Data-Driven Programs

Eric Roberts
CS 106AJ
November 17, 2017
Once upon a time . . .
Computing and the Counterculture

Two recent books argue that the personal computing revolution owes as much to the counterculture of the 1960s as it does to the technological strength and entrepreneurial spirit of Silicon Valley.
Ted Nelson’s Cyberspace Dreams

The countercultural vision comes across particularly clearly in the two-sided book *Computer Lib/Dream Machines* which was written by cyberspace visionary Ted Nelson in 1974.
Data-Driven Programs
Designing Data Structures

• When you design a program, one of the first tasks you need to undertake is understanding how the underlying data structures fit together and how each level of the data hierarchy can best be represented.

• This process is similar to that of decomposing a large problem into a set of successively simpler subproblems. In the data domain, the information your program needs to process must be decomposed into successively simpler data structures until everything can be represented using a built-in JavaScript value, such as a number or a string.

• The tools for data decomposition you have seen so far include
  - *Arrays*, which implement sequences of values
  - *Aggregates*, which represent collections of related values
  - *Maps*, which establish a relationship between keys and values
Exercise: Data Structures

• What kind of data structures would you need to keep track of Stanford students and their courses?
  – What data values would you need to store for students?
  – What data values would you need to store for courses?
  – What structures (arrays, aggregates, and maps) make sense if you are trying to represent the entire structure?
Data-Driven Programs

• In most programming languages, data structures are easier to manipulate than code. As a result, it is often useful to design applications so that as much of their behavior as possible is represented as data rather than in the form of methods. Programs that work this way are said to be *data driven*.

• In a data-driven system, the actual program (which is called a *driver*) is usually very small. Such driver programs operate in two phases:

  1. Read data from a file into a suitable internal data structure.
  2. Use the data structure to control the flow of the program.

• To illustrate the idea of a data-driven system, we’re going to spend most of this lecture building a programmed-instruction “teaching machine” of the sort that Ted Nelson discusses (mostly critically) in *Dream Machines*.
A New Approach to Files

• The teaching machine program in the textbook reads the data for the course from a file chosen by the user.

• Although the file-based model is standard practice in most programming languages, it doesn’t make sense for CS 106AJ. Security considerations make file reading too cumbersome to use in JavaScript.

• Earlier this quarter, it occurred to me that JavaScript offers an elegant solution to the problem. All you need to do is store the necessary data in the index.html file, which JavaScript programs can always read.

• Jerry and I have decided to use this strategy for both the teaching machine example and the Adventure assignment, which should make your lives much easier after the Thanksgiving holidays.
XML and the DOM

• The `index.html` file is written in the *Hypertext Markup Language* (HTML), which is a subset of the *Extensible Markup Language* (XML). In recent years, XML has become the industry standard for representing hierarchical data.

• Modern browsers parse all the XML data in the `index.html` file and store that information in an internal form called the *Document Object Model* (DOM).

• Much of JavaScript’s undeserved negative reputation comes from the fact that the DOM is a complete mess.

• In general, the best strategy is to use as little of the DOM as possible. The three methods on the next page are all you need.

• To keep the user data from appearing on the page, you need to embed its XML in a `<div style="display:none;">` block.
## Three Handy DOM Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>document.getElementById(id)</code></td>
<td>Returns the element with the specified id attribute.</td>
</tr>
<tr>
<td><code>element.getElementsByTagName(name)</code></td>
<td>Returns an array of the elements with the specified tag name.</td>
</tr>
<tr>
<td><code>element.getAttribute(name)</code></td>
<td>Returns the value of the named attribute.</td>
</tr>
</tbody>
</table>

---
Choosing an Internal Representation

The first step in building the teaching machine is to design a set of classes that can represent the data and relationships in the file. All the relevant data should be accessible from a single structure that stores the information in a nested series of classes.
<course id="CourseData" title="JavaScript">
  <question name="Q1">
    True or false: Numbers can have fractional parts.
    <answer response="true" nextQuestion="Q3" />
    <answer response="false" nextQuestion="Q2" />
  </question>
  <question name="Q2">
    That's incorrect. True or false: Numbers can be negative.
    <answer response="true" nextQuestion="Q3" />
    <answer response="false" nextQuestion="Q1" />
  </question>
  <question name="Q3">
    You seem to have mastered JavaScript. Start over?
    <answer response="yes" nextQuestion="Q1" />
    <answer response="no" nextQuestion="EXIT" />
  </question>
</course>
/*
 * File: TeachingMachine.js
 * ------------------------
 * This program executes a programmed instruction course.
 */

function TeachingMachine() {
    let course = TMCourse();
    if (course === undefined) {
        console.log("No course is defined in the HTML file");
    } else {
        course.run();
    }
}

Code for the TeachingMachine Class
/*
 * File: TMCourse.js
 * ----------------
 * This class defines the data structure for a course for use with
 * the TeachingMachine program.
 */

/*
 * Creates a TMCourse object by reading data from the index.html file.
 */

function TMCourse() {
    let element = document.getElementById("CourseData");
    if (element === undefined) return undefined;
    let questions = readQuestions(element);
    let course = { }
    course.run = function() { . . . }
    return course;
}
The `TMCourse.run` Method

```javascript
const MARKER = "-----

// Creates a new course for the teaching machine by reading the
data from the specified file, which consists of questions and
their accepted answers.

function TMCourse(filename) {
    let currentQuestion = questions["START"];  
    function askQuestion() {
        currentQuestion.printQuestionText();  
        console.requestInput("> ", checkAnswer);
    }
    function checkAnswer(line) {
        let nextQuestion = currentQuestion.getNextQuestion(line);
        if (nextQuestion === "EXIT") return;
        if (nextQuestion === undefined) {
            console.log("I don't understand that response.");
        } else {
            currentQuestion = questions[nextQuestion];
        }
        askQuestion();
    }

    askQuestion();
}
```

function TMQuestion(text, answers) {
  let question = { };

  question.printQuestionText = function() {
    for (let i = 0; i < text.length; i++) {
      console.log(text[i]);
    }
  };

  question.getNextQuestion = function(answer) {
    return answers[answer.toLowerCase()];
  }

  question.toString = function() {
    return "Question: " + text[0];
  }

  return question;
}
The readQuestions Function

/*
 * Creates the questions data structure by reading the elements
 * with the tag "question" from the XML for the course. To ensure
 * that the course starts with the first question in the file, the
 * map stores a reference to that question under the key "START".
 */

function readQuestions(courseXML) {
    let tags = courseXML.getElementsByTagName("question");
    let questions = { }
    for (let i = 0; i < tags.length; i++) {
        let questionXML = tags[i];
        let name = questionXML.getAttribute("name");
        let text = readQuestionText(questionXML);
        let answers = readAnswers(questionXML);
        let question = TMQuestion(text, answers);
        if (i === 0) questions["START"] = question;
        questions[name] = question;
    }
    return questions;
}
The **readQuestionText** Function

```javascript
function readQuestionText(questionXML) {
    let name = questionXML.getAttribute("name");
    let text = [ ];
    let lines = questionXML.innerHTML.split("\n");
    if (lines[0] === ") lines.shift();
    for (let i = 0; i < lines.length; i++) {
        let line = lines[i].trim();
        if (!line.startsWith("<")) {
            text.push(line);
        }
    }
    return text;
}
```
/ * Reads the data structure containing the answers, which maps * user responses to the name of the next question. */ 

function readAnswers(questionXML) {
    let answers = { };
    let elements = questionXML.getElementsByTagName("answer");
    for (let i = 0; i < elements.length; i++) {
        let answerXML = elements[i];
        let response = answerXML.getAttribute("response");
        let nextQuestion = answerXML.getAttribute("nextQuestion");
        answers[response.toLowerCase()] = nextQuestion;
    }
    return answers;
}
The End