DOM tree

The backbone of an HTML document is tags.

According to the Document Object Model (DOM), every HTML tag is an object. Nested tags are "children" of the enclosing one. The text inside a tag is an object as well.

All these objects are accessible using JavaScript, and we can use them to modify the page.

For example, `document.body` is the object representing the `<body>` tag.

Running this code will make the `<body>` red for 3 seconds:

```
document.body.style.background = 'red'; // make the background red

setTimeout(() => document.body.style.background = '', 3000); // return back
```

Here we used `style.background` to change the background color of `document.body`, but there are many other properties, such as:

- `innerHTML` – HTML contents of the node.
- `offsetWidth` – the node width (in pixels)
- …and so on.

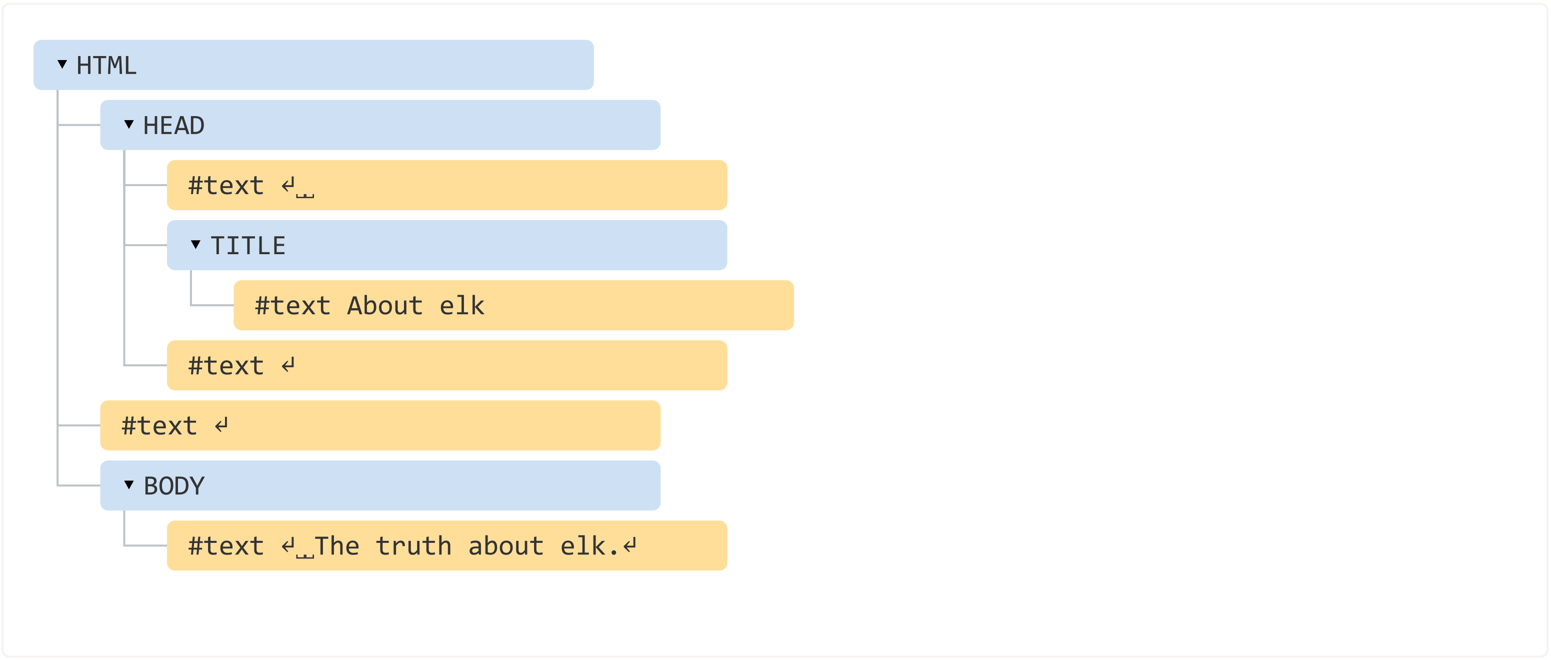
Soon we'll learn more ways to manipulate the DOM, but first we need to know about its structure.

An example of the DOM

Let's start with the following simple document:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>About elk</title>
</head>
<body>
  The truth about elk.
</body>
</html>
```

The DOM represents HTML as a tree structure of tags. Here's how it looks:



On the picture above, you can click on element nodes and their children will open/collapse.

Every tree node is an object.

Tags are *element nodes* (or just elements) and form the tree structure: `<html>` is at the root, then `<head>` and `<body>` are its children, etc.

The text inside elements forms *text nodes*, labelled as `#text`. A text node contains only a string. It may not have children and is always a leaf of the tree.

For instance, the `<title>` tag has the text `"About elk"`.

Please note the special characters in text nodes:

- a newline: `↵` (in JavaScript known as `\n`)
- a space: `␣`

Spaces and newlines are totally valid characters, like letters and digits. They form text nodes and become a part of the DOM. So, for instance, in the example above the `<head>` tag contains some spaces before `<title>`, and that text becomes a `#text` node (it contains a newline and some spaces only).

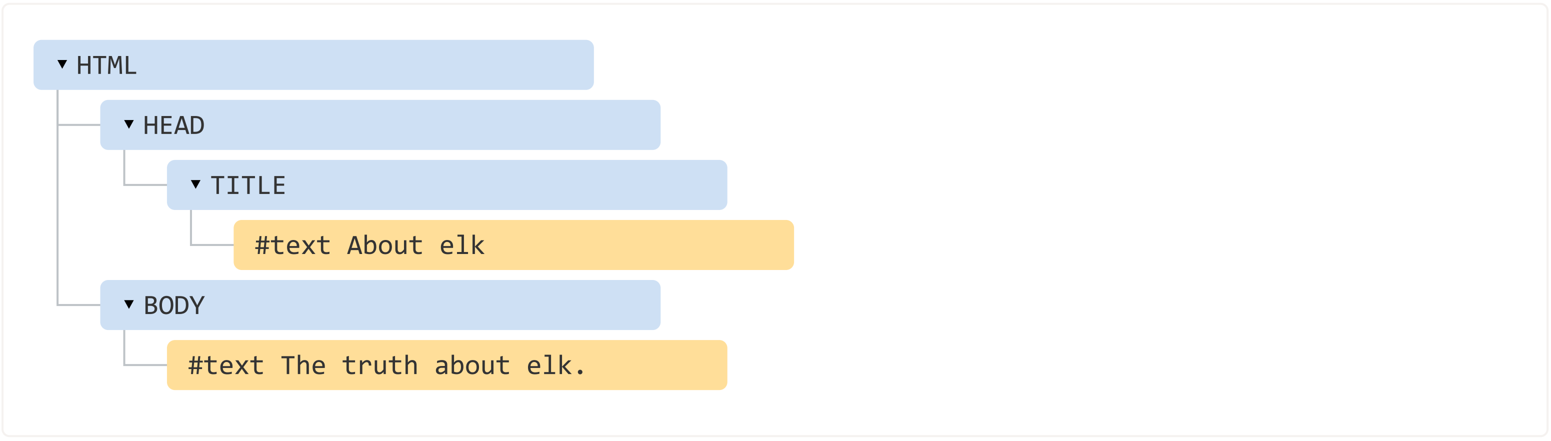
There are only two top-level exclusions:

1. Spaces and newlines before `<head>` are ignored for historical reasons.
2. If we put something after `</body>`, then that is automatically moved inside the `body`, at the end, as the HTML spec requires that all content must be inside `<body>`. So there can't be any spaces after `</body>`.

In other cases everything's straightforward – if there are spaces (just like any character) in the document, then they become text nodes in the DOM, and if we remove them, then there won't be any.

Here are no space-only text nodes:

```
<!DOCTYPE HTML>
<html><head><title>About elk</title></head><body>The truth about elk.</body></html>
```



Spaces at string start/end and space-only text nodes are usually hidden in tools

Browser tools (to be covered soon) that work with DOM usually do not show spaces at the start/end of the text and empty text nodes (line-breaks) between tags.

Developer tools save screen space this way.

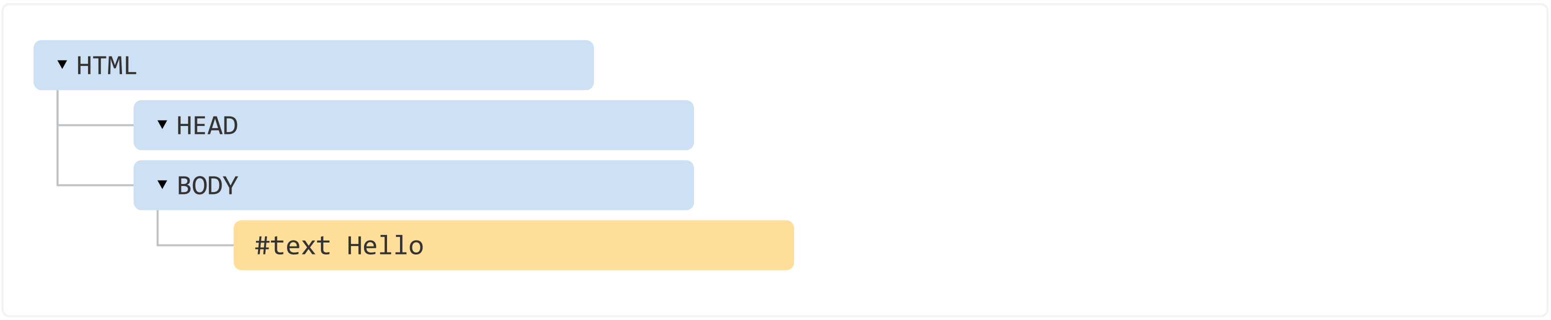
On further DOM pictures we'll sometimes omit them when they are irrelevant. Such spaces usually do not affect how the document is displayed.

Autocorrection

If the browser encounters malformed HTML, it automatically corrects it when making the DOM.

For instance, the top tag is always `<html>`. Even if it doesn't exist in the document, it will exist in the DOM, because the browser will create it. The same goes for `<body>`.

As an example, if the HTML file is the single word `"Hello"`, the browser will wrap it into `<html>` and `<body>`, and add the required `<head>`, and the DOM will be:

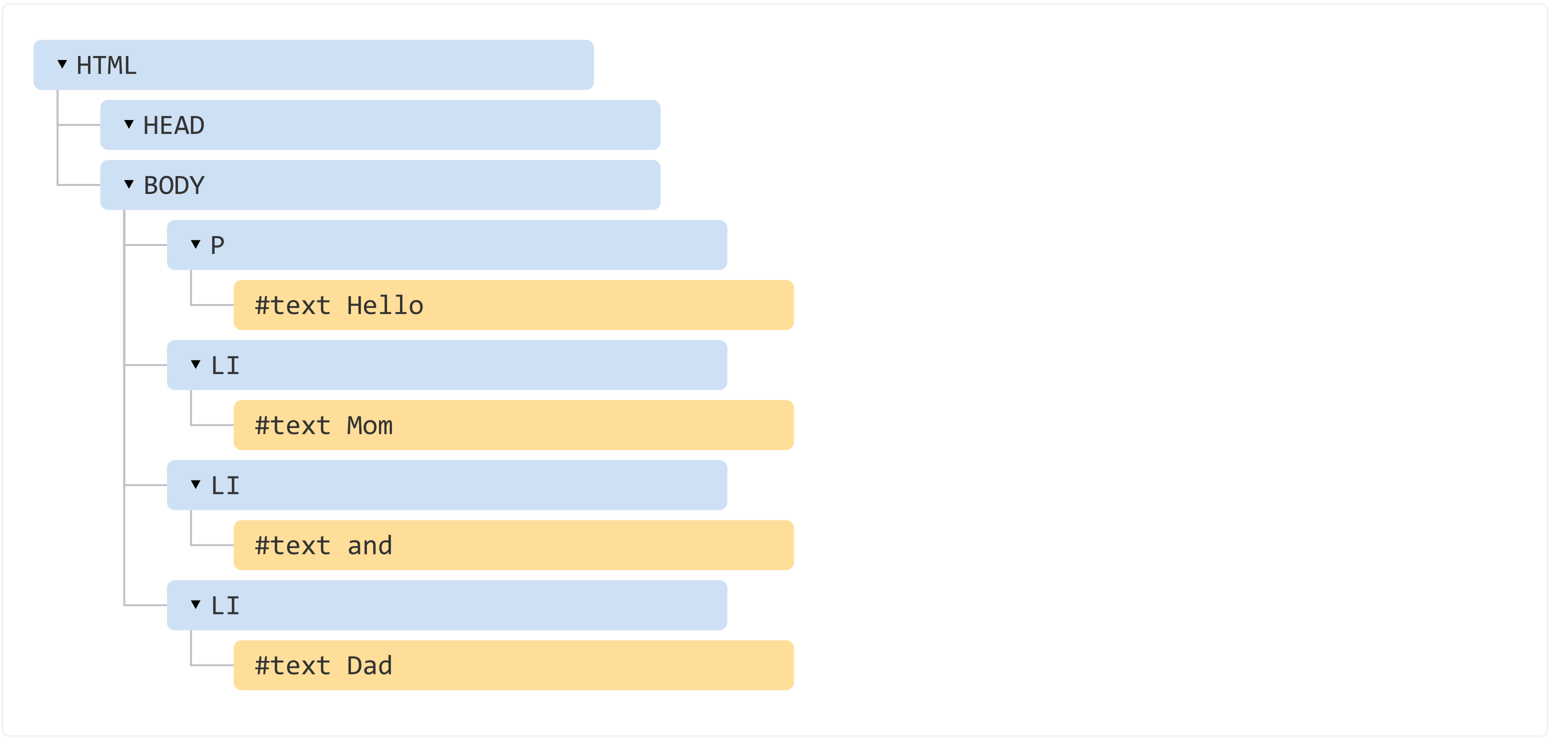


While generating the DOM, browsers automatically process errors in the document, close tags and so on.

A document with unclosed tags:

```
<p>Hello
<li>Mom
<li>and
<li>Dad
```

...will become a normal DOM as the browser reads tags and restores the missing parts:



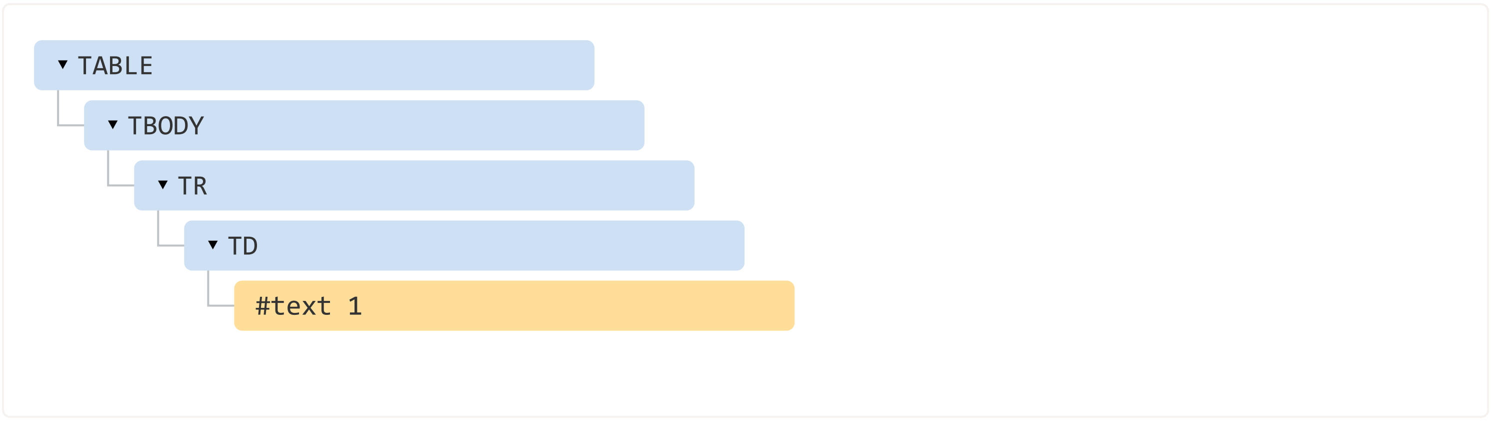
Tables always have `<tbody>`

An interesting "special case" is tables. By DOM specification they must have `<tbody>` tag, but HTML text may omit it. Then the browser creates `<tbody>` in the DOM automatically.

For the HTML:

```
<table id="table"><tr><td>1</td></tr></table>
```

DOM-structure will be:



You see? The `<tbody>` appeared out of nowhere. We should keep this in mind while working with tables to avoid surprises.

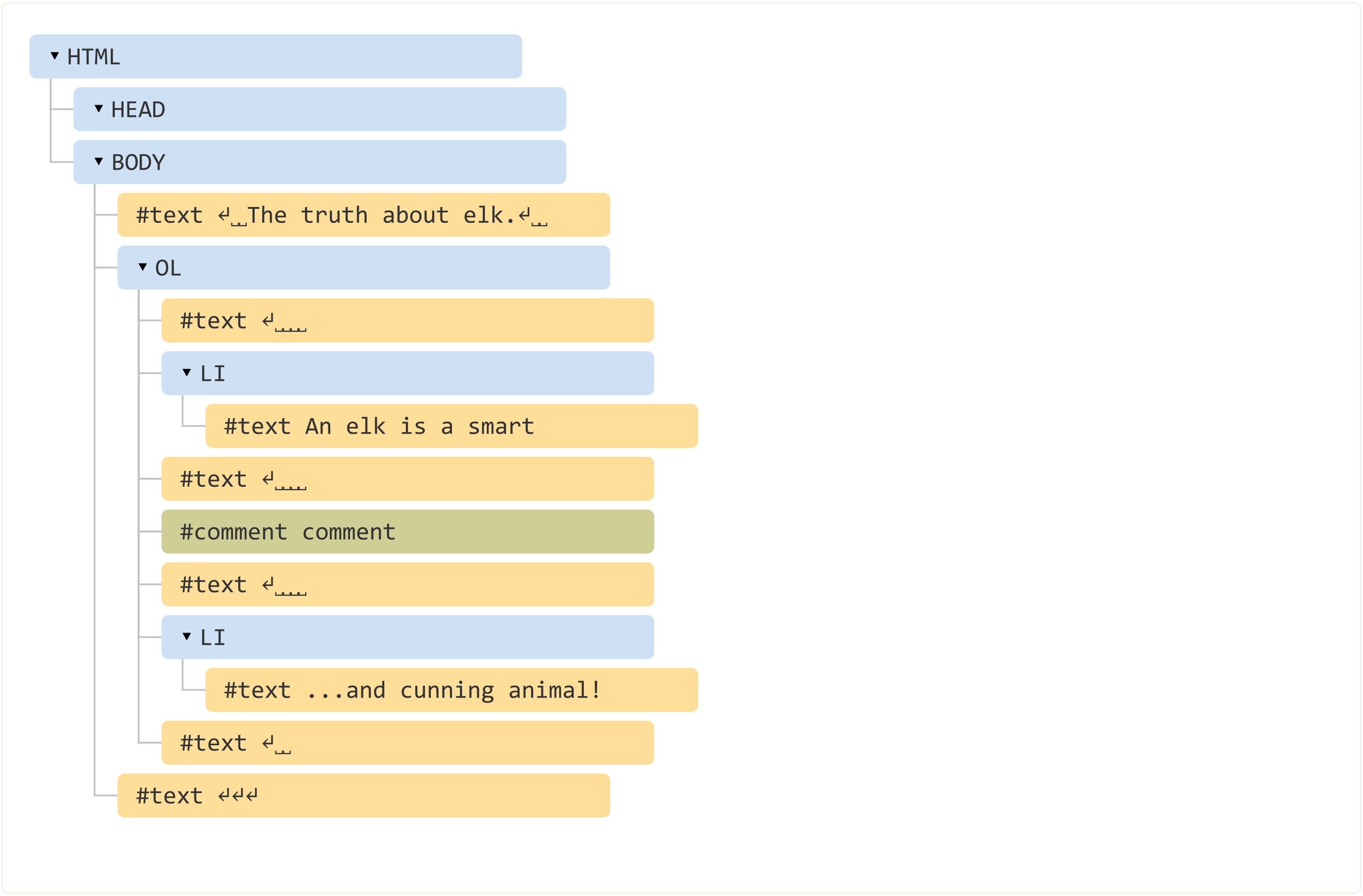
Other node types

There are some other node types besides elements and text nodes.

For example, comments:

```
<!DOCTYPE HTML>
```

```html
<html>
<body>
  The truth about elk.
  <ol>
    <li>An elk is a smart</li>
    <!-- comment -->
    <li>...and cunning animal!</li>
  </ol>
</body>
</html>
```



We can see here a new tree node type – *comment node*, labeled as `#comment`, between two text nodes.

We may think – why is a comment added to the DOM? It doesn't affect the visual representation in any way. But there's a rule – if something's in HTML, then it also must be in the DOM tree.

Everything in HTML, even comments, becomes a part of the DOM.

Even the `<!DOCTYPE...>` directive at the very beginning of HTML is also a DOM node. It's in the DOM tree right before `<html>`. Few people know about that. We are not going to touch that node, we even don't draw it on diagrams, but it's there.

The `document` object that represents the whole document is, formally, a DOM node as well.

There are 12 node types. In practice we usually work with 4 of them:

1. `document` – the "entry point" into DOM.
2. element nodes – HTML-tags, the tree building blocks.
3. text nodes – contain text.

4. comments – sometimes we can put information there, it won't be shown, but JS can read it from the DOM.

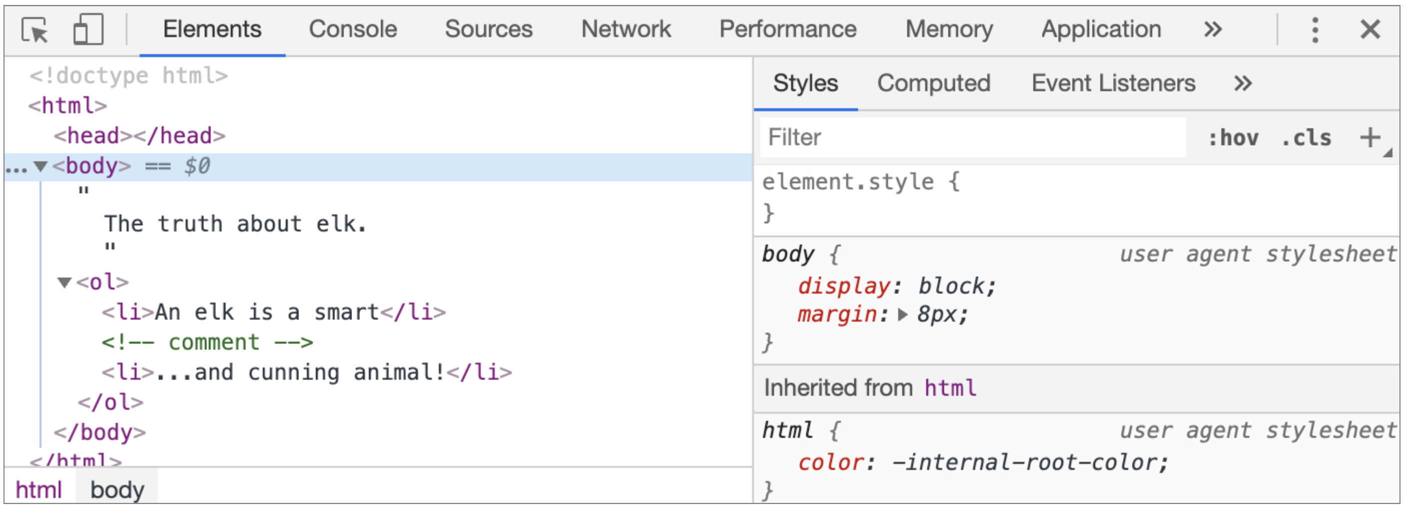
See it for yourself

To see the DOM structure in real-time, try Live DOM Viewer. Just type in the document, and it will show up as a DOM at an instant.

Another way to explore the DOM is to use the browser developer tools. Actually, that's what we use when developing.

To do so, open the web page elk.html, turn on the browser developer tools and switch to the Elements tab.

It should look like this:



```
<!doctype html>
<html>
  <head></head>
...▼<body> == $0
    "
    The truth about elk.
    "
  ▼<ol>
      <li>An elk is a smart</li>
      <!-- comment -->
      <li>...and cunning animal!</li>
    </ol>
  </body>
</html>
```

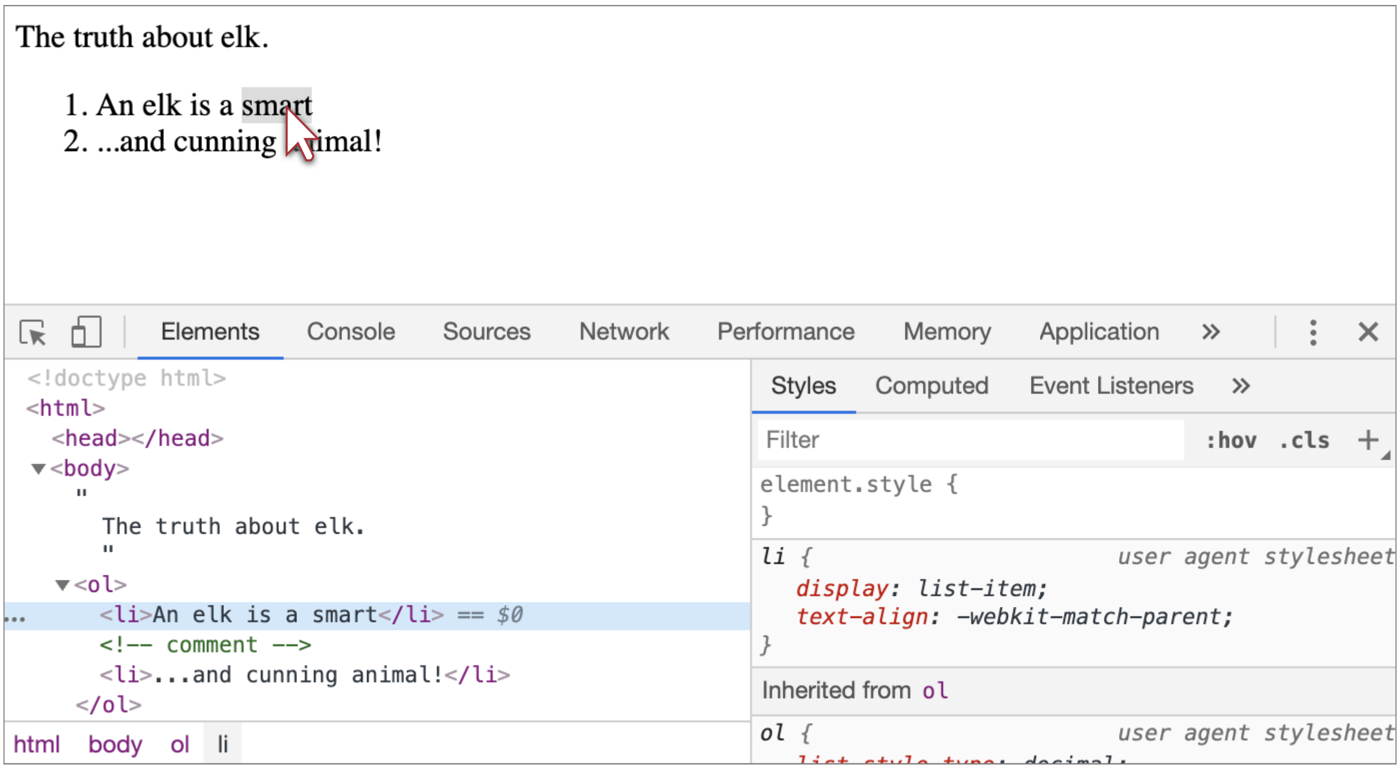
```
element.style {
}
body {                    user agent stylesheet
  display: block;
  margin: ▸ 8px;
}
html {                    user agent stylesheet
  color: -internal-root-color;
}
```

You can see the DOM, click on elements, see their details and so on.

Please note that the DOM structure in developer tools is simplified. Text nodes are shown just as text. And there are no "blank" (space only) text nodes at all. That's fine, because most of the time we are interested in element nodes.

Clicking the button in the left-upper corner allows us to choose a node from the webpage using a mouse (or other pointer devices) and "inspect" it (scroll to it in the Elements tab). This works great when we have a huge HTML page (and corresponding huge DOM) and would like to see the place of a particular element in it.

Another way to do it would be just right-clicking on a webpage and selecting "Inspect" in the context menu.



```
<!doctype html>
<html>
  <head></head>
 ▼<body>
    "
    The truth about elk.
    "
  ▼<ol>
...   <li>An elk is a smart</li> == $0
      <!-- comment -->
      <li>...and cunning animal!</li>
    </ol>
```

```
element.style {
}
li {                      user agent stylesheet
  display: list-item;
  text-align: -webkit-match-parent;
}
ol {                      user agent stylesheet
```

At the right part of the tools there are the following subtabs:

- **Styles** – we can see CSS applied to the current element rule by rule, including built-in rules (gray). Almost everything can be edited in-place, including the dimensions/margins/paddings of the box below.
- **Computed** – to see CSS applied to the element by property: for each property we can see a rule that gives it (including CSS inheritance and such).
- **Event Listeners** – to see event listeners attached to DOM elements (we'll cover them in the next part of the tutorial).
- …and so on.

The best way to study them is to click around. Most values are editable in-place.

Interaction with console

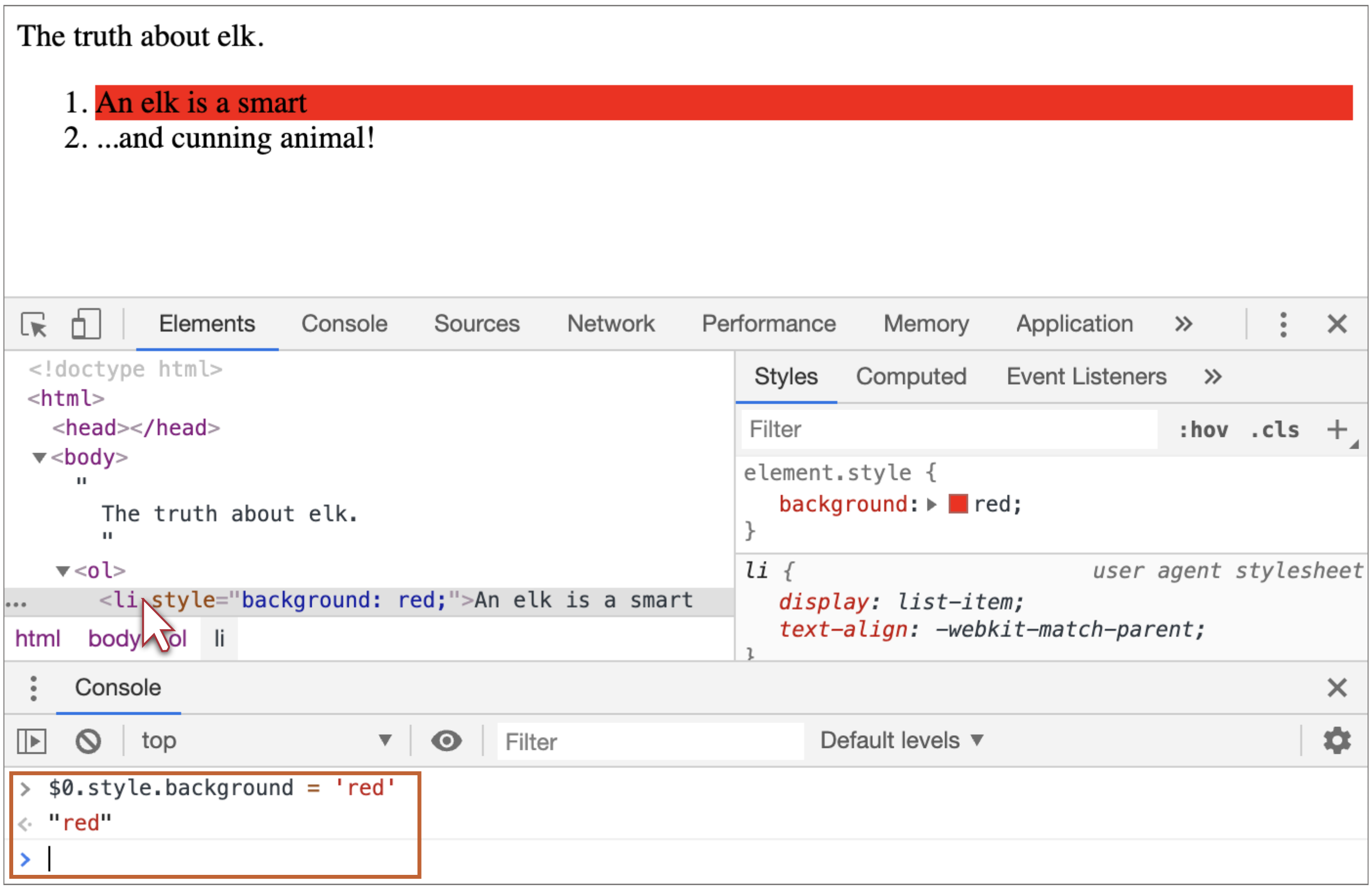
As we work the DOM, we also may want to apply JavaScript to it. Like: get a node and run some code to modify it, to see the result. Here are few tips to travel between the Elements tab and the console.

For the start:

1. Select the first `<li>` in the Elements tab.
2. Press `Esc` – it will open console right below the Elements tab.

Now the last selected element is available as `$0`, the previously selected is `$1` etc.

We can run commands on them. For instance, `$0.style.background = 'red'` makes the selected list item red, like this:


The truth about elk.

1. An elk is a smart
2. ...and cunning animal!

```
<!doctype html>
<html>
  <head></head>
  <body>
    "
      The truth about elk.
    "
    <ol>
      <li style="background: red;">An elk is a smart
```

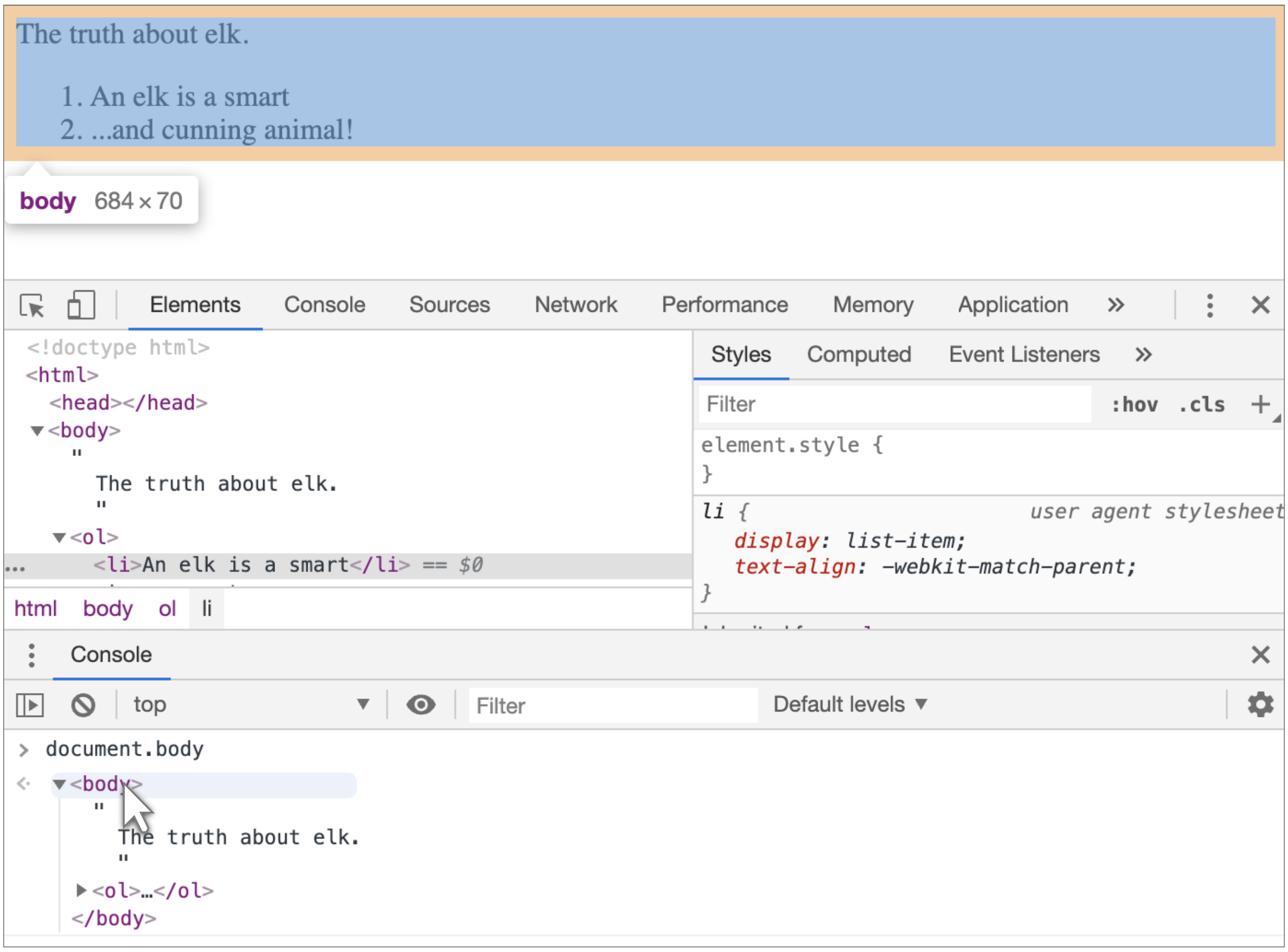
```
element.style {
    background: ■ red;
}
li {                          user agent stylesheet
    display: list-item;
    text-align: -webkit-match-parent;
}
```

```
> $0.style.background = 'red'
< "red"
> |
```

That's how to get a node from Elements in Console.

There's also a road back. If there's a variable referencing a DOM node, then we can use the command `inspect(node)` in Console to see it in the Elements pane.

Or we can just output the DOM node in the console and explore "in-place", like `document.body` below:



The truth about elk.

1. An elk is a smart
2. ...and cunning animal!

```
<!doctype html>
<html>
  <head></head>
 ▼<body>
     "
      The truth about elk.
      "
   ▼<ol>
...     <li>An elk is a smart</li> == $0
```

```
element.style {
}
li {                        user agent stylesheet
   display: list-item;
   text-align: -webkit-match-parent;
}
```

```
> document.body
<· ▼<body>
      "
       The truth about elk.
       "
    ▶<ol>…</ol>
    </body>
```

That's for debugging purposes of course. From the next chapter on we'll access and modify DOM using JavaScript.

The browser developer tools are a great help in development: we can explore the DOM, try things and see what goes wrong.

Summary

An HTML/XML document is represented inside the browser as the DOM tree.

- Tags become element nodes and form the structure.
- Text becomes text nodes.
- …etc, everything in HTML has its place in DOM, even comments.

We can use developer tools to inspect DOM and modify it manually.

Here we covered the basics, the most used and important actions to start with. There's an extensive documentation about Chrome Developer Tools at https://developers.google.com/web/tools/chrome-devtools. The best way to learn the tools is to click here and there, read menus: most options are obvious. Later, when you know them in general, read the docs and pick up the rest.

DOM nodes have properties and methods that allow us to travel between them, modify them, move around the page, and more. We'll get down to them in the next chapters.