.

This can be used to pack multiple *sprites* (image elements) into a single image file and then draw only the part you need. For example, we have this picture containing a game character in multiple poses:

By alternating which pose we draw, we can show an animation that looks like a walking character.

To animate the picture on a canvas, the `clearRect` method is useful. It resembles `fillRect`, but instead of coloring the rectangle, it makes it transparent, removing the previously drawn pixels.

We know that each *sprite*, each subpicture, is 24 pixels wide and 30 pixels high. The following code loads the image and then sets up an interval (repeated timer) to draw the next *frame*:

```
<canvas></canvas>
<script>
  var cx = document.querySelector("canvas").getContext("2d");
  var img = document.createElement("img");
  img.src = "img/player.png";
  var spriteW = 24, spriteH = 30;
  img.addEventListener("load", function() {
    var cycle = 0;
```

```
  var view = this.viewport, margin = view.width / 3;
  var player = this.level.player;
  var center = player.pos.plus(player.size.times(0.5));

  if (center.x < view.left + margin)
    view.left = Math.max(center.x - margin, 0);
  else if (center.x > view.left + view.width - margin)
    view.left = Math.min(center.x + margin - view.width,
                         this.level.width - view.width);
  if (center.y < view.top + margin)
    view.top = Math.max(center.y - margin, 0);
  else if (center.y > view.top + view.height - margin)
    view.top = Math.min(center.y + margin - view.height,
                        this.level.height - view.height);
};
```

The calls to `Math.max` and `Math.min` ensure that the viewport does not end up showing space outside of the level. `Math.max(x, 0)` ensures that the resulting number is not less than zero. `Math.min`, similarly, ensures a value stays below a given bound.

When clearing the display, we'll use a slightly different color depending on whether the game is won (brighter) or lost (darker).

```
CanvasDisplay.prototype.clearDisplay = function() {
  if (this.level.status == "won")
    this.cx.fillStyle = "rgb(68, 191, 255)";
  else if (this.level.status == "lost")
    this.cx.fillStyle = "rgb(44, 136, 214)";
  else
    this.cx.fillStyle = "rgb(52, 166, 251)";
```

Precomputed mirroring

One unfortunate thing about transformations is that they slow down drawing of bitmaps. For vector graphics, the effect is less serious since only a few points (for example, the center of a circle) need to be transformed, after which drawing can happen as normal. For a bitmap image, the position of each pixel has to be transformed, and though it is possible that browsers will get more clever about this in the future, this currently causes a measurable increase in the time it takes to draw a bitmap.

In a game like ours, where we are drawing only a single transformed sprite, this is a nonissue. But imagine that we need to draw hundreds of characters or thousands of rotating particles from an explosion.

Think of a way to allow us to draw an inverted character without loading additional image files and without having to make transformed `drawImage` calls every frame.

response. We need a mechanism that will notify us when the data is available.

For this, we must listen for the `"load"` event on the request object.

```
var req = new XMLHttpRequest();
req.open("GET", "example/data.txt", true);
req.addEventListener("load", function() {
  console.log("Done:", req.status);
});
req.send(null);
```

Just like the use of `requestAnimationFrame` in Chapter 15, this forces us to use an asynchronous style of programming, wrapping the things that have to be done after the request in a function and arranging for that to be called at the appropriate time. We will come back to this later.

Fetching XML Data

When the resource retrieved by an `XMLHttpRequest` object is an XML document, the object's `responseXML` property will hold a parsed representation of this document. This representation works much like the DOM discussed in Chapter 13, except that it doesn't have HTML-specific functionality like the `style` property. The object that `responseXML` holds corresponds to the `document` object. Its `documentElement` property refers to the outer tag of the XML document. In the following document (*example/fruit.xml*), that would be the `<fruits>` tag:

```
<fruits>
```

The resulting program is relatively compact and readable. The `catch` method is similar to `then`, except that it only expects a failure handler and will pass through the result unmodified in case of success. Much like with the `catch` clause for the `try` statement, control will continue as normal after the failure is caught. That way, the final `then`, which removes the loading message, is always executed, even if something went wrong.

You can think of the promise interface as implementing its own language for asynchronous control flow. The extra method calls and function expressions needed to achieve this make the code look somewhat awkward but not remotely as awkward as it would look if we took care of all the error handling ourselves.

Appreciating HTTP

When building a system that requires communication between a JavaScript program running in the browser (client-side) and a program on a server (server-side), there are several different ways to model this communication.

A commonly used model is that of *remote procedure calls*. In this model, communication follows the patterns of normal function calls, except that the function is actually running on another machine. Calling it involves making a request to the server that includes the function's name and arguments. The response to that request contains the returned value.

When thinking in terms of remote procedure calls, HTTP is just a

When a program is in the process of handling an action caused by some button or other control, which might require communication with the server and thus take a while, it can be a good idea to disable the control until the action finishes. That way, when the user gets impatient and clicks it again, they don't accidentally repeat their action.

The form as a whole

When a field is contained in a `<form>` element, its DOM element will have a property `form` linking back to the form's DOM element. The `<form>` element, in turn, has a property called `elements` that contains an array-like collection of the fields inside it.

The `name` attribute of a form field determines the way its value will be identified when the form is submitted. It can also be used as a property name when accessing the form's `elements` property, which acts both as an array-like object (accessible by number) and a map (accessible by name).

```
<form action="example/submit.html">
  Name: <input type="text" name="name"><br>
  Password: <input type="password" name="password"><br>
  <button type="submit">Log in</button>
</form>
<script>
  var form = document.querySelector("form");
  console.log(form.elements[1].type);
```

```
      reader.readAsText(file);
    });
  });
</script>
```

Reading a file is done by creating a `FileReader` object, registering a `"load"` event handler for it, and calling its `readAsText` method, giving it the file we want to read. Once loading finishes, the reader's `result` property contains the file's content.

The example uses `Array.prototype.forEach` to iterate over the array since in a normal loop it would be awkward to get the correct `file` and `reader` objects from the event handler. The variables would be shared by all iterations of the loop.

`FileReader`s also fire an `"error"` event when reading the file fails for any reason. The error object itself will end up in the reader's `error` property. If you don't want to remember the details of yet another inconsistent asynchronous interface, you could wrap it in a `Promise` (see Chapter 17) like this:

```
function readFile(file) {
  return new Promise(function(succeed, fail) {
    var reader = new FileReader();
    reader.addEventListener("load", function() {
      succeed(reader.result);
    });
    reader.addEventListener("error", function() {
      fail(reader.error);
    });
    reader.readAsText(file);
```

by selecting a tool from a `<select>` field and then clicking or dragging across the canvas. There are tools for drawing lines, erasing parts of the picture, adding text, and so on.

Clicking the canvas will hand off the `"mousedown"` event to the currently selected tool, which can handle it in whichever way it chooses. The line drawing tool, for example, will listen for `"mousemove"` events until the mouse button is released and draw lines along the mouse's path using the current color and brush size.

Color and brush size are selected with additional form fields. These are hooked up to update the canvas drawing context's `fillStyle`, `strokeStyle`, and `lineWidth` whenever they are changed.

You can load an image into the program in two ways. The first uses a file field, where the user can select a file on their own file system. The second asks for a URL and will fetch an image from the Web.

Images are saved in a somewhat atypical way. The save link at the right side points at the current image. It can be followed, shared, or saved. I will explain how this is achieved in a moment.

Building the DOM

Our program's interface is built from more than 30 DOM elements. We need to construct these somehow.

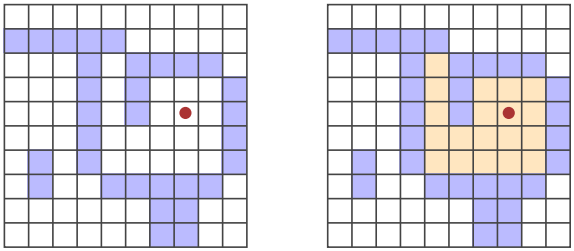
HTML is the most obvious format for defining complex DOM structures. But separating the program into a piece of HTML and a script is made difficult by the fact that many of the DOM elements need event handlers or have to be touched by the script in some other way. Thus,

some server.

Canvas elements have a convenient method, called `toDataURL`, which will return a data URL that contains the picture on the canvas as an image file. We don't want to update our save link every time the picture is changed, however. For big pictures, that involves moving quite a lot of data into a link and would be noticeably slow. Instead, we rig the link to update its `href` attribute whenever it is focused with the keyboard or the mouse is moved over it.

```
controls.save = function(cx) {
  var link = elt("a", {href: "/"}, "Save");
  function update() {
    try {
      link.href = cx.canvas.toDataURL();
    } catch (e) {
      if (e instanceof SecurityError)
        link.href = "javascript:alert(" +
          JSON.stringify("Can't save: " + e.toString()) + ")";
      else
        throw e;
    }
  }
  link.addEventListener("mouseover", update);
  link.addEventListener("focus", update);
  return link;
};
```

Thus, the link just quietly sits there, pointing at the wrong thing, but when the user approaches it, it magically updates itself to point at the



The flood fill does not leak through diagonal gaps and does not touch pixels that are not reachable, even if they have the same color as the target pixel.

You will once again need `getImageData` to find out the color for each pixel. It is probably a good idea to fetch the whole image in one go and then pick out pixel data from the resulting array. The pixels are organized in this array in a similar way to the grid elements in Chapter 7, one row at a time, except that each pixel is represented by four values. The first value for the pixel at (x,y) is at position $(x + y \times \text{width}) \times 4$.

Do include the fourth (alpha) value this time since we want to be able to tell the difference between empty and black pixels.

Finding all adjacent pixels with the same color requires you to "walk" over the pixel surface, one pixel up, down, left, or right, as long as new same-colored pixels can be found. But you won't find all pixels in a group on the first walk. Rather, you have to do something similar to the backtracking done by the regular expression matcher, described in Chapter 9. Whenever more than one possible direction to proceed is seen, you must store all the directions you do not take immediately and look at them later, when you finish your current walk.

In a normal-sized picture, there are a *lot* of pixels. Thus, you must

```
> var figlet = require("figlet");
> figlet.text("Hello world!", function(error, data) {
    if (error)
      console.error(error);
    else
      console.log(data);
  });
  _   _       _ _                                  _      _ _
 | | | | ___| | | ___   __      _____  _ __| | __| | |
 | |_| |/ _ \ | |/ _ \  \ \ /\ / / _ \| '__| |/ _` | |
 |  _  |  __/ | | (_) |  \ V  V / (_) | |  | | (_| |_|
 |_| |_|\___|_|_|\___/    \_/\_/ \___/|_|  |_|\__,_(_)
```

After running `npm install`, NPM will have created a directory called `node_modules`. Inside that directory will be a `figlet` directory, which contains the library. When we run `node` and call `require("figlet")`, this library is loaded, and we can call its `text` method to draw some big letters.

Somewhat unexpectedly perhaps, instead of simply returning the string that makes up the big letters, `figlet.text` takes a callback function that it passes its result to. It also passes the callback another argument, `error`, which will hold an error object when something goes wrong or null when everything is all right.

This is a common pattern in Node code. Rendering something with `figlet` requires the library to read a file that contains the letter shapes. Reading that file from disk is an asynchronous operation in Node, so `figlet.text` can't immediately return its result. Asynchronicity is infectious, in a way—every function that calls an asynchronous function must itself become asynchronous.

```
                              respond, request);
  else
    respond(405, "Method " + request.method +
            " not allowed.");
}).listen(8000);
```

This starts a server that just returns 405 error responses, which is the code used to indicate that a given method isn't handled by the server.

The `respond` function is passed to the functions that handle the various methods and acts as a callback to finish the request. It takes an HTTP status code, a body, and optionally a content type as arguments. If the value passed as the body is a readable stream, it will have a `pipe` method, which is used to forward a readable stream to a writable stream. If not, it is assumed to be either `null` (no body) or a string and is passed directly to the response's `end` method.

To get a path from the URL in the request, the `urlToPath` function uses Node's built-in `"url"` module to parse the URL. It takes its pathname, which will be something like `/file.txt`, decodes that to get rid of the `%20` -style escape codes, and prefixes a single dot to produce a path relative to the current directory.

```
function urlToPath(url) {
  var path = require("url").parse(url).pathname;
  return "." + decodeURIComponent(path);
}
```

If you are worried about the security of the `urlToPath` function, you are right. We will return to that in the exercises.

We will set up the `GET` method to return a list of files when reading a

Fixing a leak

For easy remote access to some files, I might get into the habit of having the file server defined in this chapter running on my machine, in the `/home/marijn/public` directory. Then, one day, I find that someone has gained access to all the passwords I stored in my browser.

What happened?

If it isn't clear to you yet, think back to the `urlToPath` function, defined like this:

```
function urlToPath(url) {
  var path = require("url").parse(url).pathname;
  return "." + decodeURIComponent(path);
}
```

Now consider the fact that paths passed to the `"fs"` functions can be relative—they may contain `"../"` to go up a directory. What happens when a client sends requests to URLs like the ones shown here?

```
http://myhostname:8000/../.config/config/google-chrome/Default/
    Web%20Data
http://myhostname:8000/../.ssh/id_dsa
http://myhostname:8000/../../../etc/passwd
```

Change `urlToPath` to fix this problem. Take into account the fact that Node on Windows allows both forward slashes and backslashes to separate directories.

Also, meditate on the fact that as soon as you expose some half-baked system on the Internet, the bugs in that system might be used to do bad things to your machine.

```
(time passes)

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 95

{"serverTime": 1405438913401,
 "talks": [{"title": "Unituning",
            "deleted": true}]}
```

When a talk has been changed, has been newly created, or has a comment added, the full representation of the talk is included in the response to the client's next polling request. When a talk is deleted, only its title and the property `deleted` are included. The client can then add talks with titles it has not seen before to its display, update talks that it was already showing, and remove those that were deleted.

The protocol described in this chapter does not do any access control. Everybody can comment, modify talks, and even delete them. Since the Internet is filled with hooligans, putting such a system online without further protection is likely to end in disaster.

A simple solution would be to put the system behind a *reverse proxy*, which is an HTTP server that accepts connections from outside the system and forwards them to HTTP servers that are running locally. Such a proxy can be configured to require a username and password, and you could make sure only the participants in the skill-sharing group have this password.

shortly, returns an array of changed talks since a given point in time. If it returns an empty array, the server does not yet have anything to send back to the client, so it stores the response object (using `waitForChanges`) to be responded to at a later time.

```
var waiting = [];

function waitForChanges(since, response) {
  var waiter = {since: since, response: response};
  waiting.push(waiter);
  setTimeout(function() {
    var found = waiting.indexOf(waiter);
    if (found > -1) {
      waiting.splice(found, 1);
      sendTalks([], response);
    }
  }, 90 * 1000);
}
```

The `splice` method is used to cut a piece out of an array. You give it an index and a number of elements, and it *mutates* the array, removing that many elements after the given index. In this case, we remove a single element, the object that tracks the waiting response, whose index we found by calling `indexOf`. If you pass additional arguments to `splice`, their values will be inserted into the array at the given position, replacing the removed elements.

When a response object is stored in the `waiting` array, a timeout is immediately set. After 90 seconds, this timeout sees whether the request is still waiting and, if it is, sends an empty response and removes

property.

This is a crude approach to templating, but it is enough to implement `drawTalk`.

```
function drawTalk(talk) {
  var node = instantiateTemplate("talk", talk);
  var comments = node.querySelector(".comments");
  talk.comments.forEach(function(comment) {
    comments.appendChild(
      instantiateTemplate("comment", comment));
  });

  node.querySelector("button.del").addEventListener(
    "click", deleteTalk.bind(null, talk.title));

  var form = node.querySelector("form");
  form.addEventListener("submit", function(event) {
    event.preventDefault();
    addComment(talk.title, form.elements.comment.value);
    form.reset();
  });
  return node;
}
```

After instantiating the `"talk"` template, there are various things that need to be patched up. First, the comments have to be filled in by repeatedly instantiating the `"comment"` template and appending the results to the node with class `"comments"`. Next, event handlers have to be attached to the button that deletes the task and the form that adds a new comment.

the sum of the two counters is even (`% 2`).

Terminating a line by adding a newline character happens after the line has been built up, so do this after the inner loop but inside of the outer loop.

Functions

Minimum

If you have trouble putting braces and parentheses in the right place to get a valid function definition, start by copying one of the examples in this chapter and modifying it.

A function may contain multiple `return` statements.

Recursion

Your function will likely look somewhat similar to the inner `find` function in the recursive `findSolution` example in this chapter, with an `if`/`else if`/`else` chain that tests which of the three cases applies. The final `else`, corresponding to the third case, makes the recursive call. Each of the branches should contain a `return` statement or in some other way arrange for a specific value to be returned.

When given a negative number, the function will recurse again and again, passing itself an ever more negative number, thus getting further and further away from returning a result. It will eventually run out of stack space and abort.

Regular Expressions

Quoting style

The most obvious solution is to only replace quotes with a nonword character on at least one side. Something like `/\W'|'\W/`. But you also have to take the start and end of the line into account.

In addition, you must ensure that the replacement also includes the characters that were matched by the `\W` pattern so that those are not dropped. This can be done by wrapping them in parentheses and including their groups in the replacement string (`$1`, `$2`). Groups that are not matched will be replaced by nothing.

Numbers again

First, do not forget the backslash in front of the dot.

Matching the optional sign in front of the number, as well as in front of the exponent, can be done with `[+\-]?` or `(\+|-|)` (plus, minus, or nothing).

The more complicated part of the exercise is the problem of matching both `"5."` and `".5"` without also matching `"."`. For this, a good solution is to use the `|` operator to separate the two cases—either one or more digits optionally followed by a dot and zero or more digits *or* a dot followed by one or more digits.

Finally, to make the *e* case-insensitive, either add an `i` option to the regular expression or use `[eE]`.

once, the result is a spiral.

The star (5) depicted is built out of `quadraticCurveTo` lines. You could also draw one with straight lines. Divide a circle into eight pieces, or a piece for each point you want your star to have. Draw lines between these points, making them curve toward the center of the star. With `quadraticCurveTo`, you can use the center as the control point.

The pie chart

You will need to call `fillText` and set the context's `textAlign` and `textBaseline` properties in such a way that the text ends up where you want it.

A sensible way to position the labels would be to put the text on the line going from the center of the pie through the middle of the slice. You don't want to put the text directly against the side of the pie but rather move the text out to the side of the pie by a given number of pixels.

The angle of this line is `currentAngle + 0.5 * sliceAngle`. The following code finds a position on this line, 120 pixels from the center:

```
var middleAngle = currentAngle + 0.5 * sliceAngle;
var textX = Math.cos(middleAngle) * 120 + centerX;
var textY = Math.sin(middleAngle) * 120 + centerY;
```

For `textBaseline`, the value `"middle"` is probably appropriate when using this approach. What to use for `textAlign` depends on the side of the circle we are on. On the left, it should be `"right"`, and on the right, it should be `"left"` so that the text is positioned away from the pie.

If you are not sure how to find out which side of the circle a given

Creating directories

You can use the function that implements the `DELETE` method as a blueprint for the `MKCOL` method. When no file is found, try to create a directory with `fs.mkdir`. When a directory exists at that path, you can return a 204 response so that directory creation requests are idempotent. If a nondirectory file exists here, return an error code. The code 400 ("bad request") would be appropriate here.

A public space on the web

You can create a `<textarea>` element to hold the content of the file that is being edited. A `GET` request, using `XMLHttpRequest`, can be used to get the current content of the file. You can use relative URLs like *index.html*, instead of *http://localhost:8000/index.html*, to refer to files on the same server as the running script.

Then, when the user clicks a button (you can use a `<form>` element and `"submit"` event or simply a `"click"` handler), make a `PUT` request to the same URL, with the content of the `<textarea>` as request body, to save the file.

You can then add a `<select>` element that contains all the files in the server's root directory by adding `<option>` elements containing the lines returned by a `GET` request to the URL `/`. When the user selects another file (a `"change"` event on the field), the script must fetch and display that file. Also make sure that when saving a file, you use the currently selected filename.

Unfortunately, the server is too simplistic to be able to reliably read

NORMAL PEOPLE

SALLY ROONEY

Author of CONVERSATIONS WITH FRIENDS

tying her scarf around her neck. She does a little knock on the door even though it’s already open.

Good to go? she says.

Yeah, says Connell.

Thanks for everything, Lorraine, says Marianne. See you next week.

Connell is already heading out the kitchen door when his mother says: You can say goodbye, can’t you? He turns to look over his shoulder but finds he cannot actually look Marianne in the eye, so he addresses himself to the floor instead. Right, bye, he says. He doesn’t wait to hear her reply.

In the car his mother puts on her seatbelt and shakes her head. You could be a bit nicer to her, she says. She doesn’t exactly have an easy time of it in school.

He puts the keys in the ignition, glances in the rear-view. I’m nice to her, he says.

She’s actually a very sensitive person, says Lorraine.

Can we talk about something else?

Lorraine makes a face. He stares out the windshield and pretends not to see.

was so wet, and she rolled her eyes back into her head and said: God, yes. And she was allowed to say it, no one would know. He was afraid he would come then just from touching her like that.

In the hallway the next morning he kissed her goodbye and her mouth tasted alkaline, like toothpaste. Thanks, she said. Then she left, before he understood what he was being thanked for. He put the bedsheets in the washing machine and took fresh linen from the hot press. He was thinking about what a secretive, independent-minded person Marianne was, that she could come over to his house and let him have sex with her, and she felt no need to tell anyone about it. She just let things happen, like nothing meant anything to her.

Lorraine got home that afternoon. Before she'd even put her keys on the table she said: Is that the washing machine? Connell nodded. She crouched down and looked through the round glass window into the drum, where his sheets were tossing around in the froth.

I'm not going to ask, she said.

What?

She started to fill the kettle, while he leaned against the countertop.

Why your bedclothes are in the wash, she said. I'm not asking.

He rolled his eyes just for something to do with his face. You think the worst of everything, he said.

She laughed, fixing the kettle into its cradle and hitting the switch. Excuse me, she said. I must be the most permissive mother of anyone in your school. As long as you're using protection, you can do what you want.

He said nothing. The kettle started to warm up and she took a clean mug down from the press.

Well? she said. Is that a yes?

Yes what? Obviously I didn't have unprotected sex with anyone while you were gone. Jesus.

So go on, what's her name?

He left the room then but he could hear his mother laughing as he went up the stairs. His life is always giving her amusement.

In school on Monday he had to avoid looking at Marianne or interacting with her in any way. He carried the secret around like something large and hot, like an overfull tray of hot drinks that he had to carry everywhere and never spill. She just acted the same as always, like it never happened,

not wanting to hang out with me anymore. I thought you liked me.

She shut her eyes. I do like you, she said.

Well, if you met someone else you liked more, I'd be pissed off, okay? Since you ask about it. I wouldn't be happy. Alright?

Your friend Eric called me flat-chested today in front of everyone.

Connell paused. She felt his breathing. I didn't hear that, he said.

You were in the bathroom or somewhere. He said I looked like an ironing board.

Fuck's sake, he's such a prick. Is that why you're in a bad mood?

She shrugged. Connell put his arms around her belly.

He's only trying to get on your nerves, he said. If he thought he had the slightest chance with you, he would be talking very differently. He just thinks you look down on him.

She shrugged again, chewing on her lower lip.

You have nothing to worry about with your appearance, Connell said.

Hm.

I don't just like you for your brains, trust me.

She laughed, feeling silly.

He rubbed her ear with his nose and added: I would miss you if you didn't want to see me anymore.

Would you miss sleeping with me? she said.

He touched his hand against her hipbone, rocking her back against his body, and said quietly: Yeah, a lot.

Can we go back to your house now?

He nodded. For a few seconds they just stood there in stillness, his arms around her, his breath on her ear. Most people go through their whole lives, Marianne thought, without ever really feeling that close with anyone.

*

Finally, after her third gin and tonic, the door bangs open and the boys arrive. The committee girls get up and start teasing them, scolding them for being late, things like that. Marianne hangs back, searching for Connell's eye contact, which he doesn't return. He's dressed in a white button-down shirt, the same Adidas sneakers he wears everywhere. The other boys are

Hello, sweetheart, said Lorraine.

Marianne’s face looked bright like a light bulb. Sorry to intrude, she said.

Connell didn’t touch her or speak to her. His chest hurt. She walked out the front door saying: Bye, sorry, thanks, sorry again. She shut the door behind her before he was even down the stairs.

Lorraine pressed her lips together like she was trying not to laugh. You can help me with the groceries, she said. She handed him one of the bags. He followed her into the kitchen and put the bag down on the table without looking at it. Rubbing his neck, he watched her unwrapping and putting away the items.

What’s so funny? he said.

There’s no need for her to run off like that just because I’m home, said Lorraine. I’m only delighted to see her, you know I’m very fond of Marianne.

He watched his mother fold away the reusable plastic bag.

Did you think I didn’t know? she said.

He closed his eyes for a few seconds and then opened them again. He shrugged.

Well, I knew someone was coming over here in the afternoons, said Lorraine. And I do work in her house, you know.

He nodded, unable to speak.

You must really like her, said Lorraine.

Why do you say that?

Isn’t that why you’re going to Trinity?

He put his face in his hands. Lorraine was laughing then, he could hear her. You’re making me not want to go there now, he said.

Oh, stop that.

He looked in the grocery bag he had left on the table and removed a packet of dried spaghetti. Self-consciously he brought it over to the press beside the fridge and put it with the other pasta.

So is Marianne your girlfriend, then? said Lorraine.

No.

her, of his own free choice, and he liked it. That says more about him, the supposedly ordinary and healthy person, than it does about her. She never went back to school again except to sit the exams. By then people were saying she had been in the mental hospital. None of that mattered now anyway.

*

Are you angry he did better than you? says her brother.

Marianne laughs. And why shouldn't she laugh? Her life here in Carricklea is over, and either a new life will begin, or it won't. Soon she will be packing things into suitcases: woollen jumpers, skirts, her two silk dresses. A set of teacups and saucers patterned with flowers. A hairdryer, a frying pan, four white cotton towels. A coffee pot. The objects of a new existence.

No, she says.

Why wouldn't you say hello to him, then?

Ask him. If you're such good friends with him, you should ask him. He knows.

Alan makes a fist with his left hand. It doesn't matter, it's over. Lately Marianne walks around Carricklea and thinks how beautiful it is in sunny weather, white clouds like chalk dust over the library, long avenues lined with trees. The arc of a tennis ball through blue air. Cars slowing at traffic lights with their windows rolled down, music bleating from the speakers. Marianne wonders what it would be like to belong here, to walk down the street greeting people and smiling. To feel that life was happening here, in this place, and not somewhere else far away.

What does that mean? says Alan.

Ask Connell Waldron why we're not speaking anymore. Call him back now if you want to, I'd be interested to hear what he has to say.

Alan bites down on the knuckle of his index finger. His arm is shaking. In just a few weeks' time Marianne will live with different people, and life will be different. But she herself will not be different. She'll be the same person, trapped inside her own body. There's nowhere she can go that would free her from this. A different place, different people, what does that matter? Alan releases his knuckle from his mouth.

Like he fucking cares, says Alan. I'm surprised he even knows your name.

Oh, we used to be quite close actually. You can ask him about that too,

You know me.

He stubbed out his cigarette and went back inside to collect his jacket. After that he left without saying goodbye to anyone, including Rachel, who broke up with him shortly afterwards. That was it, people moved away, he moved away. Their life in Carricklea, which they had imbued with such drama and significance, just ended like that with no conclusion, and it would never be picked back up again, never in the same way.

*

Yeah, well, he says to Marianne. I wasn't that compatible with Rachel, I don't think.

Marianne smiles now, a coy little smile. Hm, she says.

What?

I probably could have told you that.

Yeah, you should have, he says. You weren't really replying to my texts at the time.

Well, I felt somewhat abandoned.

I felt a bit abandoned myself, didn't I? says Connell. You disappeared. And I never had anything to do with Rachel until ages after that, by the way. Not that it matters now or anything, but I didn't.

Marianne sighs and moves her head from side to side, ambivalently.

That wasn't really why I left school, she says.

Right. I suppose you were better off out of it.

It was more of a last-straw thing.

Yeah, he says. I wondered if that was what it was.

She smiles again, a lopsided smile like she's flirting. Really? she says. Maybe you're telepathic.

I did used to think I could read your mind at times, Connell says.

In bed, you mean.

He takes a sip from his glass now. The beer is cold but the glass is room temperature. Before this evening he didn't know how Marianne would act if he ever met her in college, but now it seems inevitable, of course it would be like this. Of course she would talk drolly about their sex life, like it's a cute joke between them and not awkward. And in a way he likes it, he likes knowing how to act around her.

lit by the overhead lamp. It’s like a Diet Coke ad, said Marianne. Everyone laughed then, even Connell did. When it was just the black ball left he pointed at the top right-hand pocket and, gratifyingly, said: Alright, Marianne, are you watching? Then he potted it. Everyone applauded.

Instead of walking home that night, Connell came back to stay at hers. They lay in her bed looking up at the ceiling and talking. Until then they had always avoided discussing what had happened between them the year before, but that night Connell said: Do your friends know about us?

Marianne paused. What about us? she said eventually.

What happened in school and all that.

No, I don’t think so. Maybe they’ve picked up on something but I never told them.

For a few seconds Connell said nothing. She was attuned to his silence in the darkness.

Would you be embarrassed if they found out? he said.

In some ways, yeah.

He turned over then, so he wasn’t looking up at the ceiling anymore but facing her. Why? he said.

Because it was humiliating.

You mean like, the way I treated you.

Well, yeah, she said. And just the fact that I put up with it.

Carefully he felt for her hand under the quilt and she let him hold it. A shiver ran along her jaw and she tried to make her voice sound light and humorous.

Did you ever think about asking me to the Debs? she said. It’s such a stupid thing but I’m curious whether you thought about it.

To be honest, no. I wish I did.

She nodded. She continued looking up at the black ceiling, swallowing, worried that he could make out her expression.

Would you have said yes? he asked.

She nodded again. She tried to roll her eyes at herself but it felt ugly and self-pitying rather than funny.

I’m really sorry, he said. I did the wrong thing there. And you know, apparently people in school kind of knew about us anyway. I don’t know if

I don’t think they care very much what I do.

She covered her face using her flattened hands for a moment, and then she rubbed her nose briskly and sniffed. Connell knew she had a strained relationship with her family. He first came to realise this when they were still in school, and it didn’t strike him as unusual, because Marianne had strained relationships with everyone then. Her brother Alan was a few years older, and had what Lorraine called a ‘weak personality’. Honestly it was hard to imagine him standing his ground in a conflict with Marianne. But now they’re both grown up and still she almost never goes home, or she goes and then comes back like this, distracted and sullen, saying she had a fight with her family again, and not wanting to talk about it.

You had another falling-out with them, did you? Connell said.

She nodded. They don’t like me very much, she said.

I know it probably feels like they don’t, he said. But at the end of the day they’re your family, they love you.

Marianne said nothing. She didn’t nod or shake her head, she just sat there. Soon after that they went to bed. She was having cramps and she said it might hurt to have sex, so he just touched her until she came. Then she was in a good mood and making luxurious moaning noises and saying: God, that was so nice. He got out of bed and went to wash his hands in the en suite, a small pink-tiled room with a potted plant in the corner and little jars of face cream and perfume everywhere. Rinsing his hands under the tap, he asked Marianne if she was feeling better. And from bed she said: I feel wonderful, thank you. In the mirror he noticed he had a little blood on his lower lip. He must have brushed it with his hand by accident. He rubbed at it with the wet part of his knuckle, and from the other room Marianne said: Imagine how bitter I’m going to be when you meet someone else and fall in love. She often makes little jokes like this. He dried his hands and switched off the bathroom light.

I don’t know, he said. This is a pretty good arrangement, from my point of view.

Well, I do my best.

He got back into bed beside her and kissed her face. She had been sad before, after the film, but now she was happy. It was in Connell’s power to make her happy. It was something he could just give to her, like money or sex. With other people she seemed so independent and remote, but with Connell she was different, a different person. He was the only one who knew her like that.

interesting thought.

So what, are we not friends anymore? he says.

Of course we are.

You don’t reply to my messages very much.

Admittedly she has been ignoring him. She had to tell people what had happened between them, that he had broken up with her and moved away, and it mortified her. She was the one who had introduced Connell to everyone, who had told them all what great company he was, how sensitive and intelligent, and he had repaid her by staying in her apartment almost every night for three months, drinking the beer she bought for him, and then abruptly dumping her. It made her look like such a fool. Peggy laughed it off, of course, saying men were all the same. Joanna didn’t seem to think the situation was funny at all, but puzzling, and sad. She kept asking what each of them had specifically said during the break-up, and then she would go quiet, as if she was re-enacting the scene in her mind to try and make sense of it.

Joanna wanted to know if Connell knew about Marianne’s family. Everyone in Carricklea knows each other, Marianne said. Joanna shook her head and said: But I mean, does he know what they’re like? Marianne couldn’t answer that. She feels that even she doesn’t know what her family are like, that she’s never adequate in her attempts to describe them, that she oscillates between exaggerating their behaviour, which makes her feel guilty, or downplaying it, which also makes her feel guilty, but a different guilt, more inwardly directed. Joanna believes that she knows what Marianne’s family are like, but how can she, how can anyone, when Marianne herself doesn’t? Of course Connell can’t. He’s a well-adjusted person raised in a loving home. He just assumes the best of everyone and knows nothing.

I thought you would at least text me if you were coming home, he says. It’s kind of weird running into you when I didn’t know you were around.

At this moment she remembers leaving a flask in Connell’s car the day they drove to Howth in April, and she never got the flask back. It might still be in his glovebox. She eyes the glovebox but doesn’t feel she can open it, because he would ask what she was doing and she would have to bring up the trip to Howth. They went swimming in the sea that day and then parked his car somewhere out of sight and had sex in the back seat. It would be shameless to remind him of that day now that they’re once again in the car together, even though she would really like her flask back, or maybe it’s not about the flask, maybe she just wants to remind him he once

he said. That’s a sad thought if that’s true.

She started to kiss him then. This seemed like a strange thing to happen to him, unpleasant on the surface level, but also interesting in a way, as if his life was taking a new direction. Her mouth tasted sour like tequila. Briefly he wondered if it was legal for her to kiss him, and he concluded it must be, he couldn’t think of a reason why it wouldn’t be, and yet it felt substantially wrong. Every time he pulled away from her she seemed to follow him forward, so that he found himself puzzled about the physics of what was going on, and he was no longer sure whether he was sitting upright on the sofa or reclining backwards against the arm. As an experiment he tried to sit up, which confirmed he was in fact sitting up already, and the small red light which he thought might have been on the ceiling above him was just a standby light on the stereo system across the room.

Back in school Miss Neary had made him feel so uncomfortable. But was he mastering that discomfort now by letting her kiss him on the sofa in her living room, or just succumbing to it? He’d hardly had time to formulate this question when she started unbuttoning his jeans. In a panic he tried to push her hand away, but with such an ineffectual gesture that she appeared to think he was helping her. She got the top button undone and he told her that he was really drunk, and maybe they should stop. She put her hand inside the waistband of his underwear and said it was okay, she didn’t mind. He thought he would probably black out then, but he found he couldn’t. He wished he could have. He heard Paula saying: You’re so hard. That was an especially insane thing for her to say, because he actually wasn’t.

I’m going to get sick, he said.

She jerked back then, pulling her dress after her, and he took the opportunity to stand up from the sofa and button his jeans back up. Cautiously she asked if he was okay. When he looked at her he could make out two separate Paulas sitting on the couch, so clearly delineated that it was no longer obvious which was the real Paula and which the ghost. Sorry, he said. He woke up the next day fully clothed on the floor of his living room. He still has no idea how he made it home.

*

He must be insecure about something, says Marianne now. I don’t know what. Maybe he’d like to be more cerebral.

Maybe he just has good self-esteem.

environment. It’s not like a workplace.

Well, I doubt anyone in the workplace will spit at me over a disagreement, said Marianne. It would be pretty frowned upon, as I understand.

Denise gave a tight-lipped smile. If you can’t handle a little sibling rivalry, I don’t know how you’re going to manage adult life, darling, she said.

Let’s see how it goes.

At this, Denise struck the kitchen table with her open palm. Marianne flinched, but didn’t look up, didn’t let go of the envelope.

You think you’re special, do you? said Denise.

Marianne let her eyes close. No, she said. I don’t.

*

It’s almost one in the morning by the time Connell rings the buzzer. Marianne goes downstairs with her purse and finds the taxi is idling outside the building. In the square opposite, a mist wreathes itself around the trees. Winter nights are so exquisite, she thinks of saying to Connell. He’s standing talking to the driver through the window, with his back turned. When he hears the door he turns around, and she sees his mouth cut and bloody, dark blood like dried ink. She steps back, clutching her collarbone, and Connell goes: I know, I saw myself in the mirror. But I’m okay actually, I just need to get cleaned up. In a state of confusion she pays the driver, almost dropping her change in the gutter. On the staircase inside she sees Connell’s upper lip is swollen into a hard shiny mass on the right side. His teeth are the colour of blood. Oh god, she says. What happened? He takes her hand kindly, stroking her knuckles with his thumb.

Some guy came up and asked me for my wallet, he says. And I told him no, for some reason, and then he hit me in the face. I mean, it was a bad idea, I should have just given him the money. Sorry for calling you, it’s the only number I knew off the top of my head.

Oh, Connell, how awful. I have friends round, but what suits you? Do you want to have a shower or something and you can stay here? Or do you want to just get some cash and go home?

They’re outside the door of her apartment now, and they pause there.

Whatever’s good for you, he says. I’m really drunk, by the way. Sorry.

Oh, how drunk?

He and Niall and Elaine have arranged to get the train from Vienna to Trieste to spend their last few nights in Marianne's holiday home, before they all fly back to Dublin together. A day trip to Venice has been mentioned. Last night they got on the train with their backpacks and Connell texted Marianne: should be there by tomorrow afternoon, won't have time to reply to your email properly before then. He has almost no clean clothes left by now. He's wearing a grey T-shirt, black jeans and dirty white trainers. In his backpack: various lightly soiled clothes, one clean white T-shirt, an empty plastic bottle for water, clean underwear, a rolled-up phone charger, his passport, two packets of generic paracetamol, a very beaten-up copy of a James Salter novel, and for Marianne, an edition of Frank O'Hara's selected poems he found in an English-language bookshop in Berlin. One soft-covered grey notebook.

Elaine nudges Niall until his head jerks forward and his eyes open. He asks what time it is and where they are, and Elaine tells him. Then Niall links his fingers together and stretches his arms out in front of him. His joints crack quietly. Connell looks out the window at the passing landscape: dry yellows and greens, the orange slant of a tiled roof, a window cut flat by the sun and flashing.

*

The university scholarships were announced back in April. The Provost stood on the steps of the Exam Hall and read out a list of the scholars. The sky was extremely blue that day, delirious, like flavoured ice. Connell was wearing his jacket and Helen had her arm wrapped around his. When it came to English they read out four names, alphabetically, and the last one was: Connell Waldron. Helen threw her arms around him. That was it, they said his name and moved on. He waited in the square until they announced History and Politics, and when he heard Marianne's name he looked around to see her. He could hear a circle of her friends cheering, and some applause. He put his hands in his pockets. Hearing Marianne's name he realised how real it was, he really had won the scholarship, they both had. He doesn't remember much of what happened then. He remembers calling Lorraine after the announcements and she was just quiet on the phone, shocked, and then she murmured: Oh my god, Jesus Christ.

Niall and Elaine arrived beside him, cheering and slapping his back and calling him 'an absolute fucking nerd'. Connell was laughing at nothing, just because so much excitement demanded some kind of outward expression and he didn't want to cry. That night all the new scholars had to go to a formal black-tie meal together in the Dining Hall. Connell borrowed a tux from someone in his class, it didn't fit very well, and at dinner he felt

looked hungover, which pleased him, because the ceremony was so formal and they had to wear gowns and recite things in Latin. Afterwards they went for breakfast together in a cafe near college. They sat outside, at a table on the street, and people walked by carrying paper shopping bags and having loud conversations on the phone. Marianne drank a single cup of black coffee and ordered a croissant which she didn't finish. Connell had a large ham-and-cheese omelette with two slices of buttered toast, and tea with milk in it.

Marianne said she was worried about Peggy, who was the only one of the three of them not to get the scholarship. She said it would be hard on her. Connell inhaled and said nothing. Peggy didn't need subsidised tuition or free on-campus accommodation, because she lived at home in Blackrock and her parents were both doctors, but Marianne was intent on seeing the scholarships as a matter of personal feeling rather than economic fact.

Anyway, I'm happy for you, Marianne said.

I'm happy for you too.

But you deserve it more.

He looked up at her. He wiped his mouth with the napkin. You mean in terms of the financial stuff? he said.

Oh, she replied. Well, I meant that you're a better student.

She looked down critically at her croissant. He watched her.

Though in terms of financial circumstances too, obviously, she said. I mean, it's kind of ridiculous they don't means-test these things.

I guess we're from very different backgrounds, class-wise.

I don't think about it much, she said. Quickly she added: Sorry, that's an ignorant thing to say. Maybe I should think about it more.

You don't consider me your working-class friend?

She gave a smile that was more like a grimace and said: I'm conscious of the fact that we got to know each other because your mother works for my family. I also don't think my mother is a good employer, I don't think she pays Lorraine very well.

No, she pays her fuck all.

He cut a thin slice of omelette with his knife. The egg was more rubbery than he would have liked.

I'm surprised this hasn't come up before, she said. I think it's totally fair

and dissolve there. A cold flake alights on her top lip and she feels for it with her tongue. Head down against the cold, she is on her way to Lukas's studio. Lukas's hair is so blonde that the individual strands look white. She finds them on her clothing sometimes, finer than thread. He dresses all in black: black shirts, black zip-up hoodies, black boots with thick black rubber soles. He's an artist. The first time they met, Marianne told him she was a writer. It was a lie. Now she avoids talking to him about it.

Lukas lives near the station. She takes her hand from her pocket, blows on her fingers and presses the buzzer. He answers, in English: Who is it?

It's Marianne, she says.

Ah, you're early, says Lukas. Come on in.

Why does he say 'you're early'? Marianne thinks as she climbs the stairs. The connection was fuzzy but he seemed to say it with a smile. Was he pointing it out to make her appear too eager? But she finds she doesn't care how eager she appears, because there is no secret eagerness to be discovered in her. She could be here, ascending the staircase to Lukas's studio, or she could be in the campus library, or in the dorm making herself coffee. For weeks now she has had this feeling, the feeling of moving around inside a protective film, floating like mercury. The outside world touches against her outside skin, but not the other part of herself, inside. So whatever Lukas's reason for saying 'you're early', she finds it doesn't matter to her.

Upstairs he's setting up. Marianne removes her hat and shakes it. Lukas looks up, then back at the tripod. Are you getting used to the weather? he says. She hangs her hat on the back of the door and shrugs. She begins to take off her coat. In Sweden we have a saying, he says. There's no such thing as bad weather, only bad clothes.

Marianne hangs her coat beside her hat. What's wrong with my clothes? she says mildly.

It's just an expression, says Lukas.

She honestly can't tell now if he meant to criticise her clothes or not. She's wearing a grey lambswool sweater and a thick black skirt with knee-high boots. Lukas has bad manners, which, to Marianne, makes him seem childish. He never offers her coffee or tea when she arrives, or even a glass of water. He starts talking right away about whatever he has been reading or doing since her last visit. He doesn't seem to crave her input, and sometimes her responses confuse or disorientate him, which he claims is an effect of his bad English. In fact his comprehension is very good. Anyway,

good. He circles statement 1.

After completing the rest of the questions, all of which are intensely personal and the last one is about his sex life, he folds the pages over and hands them back to the receptionist. He doesn't know what to expect, handing over this extremely sensitive information to a stranger. He swallows and his throat is so tight it hurts. The woman takes the sheets like he's handing over a delayed college assignment and gives him a bland, cheerful smile. Thanks, she says. You can wait for the counsellor to call you now. He stands there limply. In her hand she holds the most deeply private information he has ever shared with anyone. Seeing her nonchalance, he experiences an impulse to ask for it back, as if he must have misunderstood the nature of this exchange, and maybe he should fill it out differently after all. Instead he says: Okay. He sits down again.

For a while nothing happens. His stomach is making a low whining noise now because he hasn't eaten breakfast. Lately he's too tired to cook for himself in the evenings, so he finds himself signing in for dinner on the scholars' website and eating Commons in the Dining Hall. Before the meal everyone stands for grace, which is recited in Latin. Then the food is served by other students, who are dressed all in black to differentiate them from the otherwise identical students who are being served. The meals are always the same: salty orange soup to start, with a bread roll and a square of butter wrapped in foil. Then a piece of meat in gravy, with silver dishes of potatoes passed around. Then dessert, some kind of wet sugary cake, or the fruit salad which is mostly grapes. These are all served rapidly and whisked away rapidly, while portraits of men from different centuries glare down from the walls in expensive regalia. Eating alone like this, overhearing the conversations of others but unable to join in, Connell feels profoundly and almost unendurably alienated from his own body. After the meal another grace is recited, with the ugly noise of chairs pulled back from tables. By seven he has emerged into the darkness of Front Square, and the lamps have been lit.

A middle-aged woman comes out to the waiting room now, wearing a long grey cardigan, and says: Connell? He tries to contort his face into a smile, and then, giving up, rubs his jaw with his hand instead, nodding. My name is Yvonne, she says. Would you like to come with me? He rises from the couch and follows her into a small office. She closes the door behind them. On one side of the office is a desk with an ancient Microsoft computer humming audibly; on the other side, two low mint-coloured armchairs facing one another. Now then, Connell, she says. You can sit down wherever you like. He sits on the chair facing the window, out of

mean everyone here just goes around comparing how much money their parents make. Like I'm being literal with that, I've seen that happen.

He breathes in now, feeling that he has been talking too quickly and at too great a length, but unwilling to stop.

I just feel like I left Carricklea thinking I could have a different life, he says. But I hate it here, and now I can never go back there again. I mean, those friendships are gone. Rob is gone, I can never see him again. I can never get that life back.

Yvonne pushes the box of tissues on the table towards him. He looks at the box, patterned with green palm leaves, and then at Yvonne. He touches his own face, only to discover that he has started crying. Wordlessly he removes a tissue from the box and wipes his face.

Sorry, he says.

Yvonne is making eye contact now, but he can't tell anymore whether she's been listening to him, whether she's understood or tried to understand what he's said.

What we can do here in counselling is try to work on your feelings, and your thoughts and behaviours, she says. We can't change your circumstances, but we can change how you respond to your circumstances. Do you see what I mean?

Yeah.

At this point in the session Yvonne starts to hand him worksheets, illustrated with large cartoon arrows pointing to various text boxes. He takes them and pretends that he's intending to fill them out later. She also hands him some photocopied pages about dealing with anxiety, which he pretends he will read. She prints a note for him to take to the college health service advising them about his depression, and he says he'll come back for another session in two weeks. Then he leaves the office.

*

A couple of weeks ago Connell attended a reading by a writer who was visiting the college. He sat at the back of the lecture hall on his own, self-conscious because the reading was sparsely attended and everyone else was sitting in groups. It was one of the big windowless halls in the Arts Block, with fold-out tables attached to the seats. One of his lecturers gave a short and sycophantic overview of the writer's work, and then the man himself, a youngish guy around thirty, stood at the lectern and thanked the college for the invitation. By then Connell regretted his decision to attend. Everything

Connell nods, frowning. Yeah, he says. I know what you mean.

You weren’t lonely with Helen, were you?

I don’t know. Sometimes. I didn’t feel totally myself with her all the time.

Marianne lies down flat on her back now, head on the pillow, bare legs stretched on the duvet. She stares up at the light fixture, the same lampshade from years ago, dusty green.

Connell, she says. You know when we were dancing last night?

Yeah.

For a moment she just wants to lie here prolonging the intense silence and staring at the lampshade, enjoying the sensory quality of being here in this room again with him and making him talk to her, but time moves on.

What about it? he says.

Did I do something to annoy you?

No. What do you mean by that?

When you walked off and just left me there, she says. I felt kind of awkward. I thought maybe you were gone after that girl Niamh or something, that’s why I asked about her. I don’t know.

I didn’t walk off. I asked you if you wanted to go out to the smoking area and you said no.

She sits up on her elbows and looks at him. He’s flushed now, his ears are red.

You didn’t ask, she says. You said, I’m going out to the smoking area, and then you walked away.

No, I said do you want to come out to the smoking area, and you shook your head.

Maybe I didn’t hear you right.

You must not have, he says. I definitely remember saying it to you. But the music was very loud, to be fair.

They lapse into another silence. Marianne lies back down, looks up at the light again, feels her own face glowing.

I thought you were annoyed with me, she says.

Well, sorry. I wasn’t.

What does that mean, you think I'm lying? There's nothing going on there, trust me.

For a few seconds Lorraine says nothing. He swallows some beer and puts the can down on the table. It is extremely irritating that his mother thinks he and Marianne are together, when the closest they have come in years to actually being together was earlier this evening, and it ended with him crying alone in his room.

You're just coming home every weekend to see your beloved mother, then, are you? she says.

He shrugs. If you don't want me to come home, I won't, he says.

Oh, come on now.

She gets up to fill the kettle. He watches her idly while she tamps her teabag down into her favourite cup, then he rubs at his eyes again. He feels like he has ruined the life of everyone who has ever even marginally liked him.

*

In April, Connell sent one of his short stories, the only really completed one, to Sadie Darcy-O'Shea. She emailed back within an hour:

Connell it's incredible! let us publish it please! xxx

When he read this message his pulse hammered all over his body, loud and hard like a machine. He had to lie down and stare at the white ceiling. Sadie was the editor of the college literary journal. Finally he sat up and wrote back:

I'm glad you liked it but I don't think it's good enough to be published yet, thanks though.

Instantly Sadie replied:

PLEASE? XXX

Connell's entire body was pounding like a conveyor belt. No one had ever read a word of his work before that moment. It was a wild new landscape of experience. He paced around the room massaging his neck for a while. Then he typed back:

Ok, how about this, you can publish it under a pseudonym. But you also have to promise you won't tell anyone who wrote it, even the other people who edit the magazine. Ok?

mind. But did he love her? Sometimes she felt like saying: Would you miss me, if you didn't have me anymore? She had asked him that once on the ghost estate, when they were just kids. He had said yes then, but she'd been the only thing in his life at that time, the only thing he had to himself, and it would never be that way again.

By the start of December their friends were asking about Christmas plans. Marianne still hadn't seen her family since the summer. Her mother had never tried to contact her at all. Alan had sent some text messages saying things like: Mum is not speaking to you, she says you are a disgrace. Marianne hadn't replied. She'd rehearsed in her head what kind of conversation it would be when her mother did finally get in touch, what accusations would be made, which truths she would insist on. But it never happened. Her birthday came and went without a word from home. Then it was December and she was planning to stay in college alone for Christmas and get some work done on the dissertation she was writing on Irish carceral institutions after independence. Connell wanted her to come back to Carricklea with him. Lorraine would love to have you, he said. I'll call her, you should talk to her about it. In the end Lorraine called Marianne herself and personally invited her to stay for Christmas. Marianne, trusting that Lorraine knew what was right, accepted.

On the way home from Dublin in the car, she and Connell talked without stopping, joking and putting on funny voices to make each other laugh. Looking back now, Marianne wonders if they were nervous. When they got to Foxfield it was dark and the windows were full of coloured lights. Connell carried their bags in from the boot. In the living room Marianne sat by the fire while Lorraine made tea. The tree, packed between the television and couch, was blinking light in repetitive patterns. Connell came in carrying a cup of tea and put it on the arm of her chair. Before sitting down he stopped to rearrange a piece of tinsel. It did look much better where he put it. Marianne's face and hands were very hot by the fire. Lorraine came in and started telling Connell about which relatives had visited already, and which were visiting tomorrow, and so on. Marianne felt so relaxed then that she almost wanted to close her eyes and sleep.

The house in Foxfield was busy over Christmas. Late into the night people would be arriving and leaving, brandishing wrapped biscuit tins or bottles of whiskey. Children ran past at knee height yelling unintelligibly. Someone brought a Play-Station over one night and Connell stayed up until two in the morning playing FIFA with one of his younger cousins, their bodies greenish in the screen light, a look of almost religious intensity on Connell's face. Marianne and Lorraine were in the kitchen mostly, rinsing

Without turning around, Marianne walks out the door, lets it slam behind her. She's in the hallway now with the cloakroom and can't remember whether the exit is right or left. She's shaking all over her body. The cloakroom attendant asks if she's alright. Marianne doesn't know anymore how drunk she is. She walks a few steps towards a door on the left and then puts her back against the wall and starts sliding down towards a seated position on the floor. Her breast is aching where that man grabbed it. He wasn't joking, he wanted to hurt her. She's on the floor now hugging her knees against her chest.

Up the hall the door comes open again and Karen comes out, with Eric and Rachel and Connell following. They see Marianne on the floor and Karen runs over to her while the other three stay standing where they are, not knowing what to do maybe, or not wanting to do anything. Karen hunches down in front of Marianne and touches her hand. Marianne's eyes are sore and she doesn't know where to look.

Are you alright? Karen says.

I'm fine, says Marianne. I'm sorry. I think I just had too much to drink.

Leave her, says Rachel.

Here, look, it was just a bit of fun, says Eric. Pat's actually a sound enough guy if you get to know him.

I think it was funny, says Rachel.

At this Karen snaps around and looks at them. Why are you even out here if you think it was so funny? she says. Why don't you go and pal around with your best friend Pat? If you think it's so funny to molest young girls?

How is Marianne *young*? says Eric.

We were all laughing at the time, says Rachel.

That's not true, says Connell.

Everyone looks around at him then. Marianne looks at him. Their eyes meet.

Are you okay, are you? he says.

Oh, do you want to kiss her better? says Rachel.

His face is flushed now, and he touches a hand to his brow. Everyone is still watching him. The wall feels cold against Marianne's back.

Rachel, he says, would you ever fuck off?

Goodnight, then.

She closes the door behind her. He listens to her footsteps up the stairs. After a few minutes have passed he gets up, empties the dregs of his beer down the sink and puts the can quietly in the recycling bin.

On the table his phone starts ringing. It's set to vibrate so it starts shimmying around the surface of the table, catching the light. He goes to get it before it falls over the edge, and he sees it's Marianne calling. He pauses. He looks at the screen. Finally he slides the answer button.

Hey, he says.

He can hear her breath hard on the other end of the line. He asks if she's okay.

I'm really sorry about this, she says. I feel like an idiot.

Her voice in the phone sounds clouded, like she has a bad cold, or something in her mouth. Connell swallows and walks over to the kitchen window.

About earlier? he says. I've been thinking about it as well.

No, it's not that. It's really stupid. I just tripped or something and I have a small injury. I'm sorry to bother you about it. It's nothing. I just don't know what to do.

He puts his hand on the sink.

Where are you? he says.

I'm at home. It's not serious, it just hurts, that's all. I don't really know why I'm calling. I'm sorry.

Can I come get you?

She pauses. In a muffled voice she replies: Yes, please.

I'm on my way, he says. I'm getting in the car right now, okay?

Sandwiching the phone between his ear and shoulder, he fishes his left shoe out from under the table and pulls it on.

This is really nice of you, says Marianne in his ear.

I'll see you in a few minutes. I'm leaving right now. Alright? See you soon.

Outside he gets in the car and starts the engine. The radio comes on and he snaps it off with a flat hand. His breath isn't right. After only one drink he feels out of it, not alert enough, or too alert, twitchy. The car is too silent