Chapman & Hall/CRC
Textbooks in Computing

Start Programming using HTML, CSS, and JavaScript

Iztok Fajfar

CRC Press
Taylor & Francis Group
A Chapman & Hall Book

START PROGRAMMING USING HTML, CSS, AND JAVASCRIPT

CHAPMAN & HALL/CRC
TEXTBOOKS IN COMPUTING

START PROGRAMMING USING HTML, CSS, AND JAVASCRIPT

Iztok Fajfar

University of Ljubljana
Slovenia

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20150904

International Standard Book Number-13: 978-1-4987-3145-4 (eBook - PDF)

To my family

Contents

Acknowledgments

A huge thank you goes to the guys at Taylor and Francis, especially to my editor Randi Cohen for her enthusiasm for the whole project, my project coordinator Ashley Weinstein, who oversaw production attentively, and technical reviewers for their detailed comments making the whole book more enjoyable. Many thanks also to the proofreader for correcting typos and grammar. Indeed, it was a great pleasure to work with such a professional team.

Honestly, all this wouldn't have happened were it not for Igor and the other guys from the morning-coffee crew, who suggested that I should really write a book. Thanks, chaps, it cost me a year of my life. Thank you to all my amazing students for sitting through my programming lectures and asking nasty questions. Man, how should I know all that? I shall not forget to also thank the other teaching staff from the team. The joy of working together is immeasurable. I'm deeply indebted to Žiga, who had painstakingly read the whole manuscript before releasing it to the wild. (I sincerely hope you spotted all the silly mistakes so I don't make a fool of myself.) Thank you, Andrej, for technical advice on preparing the camera-ready PDF. Those are really details that make a difference. A thousand thanks go to Tanja and Tadej for that little push that did the trick. You are terrific!

I also wish to extend my considerable gratitude to everyone that gave away their precious time, energy, and invaluable expertise answering questions on forums, posting on blogs, and writing all those wonderful LaTeXpackages. It's impossible to list you all by name because I'm contracted for only 400 or so pages.

A colossal thank you goes out to my mom and dad for instantiating and personalizing me. It wasn't the easiest assignment in the world but you did a marvelous job! Many thanks to my second parents, Dana and Ivo, for telling me that I should also eat if I am ever to finish the book. A zillion thanks go out to my close family. Thank you, Erik, for patiently checking which page I am on with an I-want-my-daddy-back determination; and thank you, Monika, for tons of understanding and supportive coffee mugs. I love you!

I am also thankful for the support of the Ministry of Education, Science, and Sport of the Republic of Slovenia within the research program P2-0246—Algorithms and Optimization Methods in Telecommunications, which made possible some of the research for this book.

And, of course, thank *you*, the reader. Without you, this book wouldn't make much sense, would it?

—Iztok Fajfar (iztok.fajfar.eu), March 2015

Introduction

Easy to Use

Normally, putting honey in my tea is not a particularly demanding task, but that morning my hand was paralyzed in astonishment, trying to do its routine job of pouring some honey in the steaming cup. Honey labels usually say things like "All Natural," "Contains Antioxidants," or "With Grandma's Recipe Book." Over time, I've got used to more absurd labels like "Improved New Flavor" or "Gathered by Real Bees." The label that knocked me out was surprisingly plain, with an award-winning message printed on it: "Easy to Use." I don't recall honey ever being hard to use, except maybe when it crystallizes, or when I was six months old, but that's probably not exactly what the author of the message had in mind.

You can also buy programming books that promise easy and quick learning, even as fast as in 24 hours. An average adult can read a novel in 24 hours. But let's face it, no one can read—let alone understand and learn—a 500-page technical book in 24 hours. While using honey is not difficult even when it doesn't explicitly say so, learning to program is not easy. It can be fun if you're motivated and have decent material to study from, but it's also an effort. If you're not ready to accept that, then this book is not for you. Otherwise, I invite you to join Maria, Mike, and me at exploring the exciting world of computer programming. It's going to be fun but it's also going to be some work.

About the Book

This handbook is a manual for undergraduate students of engineering and natural science fields written in the form of a dialog between two students and a professor discovering how computer programming works. It is organized in 13 thematic meetings with explanations and discussions, supported by gradual evolution of engaging working examples of live web documents and applications using HTML, CSS, and JavaScript. You will see how the three mainstream languages interact, and learn some of the essential practices of using them to your advantage. At the end of each meeting there is a practical homework, which is always discussed at the beginning of the next meeting. There is also a list of related keywords to help you review important topics of each meeting.

The general structure of the book is multilayered: the basic language syntax and rules are fleshed out with contents and structure while still keeping things simple and manageable, something that many introductory textbooks lack.

The main body of the text is accompanied by five appendices. The first of them contains a solution of the last homework, the second summarizes (also with examples) some major directions in which you can continue your study, including hints on some of the relevant sources. The last three appendices are abbreviated references of the three languages used in the book.

There will be situations when you need to use yet more languages and technologies in order to get the job done. Some such situations are gently dealt with in this book. For example, you will learn just enough about a Server Side Includes language to be able to include external HTML code, which will save you a tremendous amount of time and energy.

Is This Book for Me?

If you know absolutely nothing about computer programming and want to learn, this is the book for you. It has been written with a complete beginner in mind in the first place.

If you have been exposed to programming before, you might find the book useful as well. Today, many people learn from examples and forums, and thus acquired knowledge is mostly skills and not much theory. If you ever want to build more serious software, you need a firm and systematic understanding of what is going on. You need a framework to which you can systematically attach your partial skills to form a sound structure of connected knowledge. Hopefully, this book can give you this as well.

Last but not least, if you're a teacher of an introductory programming course, you might find a handful of useful examples and approaches for your classes on the few hundred pages that follow.

But most likely, as there are as many learning styles as there are learners, you will have to find out for yourself whether or not this book is for you.

How to Avoid Reading the Whole Book

Don't panic! If you are only up to JavaScript programming, you can just read Meeting 1 to get a basic idea of what HTML is (you need this in order to be able to run the JavaScript examples in this book), and then you can immediately skip to Meeting 6—more specifically, Section 6.3. There are some examples involving CSS in the JavaScript part but they won't stand in the way of your learning JavaScript. Later, if you feel like it, you can just as well skim over Meeting 3, where you can get the basic idea of what CSS is all about.

For Your Safety

This book is not about cutting-edge web technologies, so you don't need any protective equipment. It is more about general computer programming and some web-related principles using the mainstream web languages HTML, CSS, and JavaScript as examples. Some of the principles are over 40 years old, but are extremely important because they allow you to write cleaner and more easily maintainable code, and they will not go away just like that.

It's a busy world, and the sixth edition of ECMAScript standard (the standardized version of JavaScript) has just entered the official publication process. The good news is that it only introduces additions to its predecessor, so the essential concepts stay. Also, while CSS3 isn't completely finished yet, there already exist some so-called "level 4" CSS modules. Fortunately, they are also just additions to the CSS standard and there are no serious plans for a single CSS4 specification on the horizon. This book pays attention to the basic concepts that have matured with the latest HTML5, CSS3, and ECMAScript 5 standards to the point where it seems these concepts are going to persist for some time.

The Software Used

In researching this book, I used Google Chrome and Notepad++ v6.5.3 (*notepad-plus-plus.org*) on a Windows 7 Professional SP1 64bit operating system. I also used the EasyPHP DevServer 13.1 VC11 web development server (*www.easyphp.org*). However, you will be able to follow most of the examples and experiments in this book using any modern browser and plain text editor. They are already installed on your computer, so you can start experimenting right away.

Conventions Used in This Book

The following typographical conventions are used in this book:

`A monospaced font` is used for all code listings and everything that you normally type on a keyboard, including keys and key combinations.

`A monospaced italic font` is used as a general placeholder to mark items that you should replace with an actual value or expression in your code.

An italic font is used to indicate the first appearance of a term, or as an emphasis.

A sans serif font is used to indicate a menu item.

A sans serif italic font is used to indicate URLs and file names and extensions.

Feedback and Supporting Online Material

I deeply appreciate having any comments, suggestions, or errors found brought to my attention at the email address *start-programming@fajfar.eu*. You will find source code of the examples in this book and some additional materials and problems for each chapter at *fajfar.eu/start-programming*.

About the Author

Iztok Fajfar got his first computer in the early 1980s, a ZX Spectrum with an amazing 48 KB of RAM. Computers soon turned into a lifelong fascination and an indispensable companion, assisting him in his professional work and hobbies alike. Iztok has a PhD degree in electrical engineering from the University of Ljubljana, Slovenia, where he is currently Associate Professor at the Faculty of Electrical Engineering. His research topics include evolutionary algorithms, in particular, genetic programming. He teaches computer programming at all levels, from assembly to object-oriented, and to all kinds of audience. Now and then he even ventures to explain to his mother-in-law how to forward an email, and he hasn't given up yet. He is also a programmer and writer. Iztok lives with his family in Ljubljana, and when he is not programming, or teaching, or researching weird stuff, he makes the most yummy pancakes, not to mention the pizza.

MEETING 1

Content and Structure

1.1 Opening

Professor: I'm thrilled that you accepted my invitation to help me with a new book I am researching. There are three languages awaiting us in this course: HTML, CSS, and JavaScript.

Mike: Why three? You'll just confuse us, won't you?

Professor: The languages have been designed for quite specific purposes and work very differently, so there is little danger in confusing them. At the same time, the three languages nicely complement each other: HTML holds the structure and content of a web page, CSS takes care of presentation, and JavaScript is responsible for action. I like to say that HTML is bones, CSS is flesh, and JavaScript is the brain and muscles of web programming.

Maria: How much of a chance is there of us learning three languages to the level that we can use any of them to our advantage?

Professor: You don't have to be a guru in any of them to start using them effectively. It's only important that you know the basic principles. The good news is you don't have to install or learn to use any new software. All you need to start off is already installed on your computer.

Do you have any programming experience?

Maria: Actually, I use a computer a lot but not for programming. I have never written a computer program before.

Mike: Neither have I.

Professor: In a way, programming is like speaking. You speak English, right?

Mike: Yes...?

Professor: I even know people who have learned Finnish. Quite well, to be honest.

English and Finnish are examples of *natural languages*, which people learn to communicate with other people. However, if you want to talk to computers, you have to learn *artificial languages* so that computers understand and *obey* you. It's very similar. The only difference is that people won't obey you if you lack charm, while computers won't obey you if you're not accurate. Accuracy is crucial. Similar to both is that it takes a certain amount of practice before your interlocutor understands you. I won't lie to you on this one.

Maria: I'm just starting to learn Spanish and I must use a sign language a lot. I suppose you cannot use a sign language with a computer.

Professor: That's true. In natural languages, people use context and even a sign language to guess what others have to say even though what they say may not be grammatically correct. Computers don't do that, though, and that's the difficult part of programming. You have to be *exact*.

All right. Let's start programming, shall we?

1.2 Introducing HTML

Professor: To be precise, HTML is not a programming language but it is a so-called *markup language*. That's what the acronym HTML stands for: Hypertext Markup Language. Markup is a modern approach for adding different annotations to a document in such a way that these annotations are distinguishable from plain text. Markup instructions tell the program that displays your text what actions to perform while the instructions themselves are hidden from the person that views your text. For example, if you want a certain part of your text to appear as a paragraph, you simply mark up this part of the text using appropriate *tags*:

```
<p>But it's my only line!</p>
```

Maria: It looks quite straightforward. Are those p's in the angle brackets like commands?

Professor: You could say that. They are called *tags* and they *instruct* or *command* a browser to make a paragraph out of the text between them.

Mike: That's like formatting, isn't it?

Professor: In a way, yes. Tags are like commands in a word processor that allow you to format paragraphs, headings, and so forth. However, they only specify *what* to format, not *how* to do it.

The above code fragment is an example of an HTML *element*—the basic building block of an HTML document. An HTML document is composed exclusively of elements. Each element is further composed of a *start tag* and *end tag*, and everything in between is the *content*:

Start Tag	Content	End Tag
`<p>`	`But it’s my only line!`	`</p>`

The start tag is also called the *opening tag* while the end tag is also called the *closing tag*. By the way, the name, or the abbreviation of the name of the element is written inside the tags. In particular, `p` stands for a paragraph. The closing tag should have an additional slash (/) before the element’s name.

In order for a paragraph to show in the browser, we need to add two more things to get what is generally considered the *minimum HTML document*. The first line should be a special declaration called *DOCTYPE*, which makes a clear announcement that HTML5 content follows. The DOCTYPE declaration is written within angle brackets with a preceding exclamation mark and the `html` keyword after it: `<!DOCTYPE html>`. Although it looks like a tag, this is actually the only part of an HTML document that isn’t a tag or an element. As a matter of fact, this code is here for historical reasons. I don’t want to kill you with details, but you have to include it if you want your document to be interpreted by the browser correctly.

One more thing that the minimum document should contain is a `<title>` element. This element is necessary as it identifies the document even when it appears out of context, say as a user’s bookmark or in search results. The document should contain no more than one `<title>` element.

Putting it all together, we get the following code:

```
<!DOCTYPE html>
<title>The Smallest HTML Document</title>
<p>But it’s my only line!</p>
```

Maria: You just showed us what the document code should look like. But I still don’t know where to type the code and how to view the resulting page.

1.3 The Tools

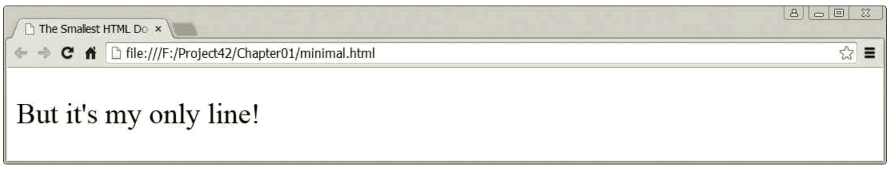
Professor: You can use any plain text editor you like. For example, you can use the Windows Notepad, which is already on your computer if you use Windows.

Mike: What if I don’t use Windows?

Professor: It doesn’t matter. Just about any operating systems contains a plain text editor. Personally, I use Notepad++, a programmer-friendly free text editor (*notepad-plus-plus.org*).

After you type the code, it is important that you save the file with a *.htm* or *.html* extension. While it doesn’t really matter which one you use, it is quite important that you choose one and stick to it consistently. Otherwise, you could throw yourself into a real mess. For example, you could easily end up editing two different files (same names, different extensions) thinking they’re one and the same file.

Now we open the file in a browser and voilà!



But it's my only line!

Notice how the content of the `<title>` element appears at the top of the browser tag.

Mike: How did you open the file in the browser?

Professor: Oh yes, sorry about that. Inside Notepad++, I chose Run→Launch in Chrome. If you use another browser, it will automatically appear under the Run menu item in your Notepad++. You can of course also simply double-click the file or drag and drop it into the browser. Once the file is open in the browser, you don't have to repeat this operation. If you modify the source code—the original HTML code, that is—you simply refresh the browser window. If you use Chrome like I do, you can do that by pressing F5. Later, you will use more than a single file to build a page. In that case, you will sometimes have to *force reload* all files of a page, which you can do by pressing Ctrl+F5 on Chrome. On Windows, to switch between the text editor and browser quickly, you press Alt+Tab, a standard key combination for switching between running tasks.

Maria: What would happen if we forgot to include the `<title>` element?

Professor: Nothing fatal, to be honest. One of the basic rules of rendering web pages is that the browser always tries its best to show the content. Of course, if the document isn't fully formatted according to the recommendations, the results are sometimes not in our favor. If you forget the title, then the name of the file containing the document usually takes over its role. If nothing else, that looks ugly and unprofessional.

1.4 Minimal HTML Document

Professor: One of the general prerequisites to good technical design is simplicity, which should not be confused with minimalism. In our last example, we saw a truly minimal HTML document, which you will rarely see in practice. Even with no extra content it is normally a good idea to flesh out this skeleton HTML document. For instance, most web developers share the belief that the traditional `<head>` and `<body>` elements can contribute to clarity, by cleanly separating your document into two sections. You pack all the content into the `<body>` section, while the other information about your page goes to the `<head>` section. Sometimes it is also a good idea to wrap both these sections in the traditional `<html>` element:

```
<!DOCTYPE html>
<html>
  <head>
    <title>The Smallest HTML Document</title>
  </head>
  <body>
    <p>But it's my only line!</p>
  </body>
</html>
```

Mike: I noticed that an element can contain not only text but another element as well. For example, you placed the `<title>` element within the `<head>` element.

Professor: Good observation! The content of an element can in fact be any valid HTML conforming to the rules of that specific element. We call putting one element into another *nesting*. When an element is nested (contains other HTML elements), it is important that it contains whole elements, including start and end tags. So if, for example, an `<elementA>` starts *before* an `<elementB>`, then it must by all means end *after* the `<elementB>`:

```
<elementA> ... <elementB> ... </elementB> ... </elementA>
```

The element that is contained inside another element *inherits* some of its behavior, and we often say that the contained element is a *descendant* of its owner, which is in turn its *parent*. The direct descendant is also called a *child*. This concept will become especially important when we come to styling elements with CSS. Now I only mention it so that later the terms will already sound familiar to you.

Maria: What are those periods inside?

Professor: Oh, yes. A set of three periods is an *ellipsis*. An ellipsis indicates the omission of content that is not important for understanding the explanation.

We will soon come back to our last example and furnish it with a little more. For that purpose we need another element called `<meta>`. This element is used to provide additional page description (so-called metadata), which is not displayed on the page, but can be read by a machine. The information stored in the `<meta>` element includes keywords, author of the document, character encoding, and other metadata. The `<meta>` element has neither content nor the closing tag:

```
<meta>
```

An element that is composed only of the opening tag is called an *empty* or *void* element.

Mike: I don't understand that. Where do you put all the information you talked about if there is no content?

Professor: That's the job for *attributes*. An attribute is the means of providing additional information about an HTML element. For example, by using the `src` attribute on the `<img>` element, one can tell the browser where to find the image to display. There are two things you should know about attributes: they are always specified after the element name in the start tag, and they come in name/value pairs like this one:

```
name="value"
```

The quotes (double style are the most common, but single style quotes are also allowed) around the value are not necessary under HTML5, as long as the value doesn't contain some restricted character (mostly =, >, or a space). That said, it is still a good practice that you always use quotes. Your code will look cleaner and more readable, which in turn lessens the possibility of errors. Likewise, it is not necessary that you use lower-case names and values, although it is recommended that you do so.

Some attributes do not have values. All that counts is their presence: they are either present or not, similar to an electrical switch, which can be put in on or off position. That type of attribute is called a *Boolean attribute* after an English mathematician, George Boole, the inventor of the so-called Boolean logic based on only two values. Because Boolean attributes have no value (the value is implied by their presence or absence), we omit the equals sign as well:

```
name
```

There are often cases when an element has more than one attribute. In that case the attributes are separated by spaces.

If I confused you with all this theoretical talking, don't worry. I will now show you how this works in practice.

Maybe you've already heard about the *character encoding*. Basically, that's a system that tells you how each character of a given repertoire is represented physically. In a computer, this physical representation consists of a series of ones and zeros, called bits. Relax, I'm not going into more detail with this explanation. The important thing is that the browser must know what encoding has been used to store the document text so it can read it back and show it properly. If you don't provide the encoding information to your markup, a browser will of course try to guess it, which may drive it into an obscure security issue. You should provide this information through the *charset* (that's short for character set) attribute of a `<meta>` element, assigning it the value `utf-8`. Today, more than half of all web pages are encoded in UTF-8 and honestly, I don't think you will ever need to use a character that is not included in UTF-8.

Let's check if you followed me. Can you add a `<meta>` element containing a character encoding attribute to our previous example without my help?

Maria: You didn't tell us in which section to put it. Let me think.... You said that the content went into the `<body>` and all the other information in the `<head>`. Something like the following code, perhaps?

```
<head>
  <meta charset="utf-8">
  <title>The Smallest HTML Document</title>
</head>
```

Professor: Exactly! One more thing here: when saving your work, don't forget to save it in UTF-8. For example, if you edit your document with Notepad (on Windows) the Save As dialog box lets you choose UTF-8 from the Encoding list at the bottom. In Notepad++, there is a top-level menu Encoding, under which you select Encode in UTF-8.

Maria: Is the order of the above elements within `<head>` important?

Professor: Not really. HTML5 only specifies that the character encoding declaration must be within the first 1024 bytes of the document. Translated into English, always include it as early as possible.

Mike: What if I wanted to write a page in some other language than English?

Professor: The encoding has nothing to do with the natural language of your web page. It only defines the set of characters you will use and, as I said before, you will hardly ever need a character (English or non-English) that is not a part of UTF-8. I'm glad you asked that, though. Your question takes us to the one last thing we should add to our basic document. Specifying a web page's natural language is considered a good style by many as it can be a useful piece of information for many users—for example, for filtering web search results by language. Interestingly, as far I as I know, Google Translate ignores this tag, relying on its own language-recognition algorithms. Anyway, you specify the language using the `lang` attribute on any element. For English, the attribute value will be `en`, and you can find codes for many other languages at *www.w3schools.com/tags/ref_language_codes.asp*. In a most likely scenario that your whole web page uses a single language (in our case English) you simply add the `lang` attribute to the `<html>` element:

```
<html lang="en">
```

By putting it all together we arrive at a decent starting point for any modern web page you want to build:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>The Smallest HTML Document</title>
  </head>
  <body>
    <p>But it's my only line!</p>
  </body>
</html>
```

Mike: I noticed that you indent the code, and even some lines more than others. How do you know how much to indent which line?

Professor: You don't have to know that. The indentation you see is in fact completely optional and without any effect on how the browser interprets the code. At the same time it is of the utmost importance for the person writing the code. Notice how the start and end tags of nested elements "see" each other. This "visibility" is made possible by indenting the content of an element. That way, the structure of the document stands out more clearly before the writer. In such a small document the advantage may not be evident at first glance, but believe me, when a document's size reaches several hundred lines, you want to have some order in your code.

Another friendly feature that helps programmers find their way through the chaos of computer code are *comments*. In HTML, comments are enclosed in a so-called *comment tag*, which starts with a left angle bracket, an exclamation mark, and two hyphens (<!−−), and ends with two hyphens and a right angle bracket (−−>). For example:

```
<!-- This is a comment and will not be visible in the browser. -->
```

Comments are completely ignored by the browser but quite useful for writing a short human-readable summary of code, for example. With proper comments, you don't need to decipher the code every time you need to upgrade it. Even if it is your own code, human readable remarks will help you tremendously to understand it later. Another practical use of comments is to temporarily switch off parts of code during testing and experimenting.

If you work in a team, or plan to make your code public, it is a good idea to include your name, contact information, and a licensing notice in comments at the top of your code. That way, people will have a chance to contact you in case they have questions about the code.

While comments are not visible in the browser window, be aware that they are accessible to the web page viewer through the View page source menu command. So, don't use comments to write filthy remarks about your boss or mother-in-law.

1.5 Formatting a Page

Professor: When I was a little boy, my mother used to read me fairy tales from the book by Joseph Jacobs, an Australian folklorist, and I still can't do without them. That's why we're going to mark up the beginning of *The Rose-Tree* as our next example. We will need three more elements for the job: a main heading (the `<h1>` element) for the title, an image (the `<img>` element) to include a fancy decorative capital letter, and quotes (the `<q>` element) for quoted speech.

In HTML, you can use six different levels of headings defined by the elements from `<h1>` to `<h6>`. `<h1>` defines the most important heading while `<h6>` defines the least important heading. Since our fairy tale only has one heading, we will of course use the `<h1>`:

```
<h1>The Rose-Tree</h1>
```

With the image element, as you might have guessed, you can include a picture into a page. You do that by specifying the file in which the picture is stored using the `src` attribute: the value of the `src` attribute is the name of the file. Note that `<img>` is an empty element so it only has the opening tag. If the image is not a key part of the content, then it is a good idea to include the `alt` attribute as well. This attribute provides a textual equivalent to show in case the image cannot be displayed or until it is downloaded.

Our fairy tale begins with a letter T. You can use some graphic software to draw a decorative T and save it to the image file named *T.jpg*. The final code for our image element looks like this:

```
<img src="T.jpg" alt="T">
```

Because the decorative capital letter is not an essential part of the content, I provided alternative text, which is obviously the letter T.

Incidentally, not all image formats are supported by browsers. The most commonly used formats that are supported are *.gif*, *.jpg* and *.png*. In the above code, I used only the file name without any path as the value of the `src` attribute. That means that the file resides in the same folder as the HTML file that contains the above `<img>` element. Should the image file be stored elsewhere, I would have to prepend the corresponding path to it.

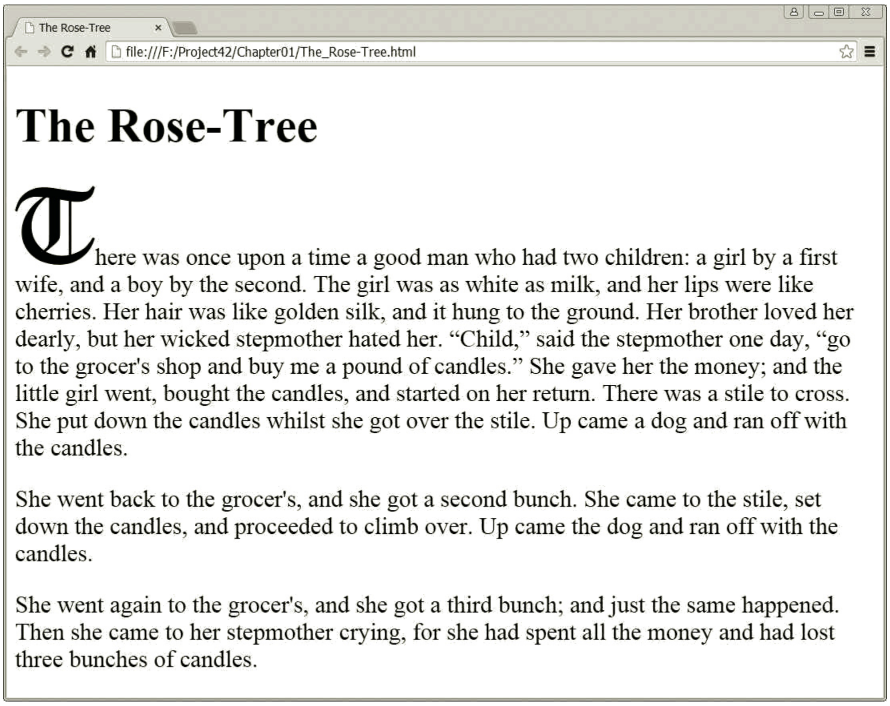
The `<q>` element is used to represent some quoted content and is usually rendered as a pair of quotes around the marked content.

OK, let's put it all together. The following code goes into the `<body>` element:

```
<!-- From the e-book English Fairy Tales, collected and
  -- edited by Joseph Jacobs. Belongs to the public domain.
  -- Source: www.authorama.com
  -->
<h1>The
     Rose-Tree</h1><p><img src="T.jpg" alt="T">here
was once upon a time a good man who had two children:

a girl by a first wife, and a boy by the second. The girl was
as
    white as milk, and her lips were like cherries. Her hair
  was like golden silk, and it hung to the ground. Her brother
  loved her dearly, but her wicked stepmother hated her.
  <q>Child,</q> said the
  stepmother one day, <q>go to the grocer's shop and buy me
   a pound of candles.

   </q> She gave her the money; and the little girl
  went, bought the candles, and started on her return. There
  was a stile to cross. She
                    put down the candles whilst she
  got over the
 stile. Up came a dog and ran off with the candles.</p><p>She
  went back to the grocer's, and she got a second bunch.
```

```
She came to the stile, set down the candles, and proceeded to
climb
        over. Up came the dog and ran off with the candles.</p>
        <p>
She went again to the grocer's,

              and she got a third bunch; and
just the same
happened. Then she came to her stepmother crying, for she had
spent all the money and had lost three bunches of candles.</p>
```

Deliberately, I made a mess out of the text and tags so that the structure of the document is not obvious at first glance. The rendering, however, is quite appealing.



The Rose-Tree

There was once upon a time a good man who had two children: a girl by a first wife, and a boy by the second. The girl was as white as milk, and her lips were like cherries. Her hair was like golden silk, and it hung to the ground. Her brother loved her dearly, but her wicked stepmother hated her. “Child,” said the stepmother one day, “go to the grocer's shop and buy me a pound of candles.” She gave her the money; and the little girl went, bought the candles, and started on her return. There was a stile to cross. She put down the candles whilst she got over the stile. Up came a dog and ran off with the candles.

She went back to the grocer's, and she got a second bunch. She came to the stile, set down the candles, and proceeded to climb over. Up came the dog and ran off with the candles.

She went again to the grocer's, and she got a third bunch; and just the same happened. Then she came to her stepmother crying, for she had spent all the money and had lost three bunches of candles.

Maria: I notice that the browser does not obey your original text formatting.

Professor: Well.... Yes and no. Actually it obeys the rules perfectly, it's just that the rules are a little different from what you might have expected.

The first thing you may have noticed is that the browser ignores spaces, tabs and newlines. OK, it does not ignore them completely. For example, if there is a separation between words, it is replaced by a single space regardless of what I have actually put there: a space, tab, newline, or even more of them. A rule of thumb is that spaces, tabs, and newlines are ignored unless they represent the only separation between two entities—words, for example. Even then they are replaced by a single space. If there exists some other separator like, for example, a tag or equals sign (=), then spaces are not needed at all and it doesn't matter whether they are there or not—the result is always the same.

That said, there are a few cases where you should be careful about spaces:

- Do not put any spaces *before* the element name in the opening or closing tag. It would be wrong to write `< p>` or `< /p>` or `</ p>`.
- Do not put any unnecessary spaces between double quotes when writing attribute values. In that sense, `lang="en"` is quite different from `lang=" en "` while `lang = "en"` is still OK while it does not change the meaning.
- You should always put spaces between two attributes of the same element, even when there's a quotation mark at the end of a value. For example, it would be incorrect to write `<img src="T.jpg"alt="T">` instead of `<img src="T.jpg" alt="T">`.

It is important at this point that you start seeing HTML as a language that gives a document a *structure* and *meaning* rather than a specific look. How a page looks is taken care of by the browser. Later, we will control the document's look by means of CSS.

Mike: I have a question. If spaces and newlines do not affect the text formatting, how can you tell the browser to start, for example, some text on a new line?

Professor: This kind of formatting is implied in the element meaning or *semantics*. For example, it is usual practice that a heading or a paragraph is displayed as a block of text occupying the whole line—nothing else can be positioned on the same line in a browser window. In other words, a line break is inserted before and after a heading or paragraph. We say that such elements have a *block display*. On the other hand, an image can happily inhabit a line together with other elements, if there is enough space, of course. Such elements are said to have an *inline display*.

You'll see later that it is possible to change a type of display for an element using CSS. Doing that, however, does not change the intrinsic HTML categorization of elements. For this reason we will use the terms *inline element* and *block element* to denote elements that have an inline or block display by default.

Mike: But what if I simply want to break a line without making a paragraph or using any other block element? Is there a way to do that?

Professor: There's an element for breaking a line called `<br>`, which is short for break. Paradoxically, that element should not be used for breaking lines unless you are breaking lines because you want to convey some meaning. Typical examples are breaking lines in poems or postal addresses. If you want to break a line because you are introducing a new paragraph, then you shouldn't use the `<br>` element. A paragraph is a meaning by itself and doesn't need other elements to promote line breaks. If you don't like the amount of space between paragraphs, however, that's not a matter of semantics. You take care of stuff like that by using CSS.

Mike: I'll try to remember that, but I think I'll have to wait to get some experience to fully understand it.

Professor: That's true.

Maria: What happens with text that is written outside of any element?

Professor: Oh yes, that is extremely important. I'm glad you asked. It is strongly recommended that you do not put any plain text directly into the body. It is a good practice that you mark up every text and that you mark it up properly. I believe that's easier said than done, but as we go along you will get some experience and feeling about how to get it right. All you have to be careful about is to think of the role of the text you are writing: is it a heading, a paragraph, a caption, a sidenote? When you decide upon the role of the text, your next step is to pick up an appropriate HTML element to mark up that text. If none seems right, then generic `<span>` and `<div>` elements come in handy, but more on these later.

Mike: Is HTML case sensitive?

Professor: For the most part, no. However, just to stay on the safe side, I strongly recommend that you only use lower-case letters. If nothing else, it will save you the trouble of remembering whether case matters or not.

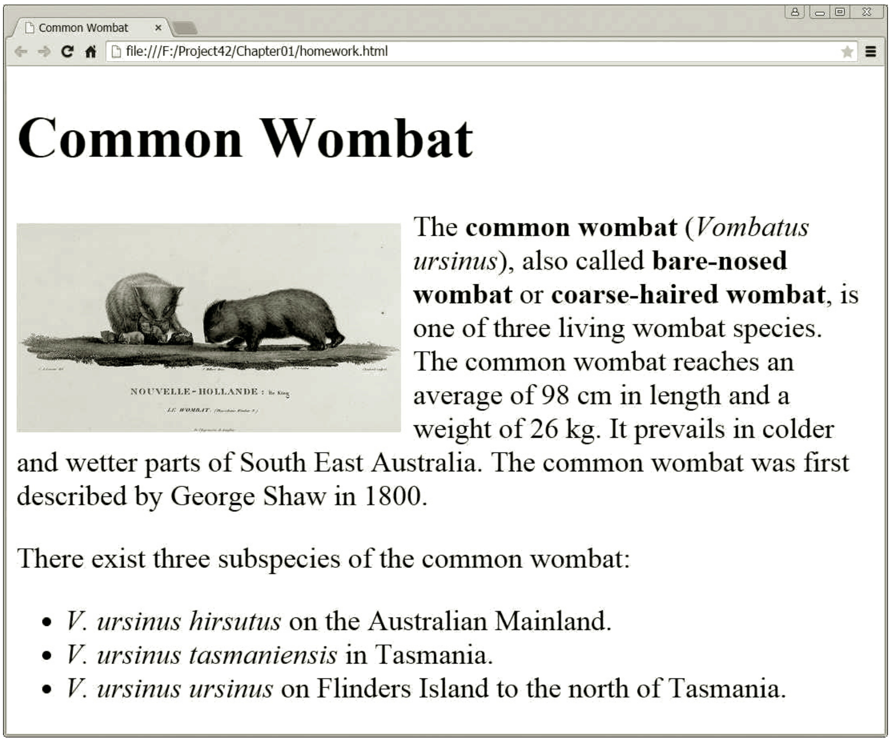
1.6 Homework

Professor: Before we call it a day, let me just give you lots of homework. Note that it's not important that you do everything right. The important thing is that you do it on your own and that you ask yourselves questions about why things work (or don't work) the way they do. Write down any unresolved questions, which we will use in our next meeting.

I encourage you to search and use, apart from the material we covered today, other sources as well. That's important. Because of the constant progress of technology, you'll always be searching for new answers. For your convenience, I prepared some reference material you will find at the end of this book. However, this is not a complete reference. I have only put into it what I thought would be important for our course. So if something is not there, that doesn't mean it doesn't exist. For the time being, I recommend that you use the site *www.w3schools.com*, which offers a decent learning experience for a beginner. One more good thing about this site is that it includes lots of working examples ready for you to play with. When you level up, however, I suggest that you start using other, more profound sources to learn from.

And now the homework. I have prepared for you a short document on wombats, which I want you to reproduce as faithfully as you can while using only HTML. If you don't like the picture, you can use another one. Personally, I like this old drawing of the now-extinct wombats of King Island, Tasmania, by Charles-Alexandre Lesueur from 1807.

Here's how the document should look:



Common Wombat

file:///F:/Project42/Chapter01/homework.html

Common Wombat

The **common wombat** (*Vombatus ursinus*), also called **bare-nosed wombat** or **coarse-haired wombat**, is one of three living wombat species. The common wombat reaches an average of 98 cm in length and a weight of 26 kg. It prevails in colder and wetter parts of South East Australia. The common wombat was first described by George Shaw in 1800.

There exist three subspecies of the common wombat:

- *V. ursinus hirsutus* on the Australian Mainland.
- *V. ursinus tasmaniensis* in Tasmania.
- *V. ursinus ursinus* on Flinders Island to the north of Tasmania.

For some more homework, here is a short list of keywords covering today's meeting. Draw a mind map using all the given keywords, adding some more if you feel like it.

In this meeting: element, start tag, end tag, content, indentation, comment, nested element, `DOCTYPE`, descendant, parent, child, markup, empty element, void element, attribute, attribute name, attribute value, Boolean attribute, `<html>`, `<head>`, keywords, author information, `<meta>`, `<title>`, `<body>`, `<p>`, `<img>`, `<h1>`, `<br>`, `alt`, `src`, `charset`, `lang`, block display, inline display, block element, inline element, spaces, generic elements, semantics

MEETING 2

Building a Sound Structure

2.1 Homework Discussion

Professor: I am anxious to see what you have written since our last meeting. Did you have any trouble?

Mike: At the beginning, yes. I had to read about lists and tables to complete the homework.

Professor: No offense, but I am sort of glad you mention tables because that's a sure signal you did it wrong. Don't panic, though. That mistake will help you tremendously with mastering basic HTML principles. What about you, Maria?

Maria: I don't know.... It wasn't as easy as it looked and I have some questions.

Professor: Good. So we have material to talk about. Mike, what did you use a table for?

Mike: I haven't found any other way to wrap text around the picture. And honestly, I had that annoying feeling that this is not the way to go. Namely, I had to split the text between two cells of the table in order to wrap it around the image. The whole thing only looked good till I resized the window. So how can this be done properly?

Professor: As I already mentioned in our previous meeting, HTML is used to give a document a structure rather than a look. So there is always one very important question to bear in mind when constructing an HTML document: is what I'm trying to do in any way affecting the *structure* of the document or is it merely a matter of *presentation*? Indeed, wrapping a text around a picture has nothing to do with structure. A paragraph is a paragraph and a picture is a picture, no matter how they are positioned and formatted. You don't have to worry about whether a paragraph is wrapped around a picture when writing an HTML. That's work for CSS. I've deliberately presented you with a document that will tempt you to think about how it looks, which is completely irrelevant at this point.

Mike: So it's not possible to wrap text using pure HTML?

Professor: No.

Maria: Why not? I managed to find an attribute called `align`, which aligns a picture to one side (left or right) and wraps text around it nicely. Have I missed something?

Professor: You missed one thing. The `align` attribute is deprecated since HTML4.01 and obsolete since HTML5, and not without a reason. Nearly all elements and attributes that were historically used for presentational purposes are considered obsolete in HTML5. That doesn't mean they don't work, though. Worse still, they will work for quite some time for backward compatibility. Browsers are really forgiving when rendering a markup. You won't notice that you did something wrong because the mistake won't show. It is entirely up to you whether you conform to the rules or not. If you want to build sound web pages, the HTML is the most important thing to do right because it serves as a foundation of your whole web page. Sticking to plain, simple HTML is also important because it keeps search engines happy. The old HTML's approach of using formatting elements like `<font>`, or tables to lay out a page, stands in the way of a search engine's job.

Mike: Is there really no way of testing whether your HTML is written properly?

Professor: In fact there is. When in doubt, you can resort to one of many on-line HTML validators (for example, *validator.w3.org*). You can upload your document to a validator and it will alert you to any errors you may have in your HTML document.

Maria: I think I see things more clearly now and meanwhile you also answered some of my questions. I even discovered another mistake I made: I used the `<h2>` element for the main heading (Common Wombat) since the `<h1>` seemed too big to me. If I understand correctly, I shouldn't have done this because I was actually concerned about presentation. In fact I should have used `<h1>` because this is the main heading.

Professor: Exactly! When writing HTML, you shouldn't worry about font size.

Maria: What still puzzles me are the elements `<b>` (bold) and `<i>` (italic), which I found in the reference you gave us. Is this not presentation?

Professor: If you use them properly, then the answer is no. Those two elements survive from the past but with a different, more semantic interpretation. In modern HTML, it would be wrong to use `<b>` and `<i>` elements merely for the purpose of making text bold or italic. It is also wrong to think of `<b>` and `<i>` elements as being old elements now replaced by the `<strong>` and `<em>` elements only because they are rendered in the same way by many browsers. The `<strong>` element is used to express strong importance of its content and the `<em>` element is used to emphasize its content. The `<b>` and `<i>` elements are not used to stress importance or emphasis. Rather, they are considered as generic bold and italic elements used in cases when normally—but not necessarily—bold and italic fonts are used. Because the meaning of these two elements is not evident from their names, authors often use the `class` attribute to clearly identify their semantic meaning. For example:

```
<i class="latin-taxonomy">Vombatus ursinus ursinus</i>
<b class="english-definition">bare-nosed wombat</b>
```

The `class` attribute not only gives a clear meaning to both elements but also allows CSS to access these elements for formatting purposes—but more on that later.

Mike: There's one more question bothering me. I have found an element for writing unordered lists (`<ul>`) and I'm not quite sure whether I should put it inside a paragraph or not. A list, in my opinion, is a part of a paragraph but visually it is represented as a separate block. I know that I am mixing two concepts, which you've just clearly separated, but in this case my thinking is a bit blurred.

Professor: I agree that it is a matter of debate whether a list is a separate paragraph or not. In practice, things are sometimes not as clear as in theory. Luckily, in this special case we have recourse to the additional rules concerning *context* in which certain HTML elements can appear, and the *content* that they are allowed to include. One or both of the terms also appear with some of the element descriptions in the concise reference at the end of this book. For example, you will find that the `<p>` element should only contain text and inline elements. Because the `<ul>` element is a block element, you shouldn't put it inside a `<p>` element.

Mike: You have already told us about the block-inline categorization of elements based on their default display setting, but I'm still confused. When you are talking about display, isn't that presentation?

Professor: You couldn't be more right about that. In HTML5, a display has become purely a CSS term since it defines the visual behavior of an element. In the past, the categorization of block and inline elements helped authors in deciding which element is allowed as a content (descendant) of the other. In HTML5, this binary categorization has been replaced by a more complex one and you will hear terms like the *flow content*, *sectioning content*, *phrasing content*, and so on. If you are interested in studying these (which I don't actually recommend at this stage), I suggest that you visit the site *developer.mozilla.org*, which is written on a higher technical level than *www.w3schools.com*, but still more understandable for a beginner than the World Wide Web Consortium's (W3C) page, *www.w3.org*, which publishes original web standards and is quite a demanding read.

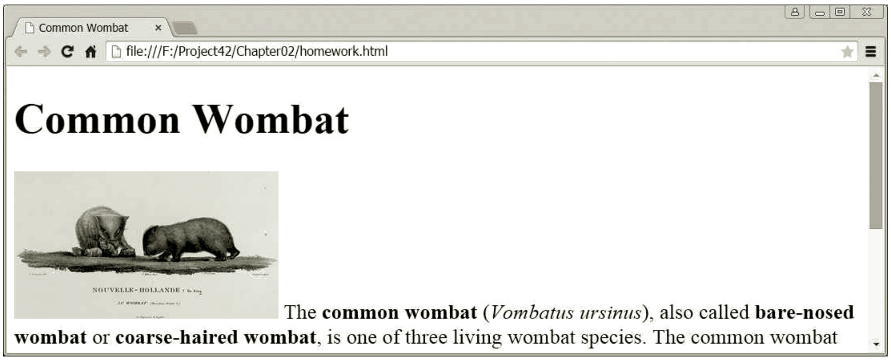
To keep things simple, some authors equate the flow content category roughly with the block display category and the phrasing content category roughly with the inline display category. This block-inline categorization is easier to understand. That's why it is still used by some authors.

As a rule of thumb, an inline element can only contain inline elements while a block element can contain inline elements as well as block elements. A notable exception to the rule are elements `<p>` and `<h1>` to `<h6>`, which are block elements but cannot contain block elements. Again, when in doubt, you can use an HTML validator.

OK, that's it. Here is a possible final solution of your homework:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Common Wombat</title>
  </head>
  <body>
    <h1>Common Wombat</h1>
    <p><img src="King_Island_wombats.jpg">
      The <b class="english-definition">common wombat</b>
      (<i class="latin-taxonomy">Vombatus ursinus</i>), also called
      <b class="english-definition">bare-nosed wombat</b> or
      <b class="english-definition">coarse-haired wombat</b>, is
      one of three living wombat species. The common wombat reaches
      an average of 98 cm in length and a weight of 26 kg. It
      prevails in colder and wetter parts of South East Australia.
      The common wombat was first described by George Shaw in 1800.
    </p>
    <p>
      There exist three subspecies of the common wombat:
    </p>
    <ul>
      <li><i class="latin-taxonomy">V. ursinus hirsutus</i>
        on the Australian Mainland.</li>
      <li><i class="latin-taxonomy">V. ursinus tasmaniensis</i>
        in Tasmania.</li>
      <li><i class="latin-taxonomy">V. ursinus ursinus</i>
        on Flinders Island to the north of Tasmania.</li>
    </ul>
  </body>
</html>
```

Note that text will not wrap around the picture because, as I already pointed out, this cannot be done using only HTML.


Common Wombat

file:///F:/Project42/Chapter02/homework.html

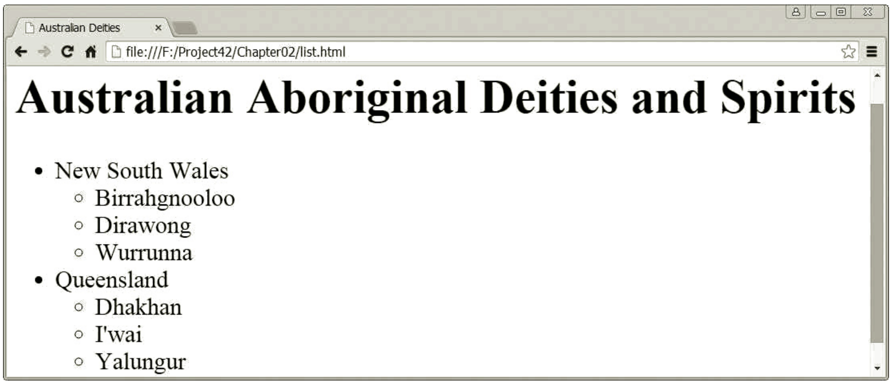
Common Wombat

The **common wombat** (*Vombatus ursinus*), also called **bare-nosed wombat** or **coarse-haired wombat**, is one of three living wombat species. The common wombat

2.2 Lists and Tables

Professor: I'm glad that you succeeded in finding and using unordered list on your own. Still, I would like to point out some things about elements `<ul>` (unordered list) and `<ol>` (ordered list). Basically, they are the same, only the first one is used when the order in which the items are listed is completely irrelevant, while the second one is used when the order is important. The items in an unordered list are usually preceded by bullets while in an ordered list they come with ordinal numbers or letters. The use of lists is quite straightforward. There's one thing, however, you need to be careful about.

If you look at element descriptions in the reference at the end of this book, you will notice that the elements `<ul>` and `<ol>` alike can only contain `<li>` (list item) elements and nothing else. Since a list is composed of items, this is hardly a surprise. Still, many people find themselves completely at a loss for where to put sublists. As we already discussed, browsers are quite tolerant of bad HTML code and you won't know whether you did it right unless you use a validator. So, let me ask you a question. Where do you think one should put a sublist? Or, more specifically, could you write HTML code for the following list?



Australian Aboriginal Deities and Spirits

- New South Wales
  - Birrahgnooloo
  - Dirawong
  - Wurrunna
- Queensland
  - Dhakhan
  - I'wai
  - Yalungur

Maria: Let me think. A `<ul>` element can only contain `<li>` elements. That means that I cannot put another list inside a list. On the other hand, a sublist *should* be a part of a list. Now, the only possibility I see is to put a sublist inside an `<li>` element. Am I right? Like this:

```
<ul>
  <li>New South Wales
    <ul>
      <li>Birrahgnooloo</li>
      <li>Dirawong</li>
      <li>Wurrunna</li>
    </ul>
  </li>
  <li>Queensland
    <ul>
      <li>Dhakhan</li>
      <li>I'wai</li>
      <li>Yalungur</li>
    </ul>
  </li>
</ul>
```

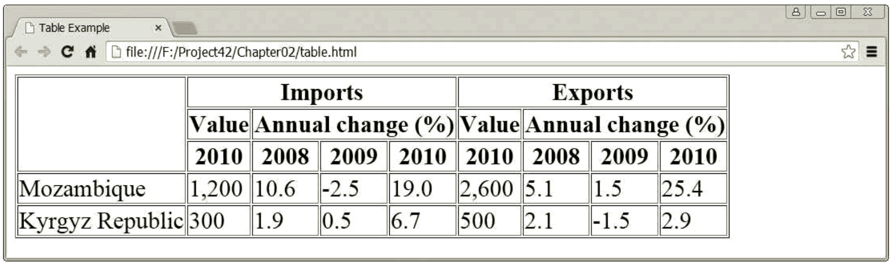
Professor: Perfect! When you think of it, a sublist in fact always relates to a specific list item rather than a list as a whole. It is therefore the only logical solution to put a sublist inside a list item.

Lists allow authors to organize document data in a specific way. Another such element is `<table>`. Just like lists, the `<table>` element also has its content limited to a small number of allowed direct descendants. Amongst them you'll find an optional `<caption>` element and an obligatory `<tr>` element. Each `<tr>` (table row) element represents a row in a table and its only direct descendants can be `<td>` (table data) and `<th>` (table header) elements. The former represent table data cells and the latter table header cells. You can stretch any data or header cell over more rows or columns using their `rowspan` and `colspan` attributes, respectively. The `<caption>` element holds the table caption and should appear before any `<tr>` elements.

Here's a complicated example:

```
<table border="1">
  <tr>
    <td rowspan="3"></td>
    <th colspan="4">Imports</th><th colspan="4">Exports</th>
  </tr>
  <tr>
    <th>Value</th><th colspan="3">Annual change (%)</th>
    <th>Value</th><th colspan="3">Annual change (%)</th>
  </tr>
  <tr>
    <th>2010</th><th>2008</th><th>2009</th><th>2010</th>
    <th>2010</th><th>2008</th><th>2009</th><th>2010</th>
  </tr>
  <tr>
    <td>Mozambique</td>
    <td>1,200</td><td>10.6</td><td>-2.5</td><td>19.0</td>
    <td>2,600</td><td>5.1</td><td>1.5</td><td>25.4</td>
  </tr>
  <tr>
    <td>Kyrgyz Republic</td>
    <td>300</td><td>1.9</td><td>0.5</td><td>6.7</td>
    <td>500</td><td>2.1</td><td>-1.5</td><td>2.9</td>
  </tr>
</table>
```

And the result in the browser.



	Imports				Exports			
	Value	Annual change (%)			Value	Annual change (%)		
	2010	2008	2009	2010	2010	2008	2009	2010
Mozambique	1,200	10.6	-2.5	19.0	2,600	5.1	1.5	25.4
Kyrgyz Republic	300	1.9	0.5	6.7	500	2.1	-1.5	2.9

Mike: I think I will study this when it's quiet and do some experimenting to see how things behave.

Professor: I just wanted to suggest the same. There's really not much to explain, you'll simply have to try it out by yourselves. However, before you start pulling your hair out figuring out why I used an obsolete `border` attribute, let me tell you that I haven't had any other choice.

Maria: That's true. Border is a matter of presentation, not content.

Professor: Precisely. However, browsers by default don't draw any table borders and the above table would be completely illegible without them. The only way to draw borders properly according to HTML5 standard is to use CSS. In the above example, I resorted to the possibility of limited use of the `border` attribute in the case of absence of CSS. W3C allows that but you can only use one of the two possible values for this purpose: the empty string or the value 1.

Maria: What about `rowspan` and `colspan`? Aren't they also a matter of presentation?

Professor: Think again.

Maria: Oops! I get it. We stretched a header across many columns because they share the same *meaning*. For example, there are four columns about export.

Professor: Exactly!

2.3 Generic `<div>` and `<span>` Elements

Professor: By now, you already understand that writing HTML code is thinking primarily about meaning and structure.

Equally important is that you use the right element for assigning a part of a document the right meaning. For example, it would be wrong to use any of the six headings elements `<h1>` to `<h6>` for a table caption. For that purpose, you should use the `<caption>` element and nothing else.

The difficulty with the older HTML standards was that there really weren't many semantic elements available and authors had to resort to the generic `<div>` and `<span>` elements. In this sense, the HTML5 standard has made a significant step forward by introducing many new semantic elements.

Before I introduce you those newbies, I think it's important that you understand the role of the `<div>` and `<span>` elements. There are two good reasons for that: First, designers have used them in the past very often to organize their documents, which will continue to live on the Web for years to come. Second, they are still very useful in special cases when you don't find a more appropriate element.

`<div>` and `<span>` are called *generic* elements. That means there really isn't any inherent meaning attached to them nor is there provided any default browser rendering for them (except that `<div>` has block display while `<span>` has inline display). Rather, they can be used generally wherever needed. Authors use the `class` or `id` attributes to attach to them a specific semantic meaning, and CSS to make them look any way they want. For example, you can use the `<div>` element to divide a page into logical pieces like headers, footers, banners, sidenotes, and so forth. Later, you can design and position each piece to create page layouts to your liking using CSS.

Say you want to include in your website a copyright notice, which usually comes at the bottom of every page of your website. In typography, a piece of information that is separated from the main body of text and appears at the bottom of the page is called a *footer*. Using the `<div>` element, the code for a copyright notice could look something like this:

```
<div class="footer">
  (C) ACME 2015 All Rights Reserved
</div>
```

With the help of a `<div>` element with the `class` attribute set to `footer`, I have clearly marked the copyright text as belonging to the footer of a page. The same `class` attribute could help me later to access and visually design the element by CSS.

The generic `<div>` and `<span>` elements are useful for giving parts of a document an arbitrary, user-defined meaning. However, one of the ambitions of HTML5 is to bring into play other, more semantic elements to enable authors to easily and more accurately describe the content of their documents. For example, there actually exists the `<footer>` element and you can rewrite the above code as:

```
<footer>
  (C) ACME 2015 All Rights Reserved
</footer>
```

Maria: I see. And now the browser will already know that this is a footer and will render it smaller, won't it?

Professor: Actually, no. The `<footer>` element doesn't do anything on its own except render its content as a separate block. As a matter of fact, none of the new semantic elements will do anything special except display themselves as a block of text. Apart from that, the four sectioning elements that we are going to meet shortly, `<article>`, `<aside>`, `<nav>`, and `<section>` make their headings smaller and that's about it.

Maria: What's the use, then?

Professor: There are many reasons, actually. One of them is code readability. The limitation of the generic `<div>` and `<span>` elements is that they don't carry any information about the structure of the page. OK, authors label their generic elements using `class` and `id` attributes, but the values given to them are quite arbitrary. Sometimes it takes a lot of digging through HTML and CSS files to unscramble what the author actually had in mind. Even if a human can do that unscrambling, a search bot or a screen reader surely can't. The page structure remains a complete mystery to them.

Mike: Are you saying that search engines can profit from the sound page structure?

Professor: Right now the concept is maturing. Designers already use HTML5 semantic elements to build clearer document structures. They can be used to advantage by search engines to build a better page preview, or by screen readers to better guide visually impaired users through a deep forest of sections and subsections.

Let's take another example. Imagine you want to add to your page an enhanced heading, which is not just a title but also includes a subtitle, a byline, and a teaser. For that purpose, the `<header>` and `<hgroup>` elements come in handy:

```
<header>
  <hgroup>
    <h1>Twelfth Night</h1>
    <h2>Or what you will</h2>
  </hgroup>
  <div class="byline">by William Shakespeare</div>
  <div class="teaser">Things get complicated as Lady Olivia flips
  over Cesario (who is really Viola in disguise) and Viola secretly
  loves the Duke, who thinks she is a man.</div>
</header>
```

The `<hgroup>` element can only contain headings (`<h1>` to `<h6>`) and its purpose is to hide all the lower-rank headings and expose only the highest-rank heading to the document structure. That way you can use the `<h2>` heading to include a subtitle without a danger that the rest of the text becomes subordinate to the `<h2>` heading. If that happened, the entire play would end up as one big subsection, which wouldn't make much sense. For a byline and teaser there's no semantic elements; that's why I've used `<div>` elements with the `class` attribute. Again, this attribute could help me visually design both elements later using CSS. I wrapped my complete enhanced heading into a `<header>` element, signifying that everything is in fact the heading of the play.

2.4 Sectioning Elements

Professor: Although the `<header>` and `<footer>` elements are important for structuring a document, they don't contribute to the *outline* of the page. An outline is an important notion in HTML. You can think of it as a table of contents, where each section or subsection has its own title and hierarchical position within a document. In HTML, only the so-called *sectioning elements* contribute to the document outline. There are four of them: `<article>`, `<section>`, `<aside>`, and `<nav>`. While each of the six heading elements (`<h1>` to `<h6>`) also starts a new section, they are not considered sectioning elements per se because they do not *contain* a section. Rather, they only mark the *beginning* of a section. Therefore, their control over sections is somewhat limited.

Mike: Is an outline like a mind map?

Professor: As a matter of fact, it is. A mind map or table of contents—it's all the same. A mind map helps you organize your thoughts, and the structure (big picture) is just as important as the content itself. An outline is like a mind map not so much helping humans directly as aiding different software like screen readers or search engines.

Maria: I'm curious how our last example looks in a browser.

Professor: This is not relevant right now. Although we *will* try and shape it into a pleasing form, that comes later on our menu.

Maria: OK, I understand that the look is not important right now. You mentioned, however, that the `<hgroup>` element hides all the lower-rank headings. Does that mean that we won't see the `<h2>` heading?

Professor: The heading is not hidden from the document in the browser window but from the *document outline*. Put differently, if you wanted to create a table of contents, then the `<h2>` heading from our last example wouldn't be in it.

I would like to show you how the document outlining works with the next example:

```
<body>
  ...
  <article>
    <header>
      <hgroup>
        <h1>10 Things You Should Do Before The Deluge</h1>
        <h2>You Better Hurry Up</h2>
      <hgroup>
    </header>
    <p>...</p>
    <section>
      <h2>Pay Taxes</h2>
      <p>...</p>
      <aside>...</aside>
    </section>
    <section>
      <h2>Visit People</h2>
      <p>...</p>
      <section>
        <h3>Your Mom</h3>
        ...
      </section>
      <section>
        <h3>Your Best Friend From Childhood</h3>
        ...
      </section>
    </section>
    ...
    <footer>
    ...
    </footer>
  </article>
  ...
</body>
```

There are lots of things going on in the code. Yet it is not as complicated as it seems. Every sectioning element defines its own section, and whenever one sectioning element comes nested within another, the corresponding sections become hierarchically related. The first heading element inside a sectioning element takes the role of the heading for that section. If there's no heading element within a sectioning element, then the section is considered untitled. Visualization can probably help you understand the outlining concept, even more so if you can try it by yourselves. I suggest that you take one of the many free outlining tools or *outliners*, and experiment at home to see how it works. For example, *gsnedders.html5.org/outliner*. You can either upload a file or simply copy and paste HTML code into the provided text box. Unlike a validator, an outliner does not require a complete HTML document. You only need to include elements that contribute to the document outline.

After feeding the outliner with our last example, we get the following outline:

1. *Untitled Section*
    1. 10 Things You Should Do Before The Deluge
        1. Pay Taxes
            1. *Untitled Section*
        2. Visit People
            1. Your Mom
            2. Your Best Friend From Childhood

Notice the *Untitled Section* subordinate to the *Pay Taxes* section, which was generated by the `<aside>` element without a heading. By the way, the `<aside>` element is normally used for a content that is related to the main text but not essential for its understanding. So if you remove the `<aside>` element, the main text should still make sense. From the document outline point of view, however, the `<aside>` element does exactly the same as `<article>` and `<section>` elements do.

Mike: What about the main *Untitled Section*? Where does that come from? If I get it right, then the `<article>` element is the main section, which should be assigned the title of the first heading element inside the `<hgroup>` element, shouldn't it?

Professor: Why do you think so?

Mike: Simply because `<header>` and `<hgroup>` elements do not contribute to the document's outline, as you said before. You also said that the `<hgroup>` element hides all its contained headings except the first one, which is exposed to the outline. That's OK because *You Better Hurry Up* doesn't appear in the outline. Still, the main title should be the title of the top-level section, which it is not. Have I missed something?

Professor: No, you're right. I see that you understand the concept, so we are ready to move on.

There is another group of elements called *sectioning roots*. A sectioning root defines a root, or top-level section. The sections inside a sectioning root do not contribute to the outlines of other sectioning roots. Rather, they form a new outline of their own. At this moment, that's not very important, though. The only reason I'm telling you that is because `<body>` is one of the sectioning roots. Therefore, the main *Untitled Section* you see in the above outline belongs to the `<body>` element, which acts as a sectioning root for the entire page (provided there are no other sectioning roots). If you want your page outline to have a title, then you should include an appropriate heading element outside of any other sectioning element (but inside the sectioning root). At home, try to add a heading element outside of the `<article>` element in the above example (either before or after it) to see what happens. As far as the outline is concerned, it doesn't matter where inside an element a heading appears as long as it is the first one—although it is a little weird to write the page title at the end, or even in the middle of the document.

Maria: What happens if you include more headings inside a section? Are they hidden the same way as within the `<header>` element?

Professor: No. Each subsequent heading within a section implicitly starts a new section. If a heading has a lower rank than the one before it, then a new section is created, which is subordinate to the enclosing section. If, however, a heading has the same or higher rank than the one before it, then a new section is created on the same level in the hierarchy as the enclosing section of that heading. The enclosing section is closed even though its closing tag has not been reached yet because the newly created section is on the same hierarchical level. Needless to say, that can cause some confusion. I suggest that for homework you add some headings of different ranks to different places in our last example and observe what happens. That way, if you don't get lost, you will understand things a little better.

Maria: If a heading itself already starts a new section, why then do we need additional explicit section definitions?

Professor: There are numerous reasons for that, many of which may not be quite evident at first glance. Consider, for example, that a website reuses some material from other sites, say in the form of a news feed. Such a process is called *web syndication*. It can easily happen that an article from another web page is pulled under a heading with a rank that is lower than that of the included article. Although the page will still work fine, the hierarchy will become disturbed, which could make the page more difficult for search engines and other software to process. However, if you use HTML5 sectioning elements, the hierarchy of the sections is defined by the placing of the sectioning elements regardless of the ranks of headings within them.

2.5 Hyperlinks

Professor: All our discussion so far was related to the meaningful structure of a single web page. A modern website is, however, more than just a single page. It is important that you can build from different pages an ordered and logical structure where your visitor will promptly and effortlessly find the right information. That's an important part of what we call *web design*, which is unfortunately not the main focus of our course. We concentrate more on how things are done technically, although not in complete oblivion of web design considerations.

Technically, for connecting one web page with another document, we can establish a