

Основы Веба

Представление данных в вебе.

Со временем количество и разнообразие данных, доступных в интернете, значительно выросло. Для эффективной обработки и представления этой информации были созданы различные технологии, позволяющие организовать данные в удобный для пользователя формат. Одной из таких технологий является представление данных в вебе.

Представление данных в вебе означает организацию информации в виде веб-страниц, которые могут быть просмотрены с помощью браузера. Для этого данные должны быть структурированы и представлены в определенном формате, который может быть легко интерпретирован и отображен на экране.

Для представления данных в вебе используются различные языки программирования и технологии. Одним из основных языков является HTML (HyperText Markup Language), который используется для создания структуры страницы. Другим важным языком является CSS (Cascading Style Sheets), который используется для задания стилей и оформления страницы.

Для представления данных в вебе также используются различные базы данных, такие как MySQL, PostgreSQL и MongoDB. Базы данных хранят информацию в определенном формате, который может быть легко извлечена и представлена в веб-странице.

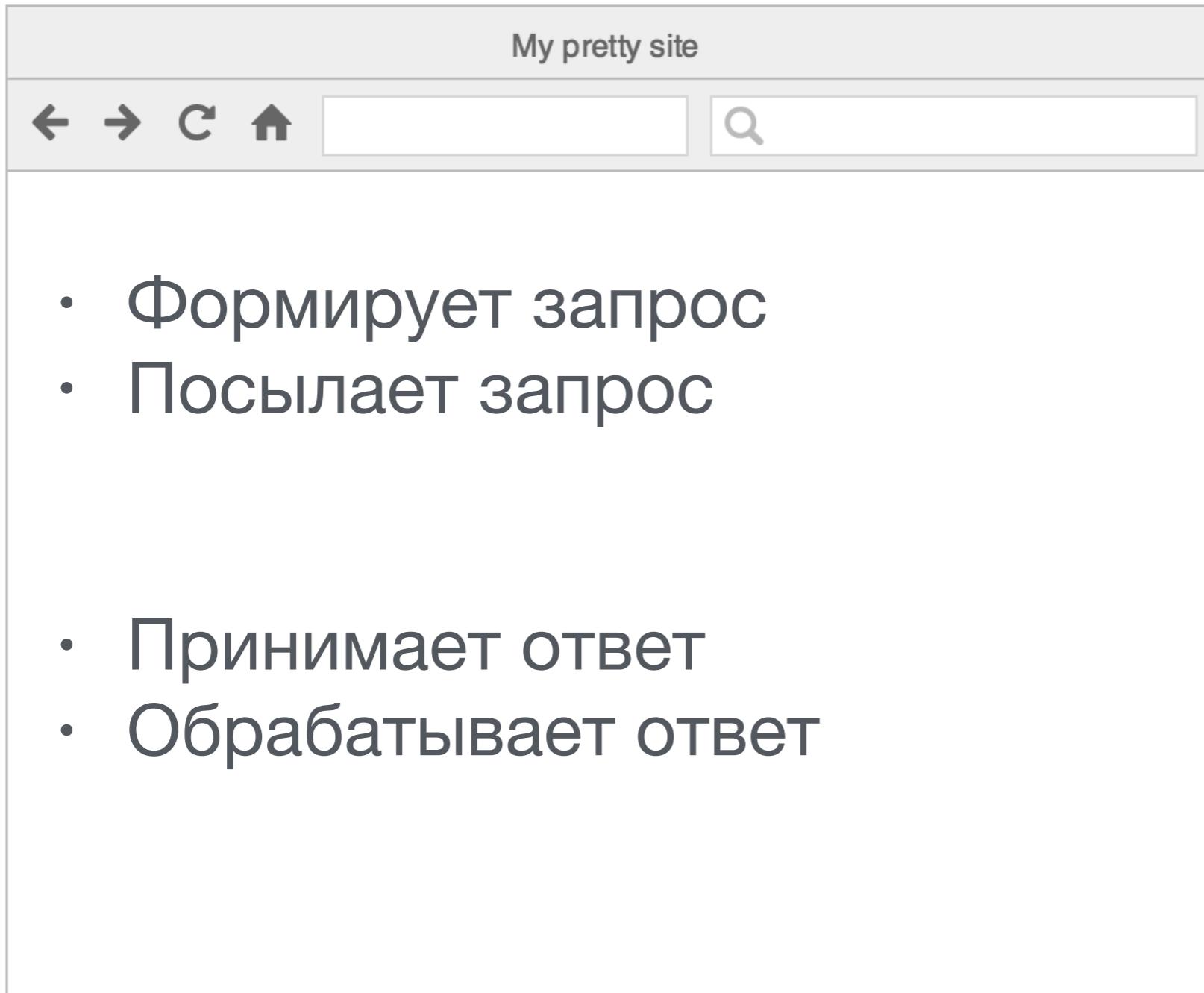
Помимо языков программирования и баз данных, для представления данных в вебе используются различные инструменты и технологии, такие как JavaScript, Python, Java и .NET. Эти инструменты позволяют создавать более сложные и互動ные веб-страницы, которые могут взаимодействовать с пользователем.

Представление данных в вебе имеет множество преимуществ. Во-первых, оно позволяет легко организовать и представить информацию, которая может быть доступна для миллионов пользователей по всему миру. Во-вторых, оно позволяет легко обновлять и изменять информацию, не требуя физической модификации страницы. В-третьих, оно позволяет легко интегрировать различные данные и сервисы, что делает веб-страницы более функциональными и полезными для пользователя.

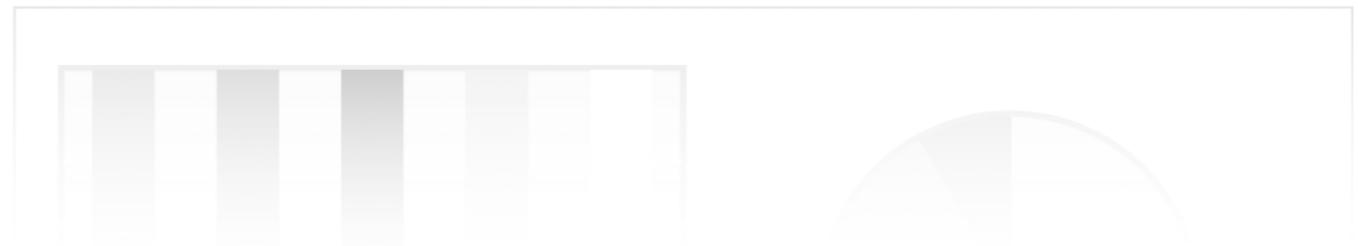
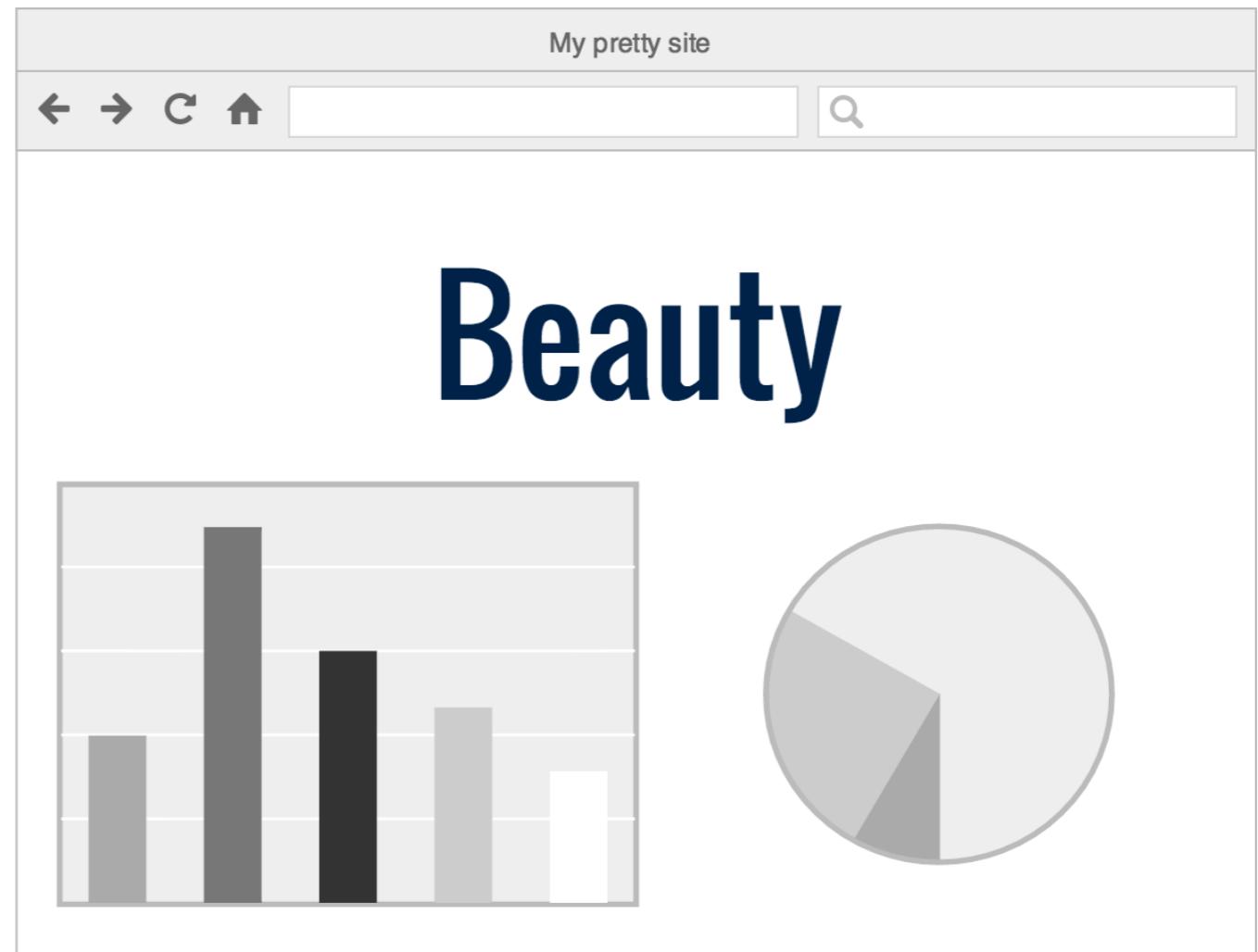
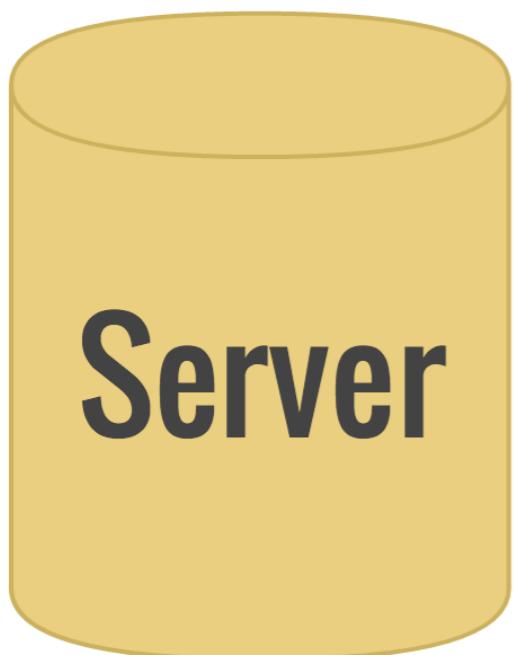
Однако, представление данных в вебе также имеет некоторые недостатки. Одним из них является то, что данные должны быть структурированы и организованы в определенном формате, что может быть сложно для некоторых типов данных. Другим недостатком является то, что представление данных в вебе может быть медленным и неэффективным, особенно при работе с большими объемами данных.

В целом, представление данных в вебе является важной технологией, которая позволяет легко организовать и представить информацию, доступную в интернете. Это делает веб-страницы более удобными и полезными для пользователя, а также позволяет легко интегрировать различные данные и сервисы.

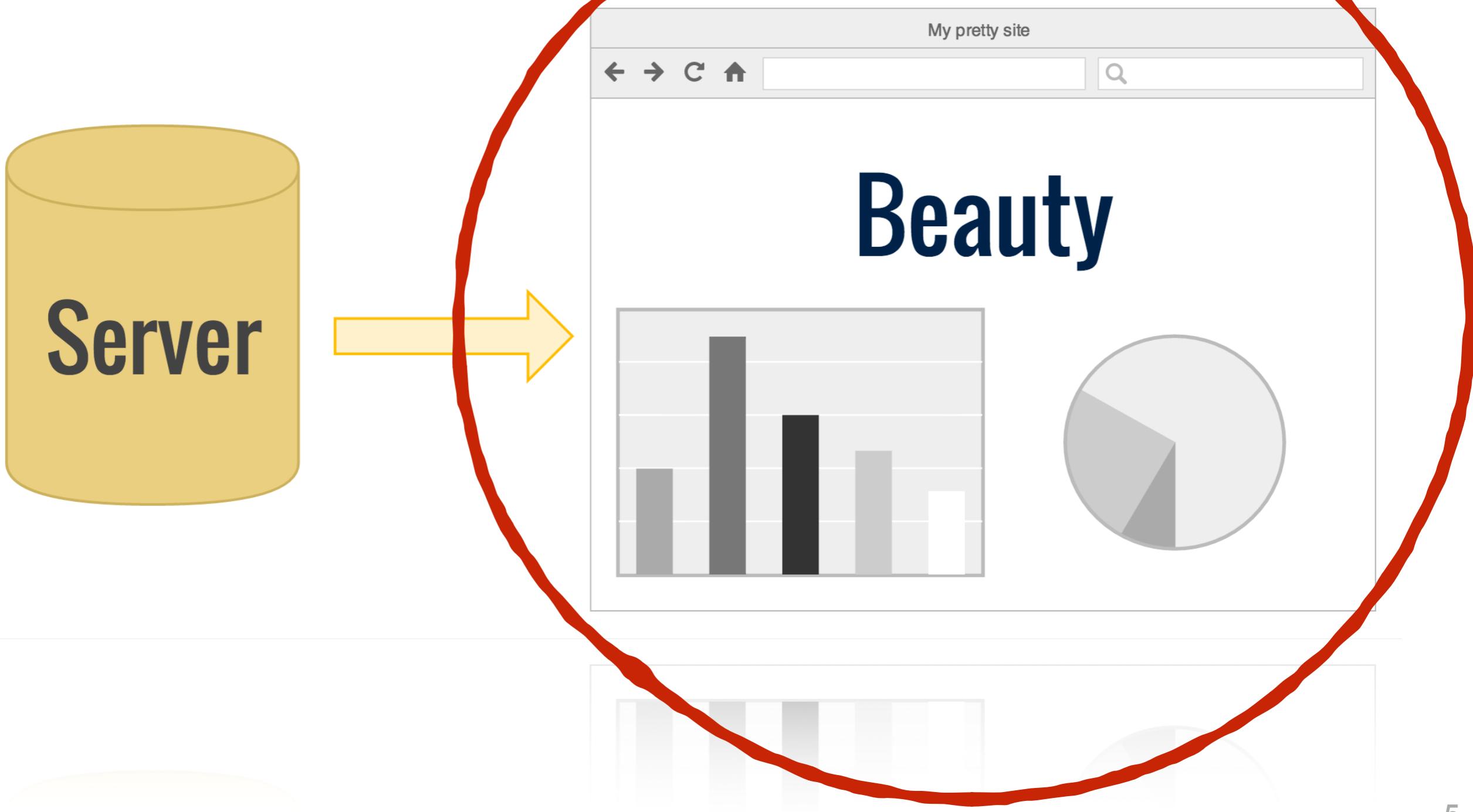
Как работает браузер



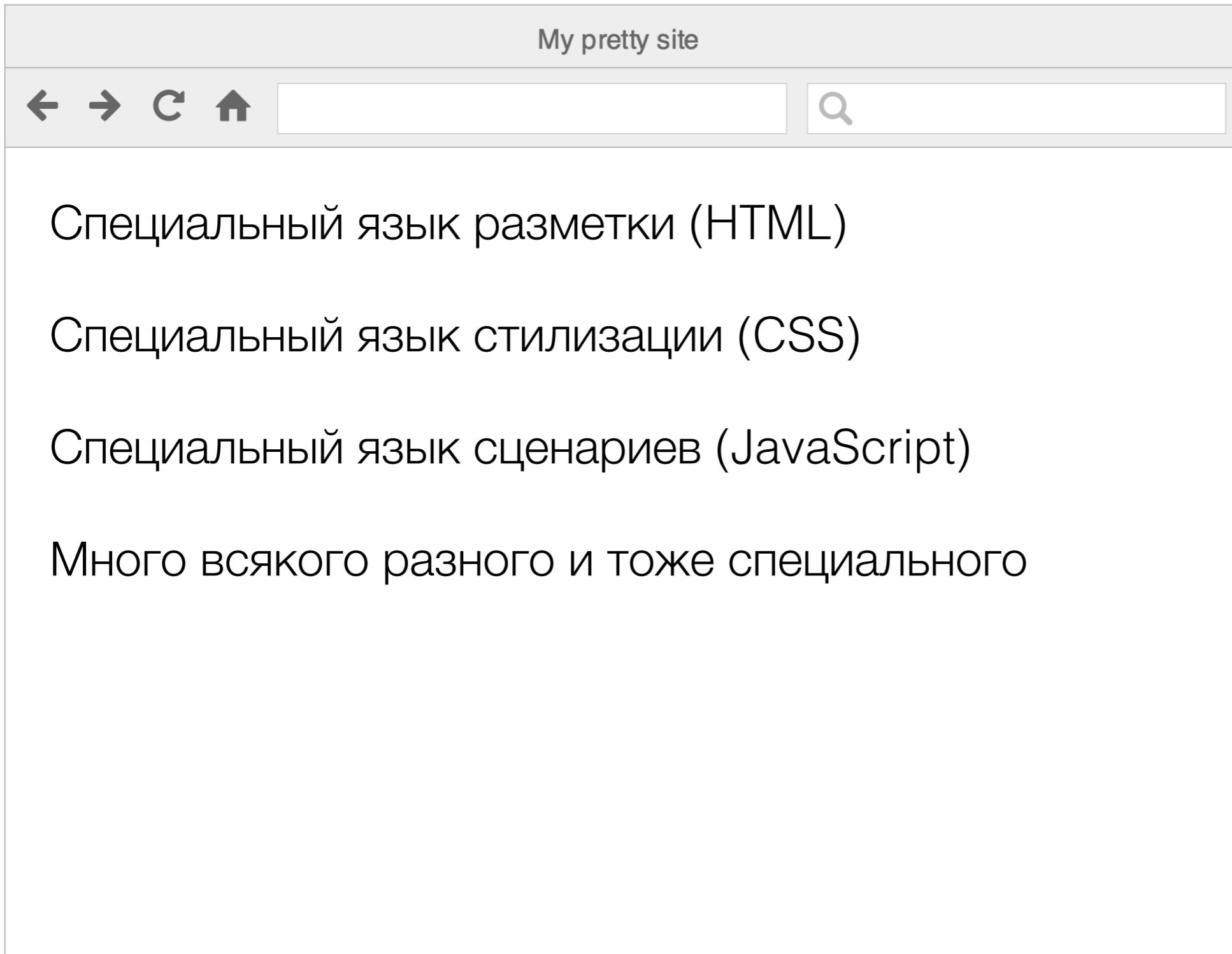
Как работает браузер



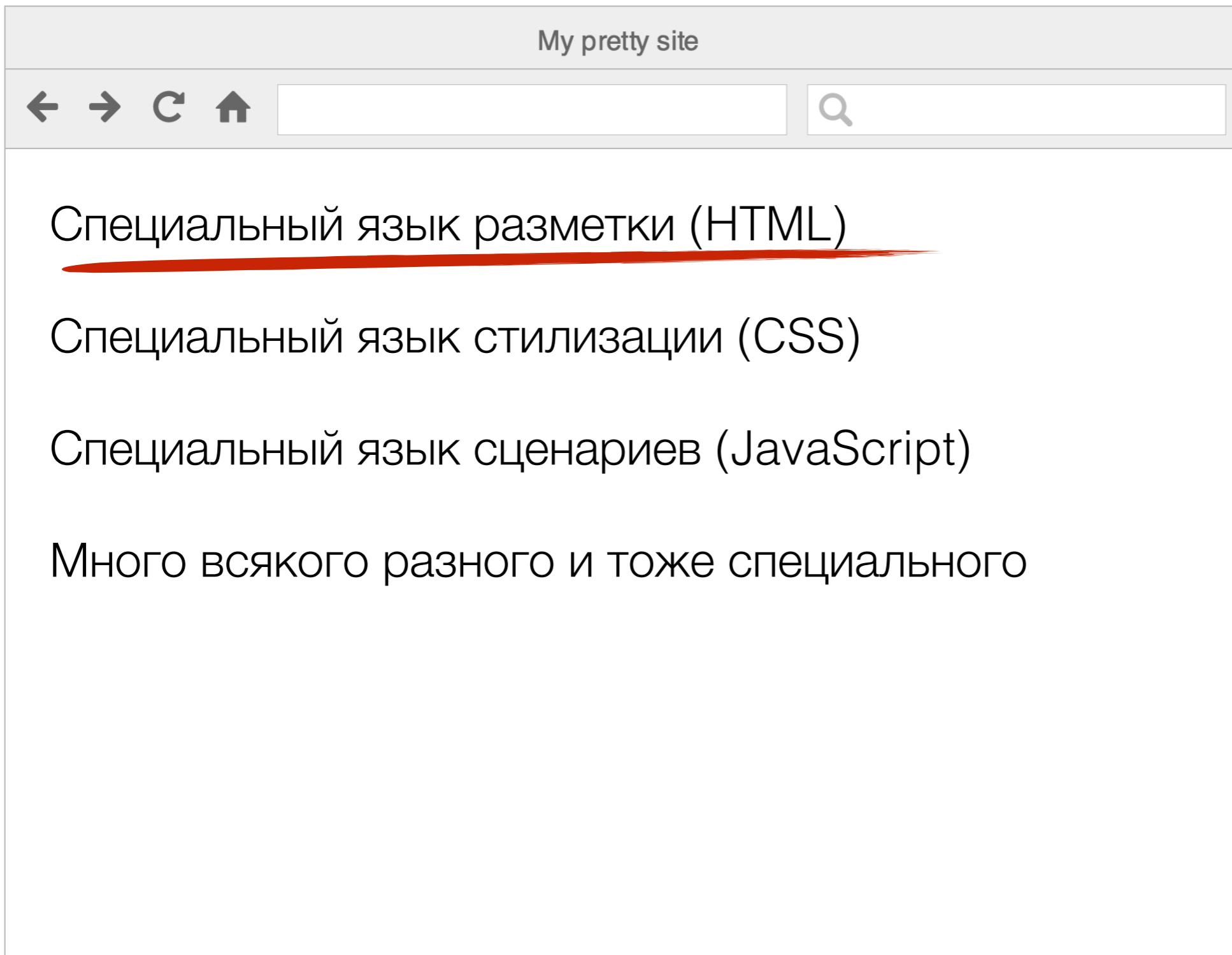
Как работает браузер



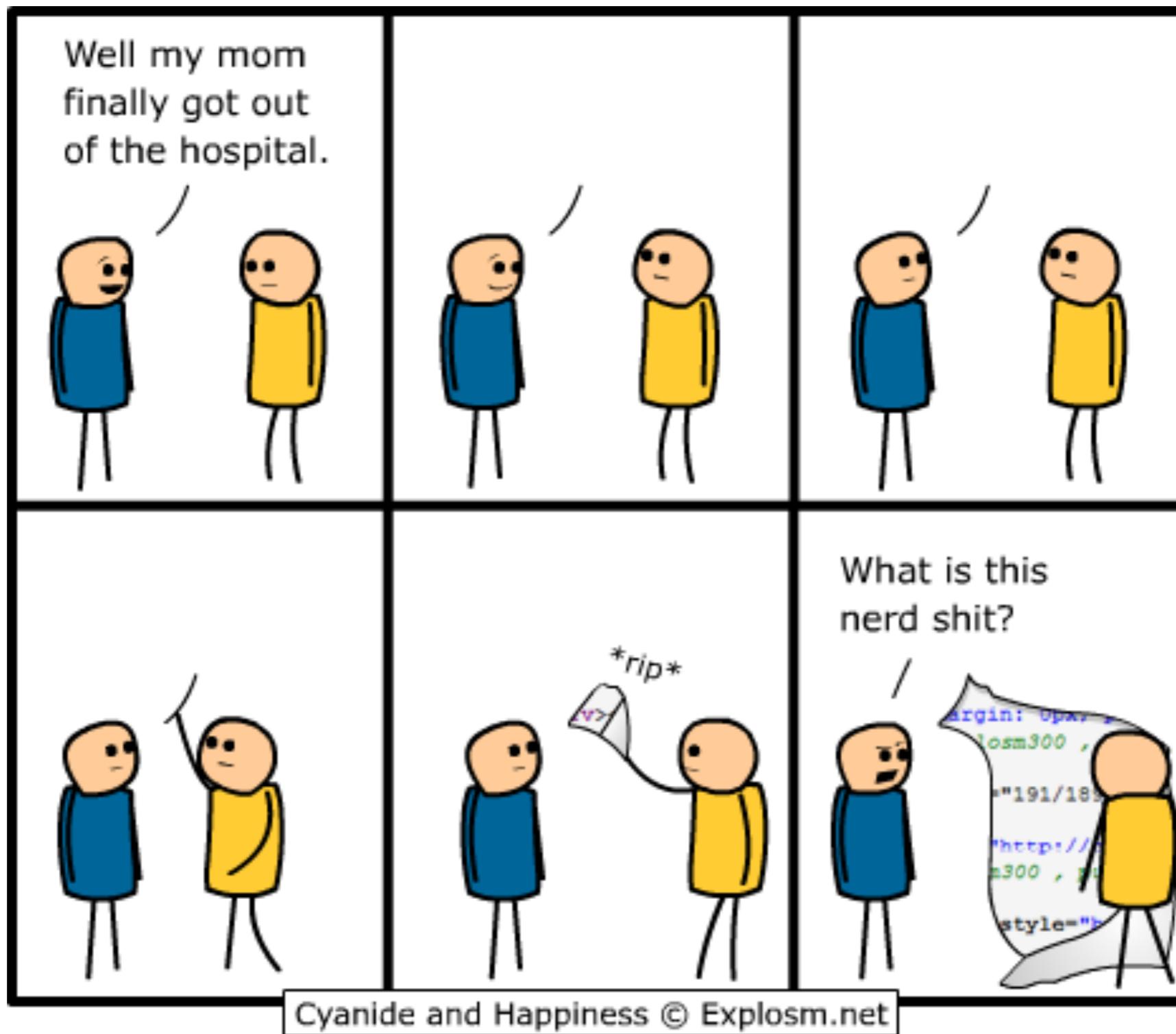
Как работает браузер



Как работает браузер



HTML



вообще-то

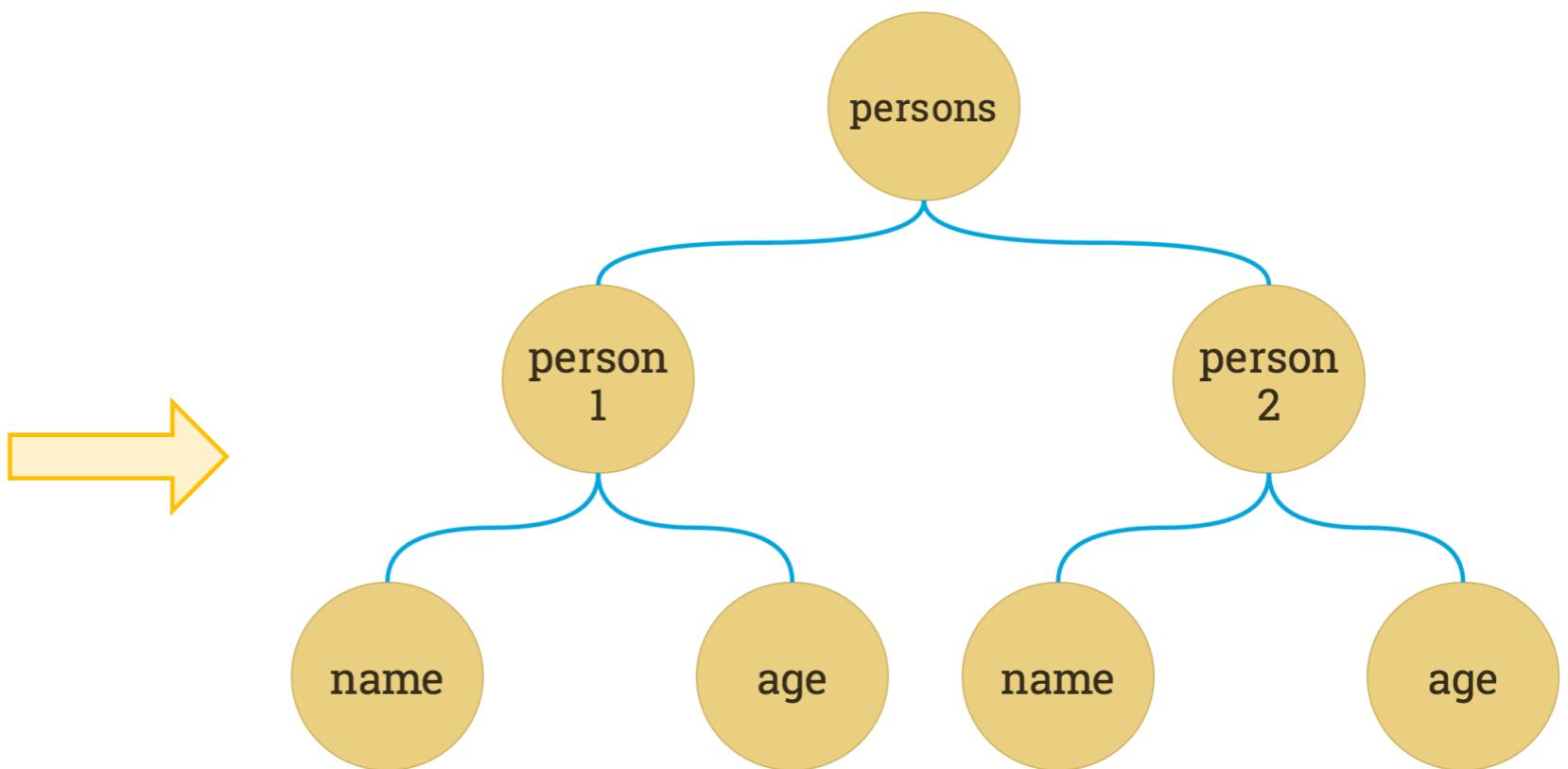
HTML ∈ XML

XML

extensible markup language

```
<persons>
  <person id="1">
    <name>John</name>
    <age>26</age>
  </person>

  <person id="2">
    <name>Jane</name>
    <age>24</age>
  </person>
</persons>
```

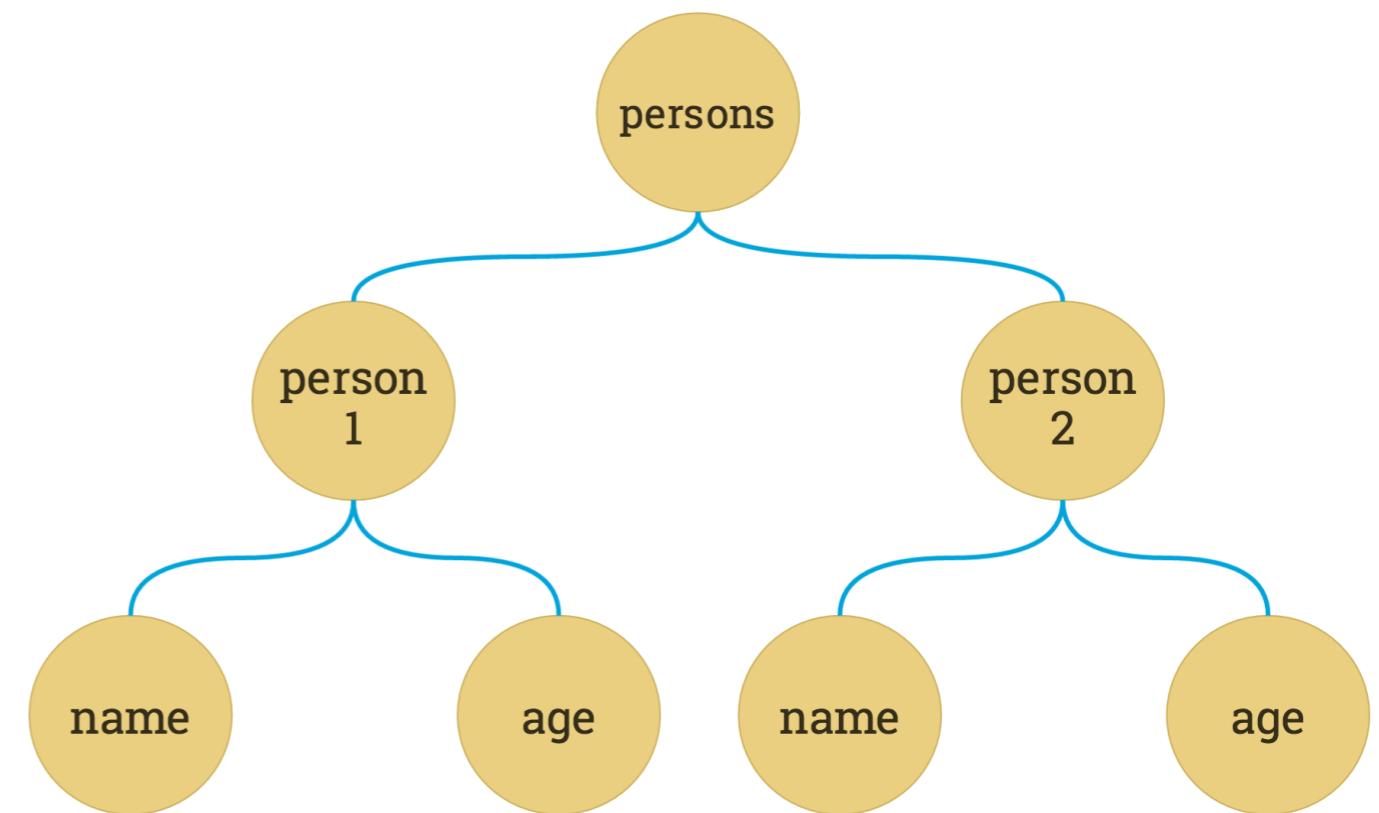


XML

extensible markup language

```
<persons>
  <person id="1">
    <name>John</name>
    <age>26</age>
  </person>

  <person id="2">
    <name>Jane</name>
    <age>24</age>
  </person>
</persons>
```



Синтаксис XML в общем случае

```
01: <node attribute="value of attribute">  
02:  
03:   <text-node>  
04:     Body of text-node  
05:   </text-node>  
06:  
07: </node>
```

HTML

HyperText Markup Language

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Привет, Moscow Coding School! </title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1> Заголовок </h2>
    
  </body>
</html>
```

HTML

HyperText Markup Language

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Привет, Moscow Coding School! </title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1> Заголовок </h2>
    
  </body>
</html>
```

Тэг, элемент, узел, node

HTML

HyperText Markup Language

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Привет, Moscow Coding School! </title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1> Заголовок </h2>
    
  </body>
</html>
```

Тэг, element, узел, node

HTML

HyperText Markup Language

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Привет, Moscow Coding School! </title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1> Заголовок </h2>
    
  </body>
</html>
```

Тэг, element, узел, node

HTML

HyperText Markup Language

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Привет, Moscow Coding School! </title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1> Заголовок </h2>
    
  </body>
</html>
```

Тэг, element, узел, node

HTML

HyperText Markup Language

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Привет, Moscow Coding School! </title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1> Заголовок </h2>
    
  </body>
</html>
```

Атрибут (имя)

HTML

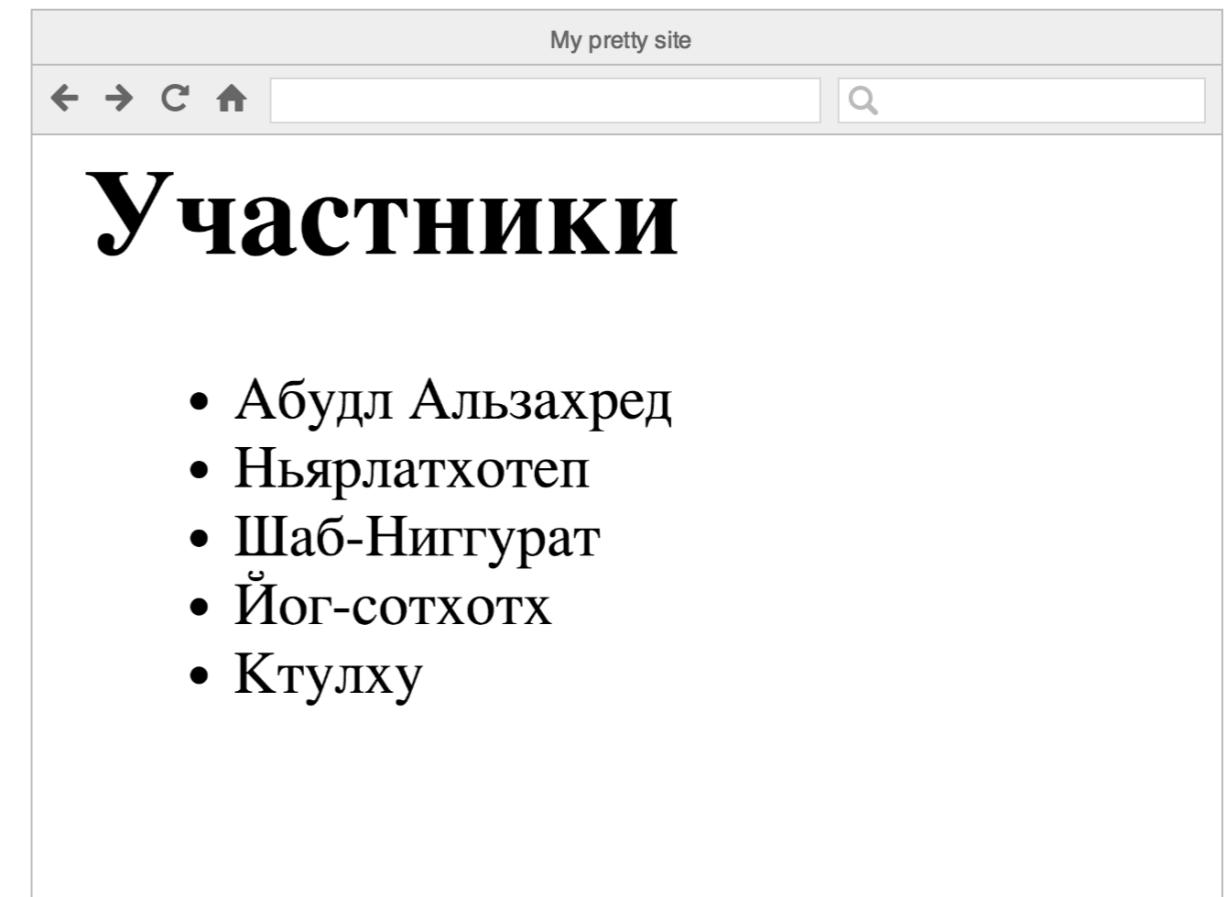
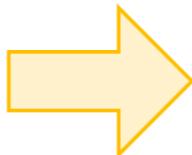
HyperText Markup Language

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Привет, Moscow Coding School! </title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1> Заголовок </h2>
    
  </body>
</html>
```

Атрибут (значение)

HTML - это иерархия документа

```
<h1>Участники</h1>
<ul>
  <li>Абдул Альзахред</li>
  <li>Нъярлатхотеп</li>
  <li>Шаб-Ниггурат</li>
  <li>Йог-сотхотх</li>
  <li>Ктулху</li>
</ul>
```



Объектная модель документа

```
01: <!DOCTYPE html>
02: <html>
03:   <head>
04:     <title> Привет, Moscow Coding School! </title>
06:   </head>
07:   <body>
08:     <h1>Участники</h2>
09:     <ul>
10:       <li>Абдул Альзахред</li>
11:       <li>Нъярлатхотеп</li>
12:       <li>Шаб-Ниггурат</li>
13:       <li>Йог-сотхотх</li>
14:       <li>Ктулху</li>
15:     </ul>
16:   </body>
17: </html>
```

Объектная модель документа

Участники

- Абдул Альзахред
- Ньярлатхотеп
- Шаб-Ниггурат
- Йог-сотхотх
- Ктулху

The screenshot shows the 'Elements' tab of a browser's developer tools. The DOM tree is displayed, starting with the document type declaration <!DOCTYPE html>. Below it is the <html> element, which contains the <head> and <body> elements. The <body> element contains an <h1> element with the text 'Участники'. This <h1> element is highlighted with a blue selection bar. Below it is a element, which contains five elements, each listing one of the participants from the bullet list. The element is also highlighted with a blue selection bar. At the bottom of the developer tools interface, there is a navigation bar with tabs labeled 'html', 'body', and 'h1', where 'h1' is currently selected.

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <h1>Участники</h1>
    <ul>
      <li>Абдул Альзахред</li>
      <li>Ньярлатхотеп</li>
      <li>Шаб-Ниггурат</li>
      <li>Йог-сотхотх</li>
      <li>Ктулху</li>
    </ul>
  </body>
</html>
```

```
01: <!DOCTYPE html>
02: <html>
03:   <head>
04:     <title> Привет, Moscow Coding School! </title>
06:   </head>
07:   <body>
08:     <h1 class="heading main-heading">Участники</h2>
09:     <ul id="main-list">
10:       <li>Абдул Альзахред</li>
11:       <li>Нъярлатхотеп</li>
12:       <li>Шаб-Ниггурат</li>
13:       <li>Йог-сотхотх</li>
14:       <li>Ктулху</li>
15:     </ul>
16:   </body>
17: </html>
```

```
01: <!DOCTYPE html>
02: <html>
03:   <head>
04:     <title> Привет, Moscow Coding School! </title>
06:   </head>
07:   <body>
08:     <h1 class="heading main-heading">Участники</h2>
09:     <ul id="main-list">
10:       <li>Абдул Альзахред</li>
11:       <li>Нъярлатхотеп</li>
12:       <li>Шаб-Ниггурат</li>
13:       <li>Йог-сотхотх</li>
14:       <li>Ктулху</li>
15:     </ul>
16:   </body>
17: </html>
```

The screenshot shows the DOM tree of a web page. At the top, there is a toolbar with icons for selecting elements, zooming, and other developer tools features. Below the toolbar, the main interface displays the following HTML structure:

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body style="height: 619px;">
    <h1 class="heading main-heading">Участники</h1>
    <ul id="main-list">
      <li>Абдул Альзахред</li>
      <li>Нъярлатхотеп</li>
      <li>Шаб-Ниггурат</li>
      <li>Йог-сотхотх</li>
      <li>Ктулху</li>
    </ul>
  </body>
</html>
```

The element `<h1 class="heading main-heading">Участники</h1>` is currently selected, highlighted with a blue background. In the bottom navigation bar, the path `html body h1.heading.main-heading` is shown, with the last part `h1.heading.main-heading` also highlighted in blue.

Визуализация

только HTML, только хардкор

Table Heading Cell	Table Heading Cell
Table Body Cell	Table Body Cell
Table Body Cell	Table Body Cell

```
01: <table>
02:   <thead>
03:     <tr>
04:       <th>Table Heading Cell</th>
05:       <th>Table Heading Cell</th>
06:     </tr>
07:   </thead>
08:   <tbody>
09:     <tr>
10:       <td>Table Body Cell</td>
11:       <td>Table Body Cell</td>
12:     </tr>
13:     <tr>
14:       <td>Table Body Cell</td>
15:       <td>Table Body Cell</td>
16:     </tr>
17:   </tbody>
18: </table>
```

<table>

Table Heading Cell	Table Heading Cell
Table Body Cell	Table Body Cell
Table Body Cell	Table Body Cell

<thead>

The diagram illustrates a table structure with the following characteristics:

- Header Row:** The first row is highlighted with a large red rounded rectangle spanning the width of the table. It contains two **Table Heading Cells**, both labeled "Table Heading Cell".
- Body Rows:** There are three additional rows below the header. The second row contains two **Table Body Cells**, both labeled "Table Body Cell". The third row also contains two **Table Body Cells**, both labeled "Table Body Cell".
- Cell Content:** All cells contain the text "Table Body Cell" or "Table Heading Cell" centered within them.
- Table Structure:** The table is defined by a dark blue border around the entire structure, and internal grid lines divide it into four columns and four rows.

Table Heading Cell	Table Heading Cell
Table Body Cell	Table Body Cell
Table Body Cell	Table Body Cell

<tbody>

Table Heading Cell	Table Heading Cell	<tr>
Table Body Cell	Table Body Cell	<tr>
Table Body Cell	Table Body Cell	<tr>

<p>Table Heading Cell</p> <p><th></p>	<p>Table Heading Cell</p> <p><th></p>
<p>Table Body Cell</p> <p><td></p>	<p>Table Body Cell</p> <p><td></p>
<p>Table Body Cell</p> <p><td></p>	<p>Table Body Cell</p> <p><td></p>

Table Heading Cell	Table Heading Cell
Table Body Cell	Table Body Cell
Table Body Cell	Table Body Cell

```
01: <table border="1" cellpadding="10">
02:   <thead>
03:     <tr>
04:       <th>Table Heading Cell</th>
05:       <th>Table Heading Cell</th>
06:     </tr>
07:   </thead>
08:   <tbody>
09:     <tr>
10:       <td>Table Body Cell</td>
11:       <td>Table Body Cell</td>
12:     </tr>
13:     <tr>
14:       <td>Table Body Cell</td>
15:       <td>Table Body Cell</td>
16:     </tr>
17:   </tbody>
18: </table>
```

Table Heading Cell	Table Heading Cell
Table Body Cell	Table Body Cell
Table Body Cell	Table Body Cell

SVG

вообще-то

SVG ∈ XML

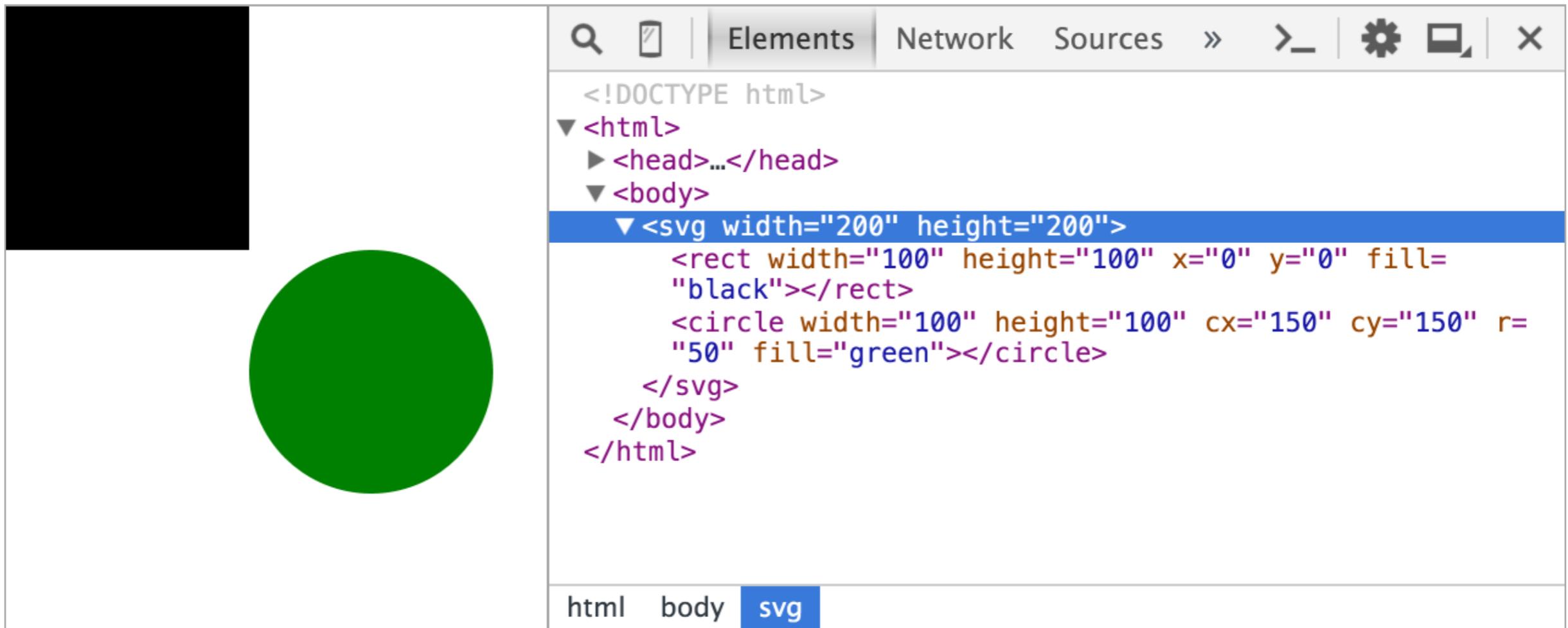
SVG

Scalable Vector Graphic

```
01: <svg width="200" height="200">
02:   <rect width="100" height="100"
03:     x="0" y="0" fill="black" />
04:
05:   <circle cx="150" cy="150" r="50" fill="green" />
06: </svg>
```

SVG

Scalable Vector Graphic

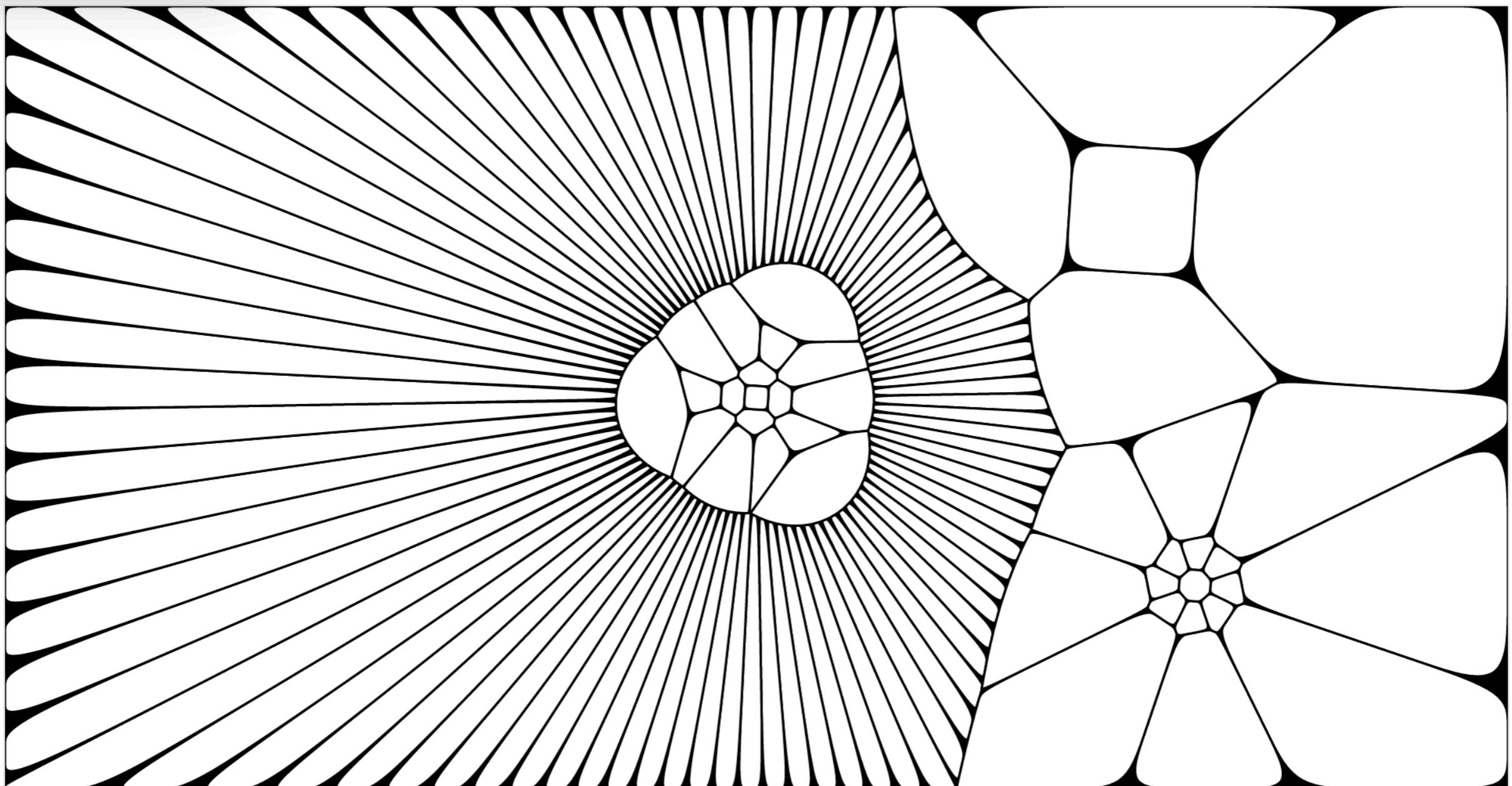


```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <svg width="200" height="200">
      <rect width="100" height="100" x="0" y="0" fill="black"></rect>
      <circle width="100" height="100" cx="150" cy="150" r="50" fill="green"></circle>
    </svg>
  </body>
</html>
```

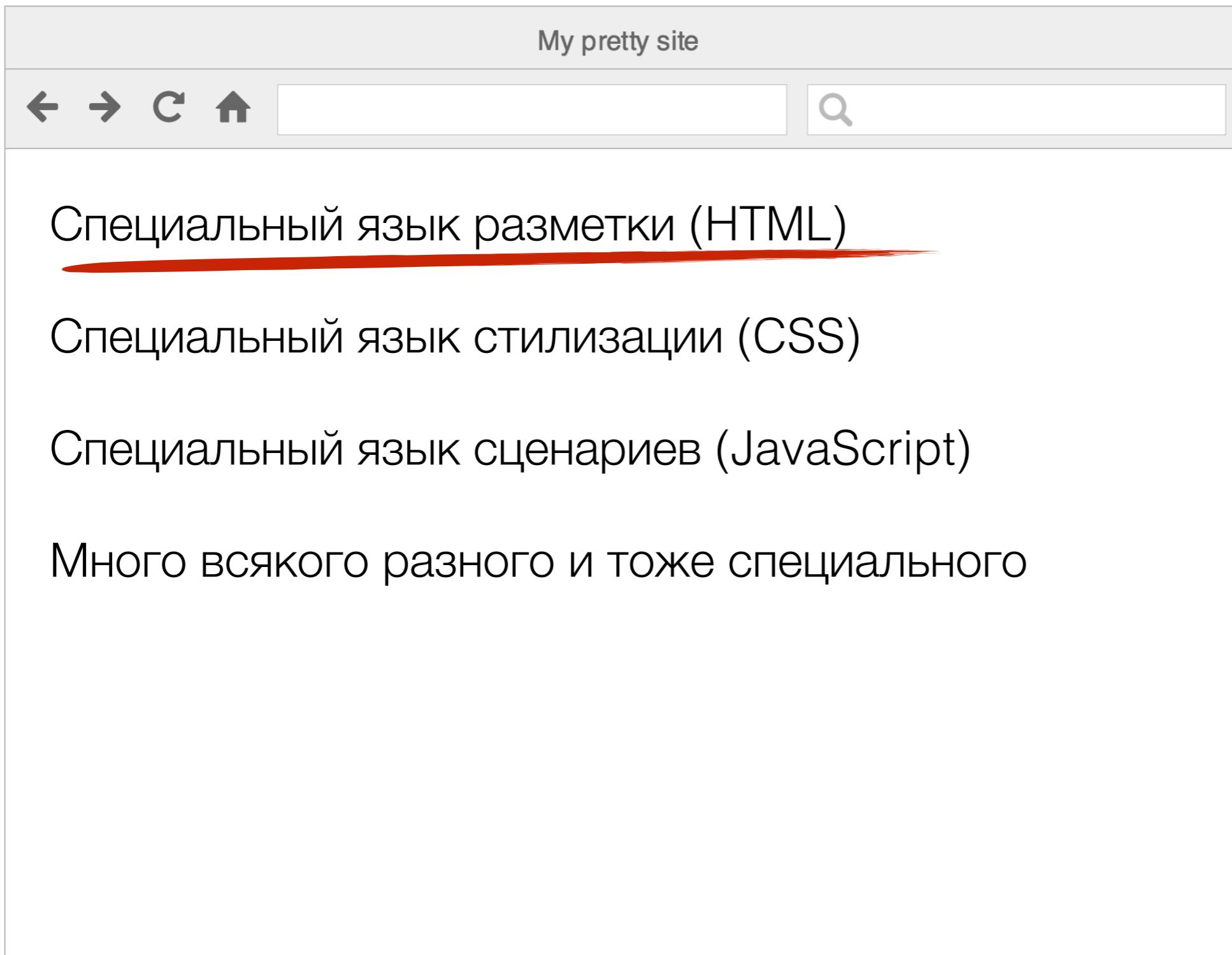
html body **svg**

SVG

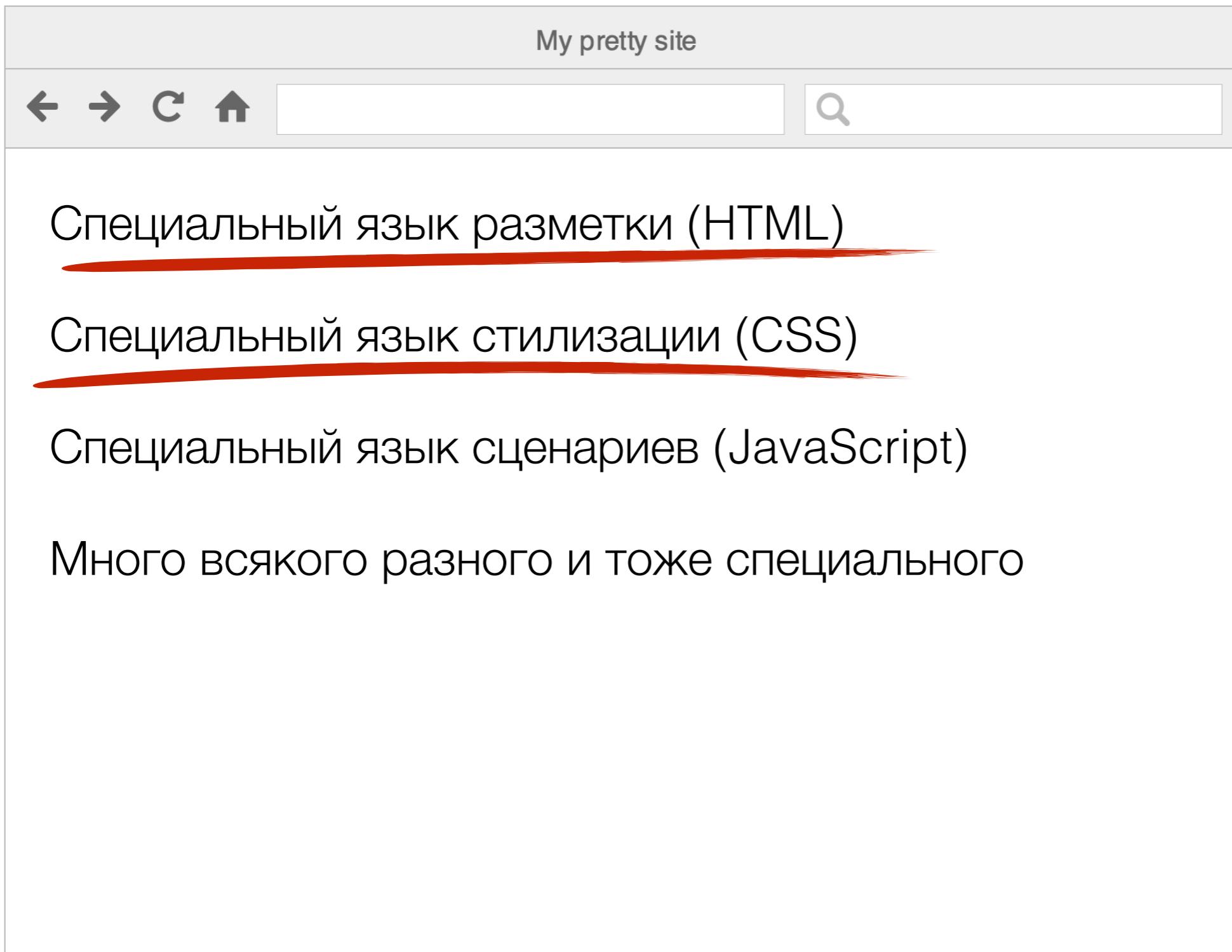
Scalable Vector Graphic

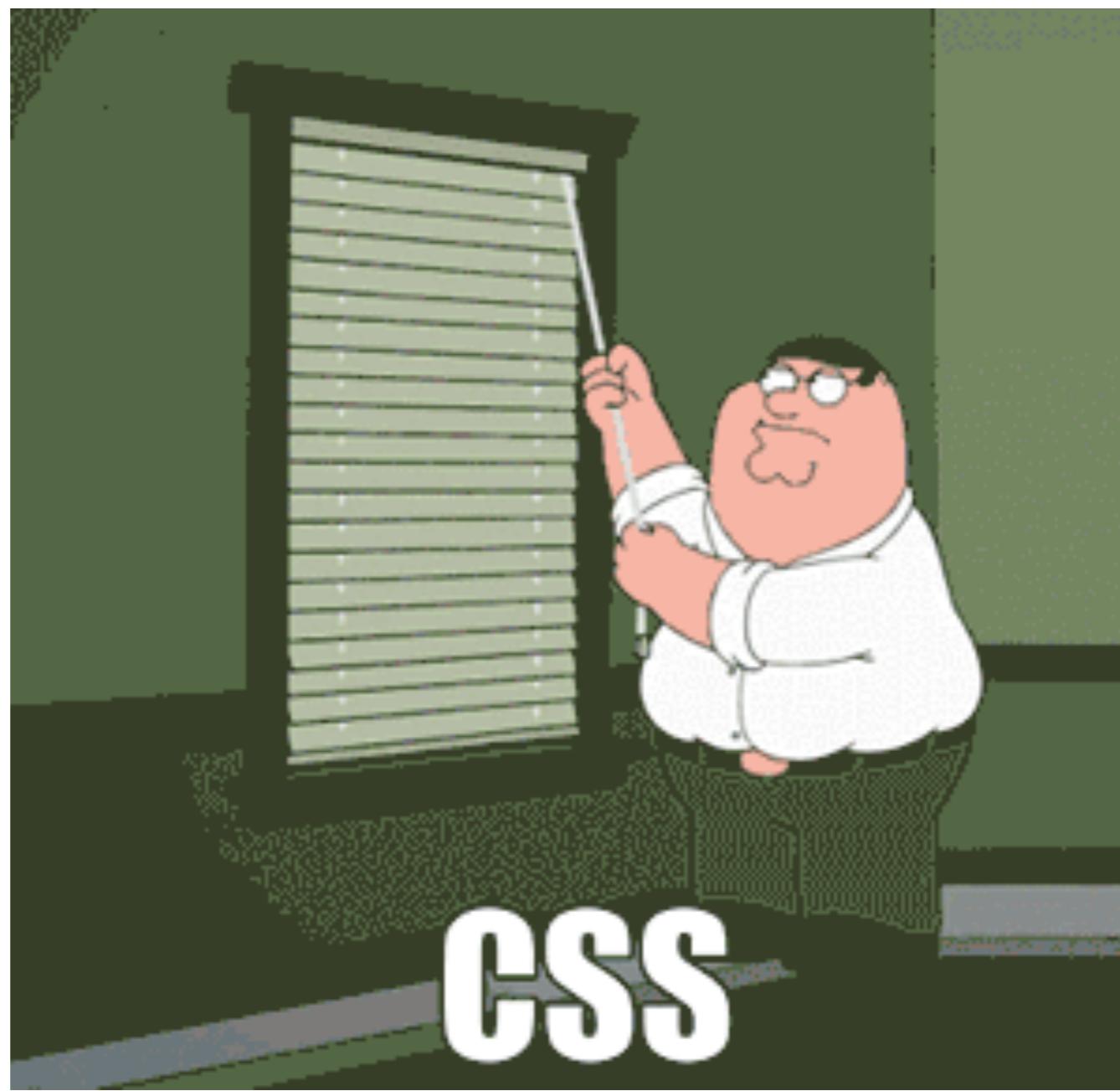


Как работает браузер



Как работает браузер





CSS

CSS

Cascading Style Sheets

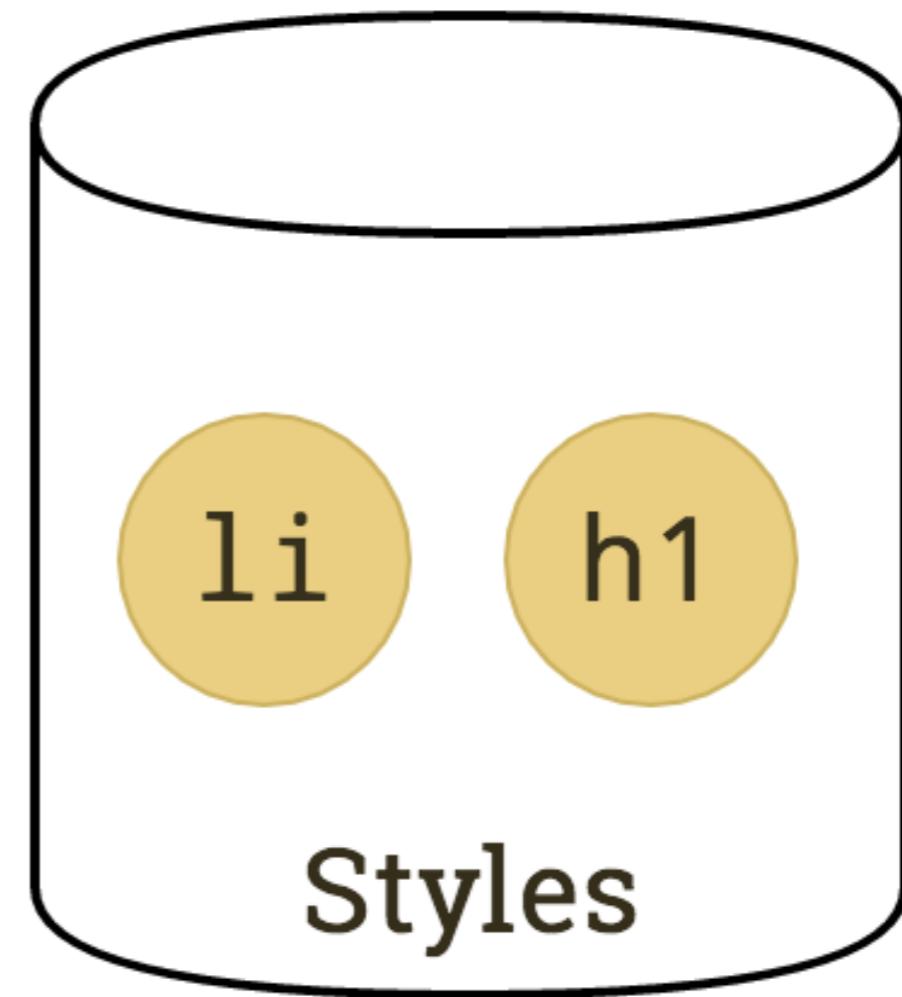
```
01: h1 {  
02:   color: #000000;  
03:   font-size: 18px;  
04:   font-weight: bold;  
05:   text-decoration: underline;  
06: }  
07:  
08: li {  
09:   color: blue;  
10:   font-family: Arial, sans-serif;  
11: }
```

CSS

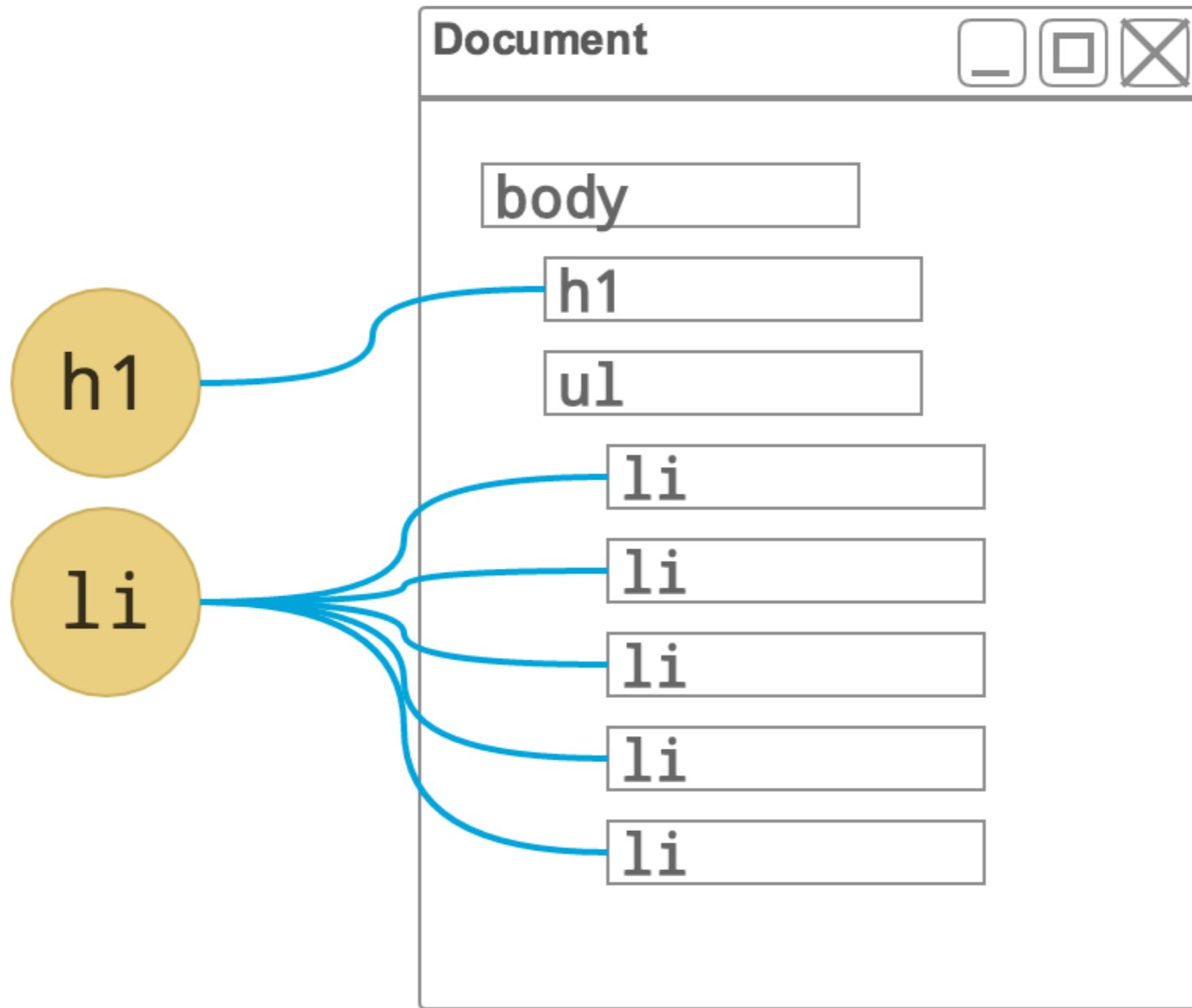
Cascading Style Sheets

styles.css

```
01: h1 {  
02: ...  
03: }  
04:  
05: li {  
06: ...  
07: }
```



Style matching



Участники

- Абдул Альзахред
- Нъярлатхотеп
- Шаб-Ниггурат
- Йог-сотхотх
- Ктулху

The screenshot shows the browser's developer tools with the 'Elements' tab selected. The DOM tree is displayed, highlighting the `<h1>Участники</h1>` element. Below the tree, the 'Styles' tab is active, showing the following CSS rules:

```
h1 { color: #000000; font-size: 18px; font-weight: bold; text-decoration: underline; } h1 { display: block; }
```

The first rule is annotated with `inspector-stylesheet:1` and the second with `user agent stylesheet`.

Синтаксис - общий случай

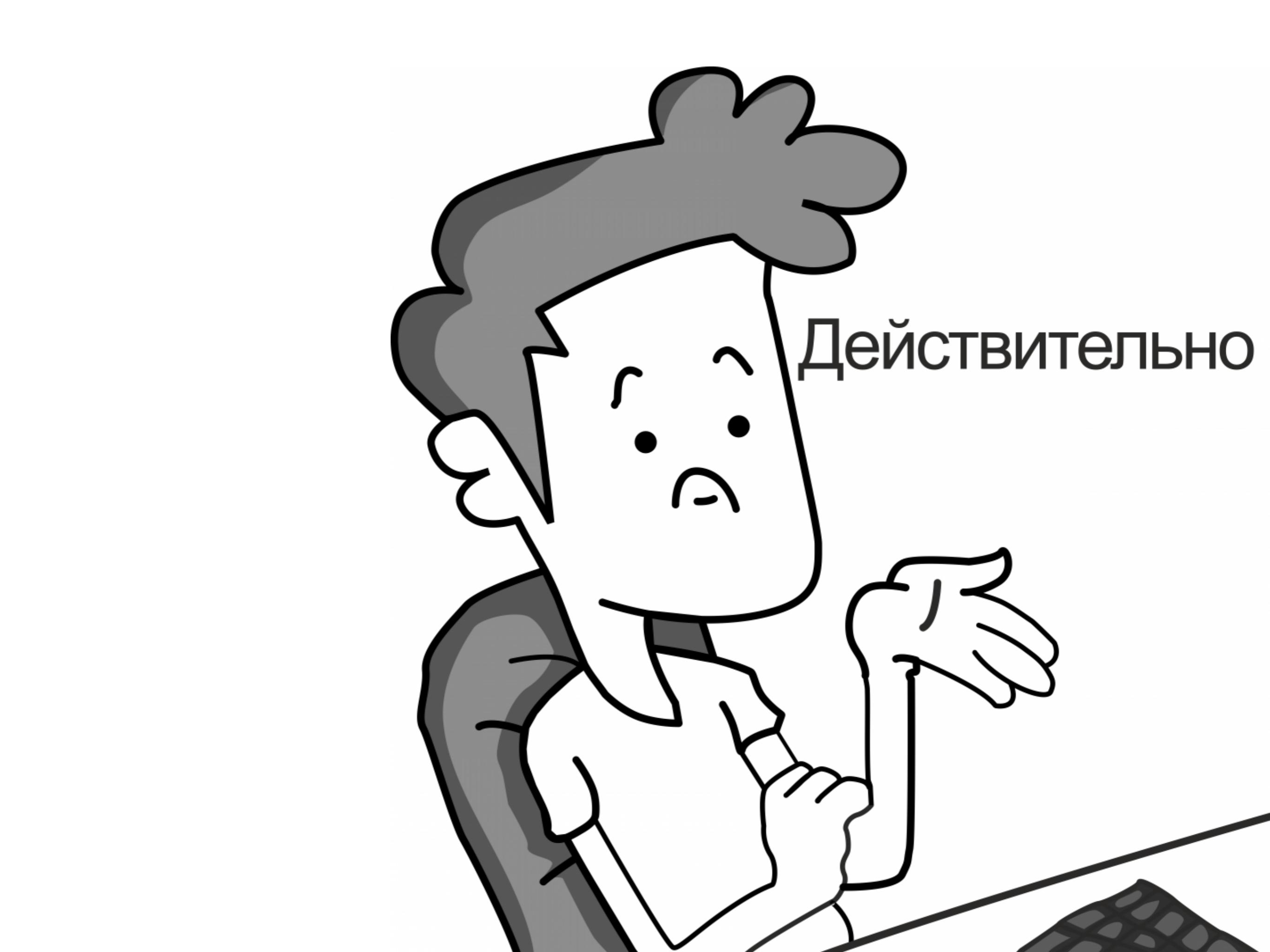
Ничего хитрого

```
01: селектор {  
02:     правило: значение;  
03: }
```

Селекторы

Селекторы

селекторы - выбирают

A black and white cartoon illustration of a woman with dark, curly hair tied back with a large bow. She has a shocked or surprised expression, indicated by wide eyes, a small mouth, and a question mark above her head. She is wearing a dark top and a necklace. Her hands are clasped together in front of her. To the right of the character, the word "Действительно" is written in a bold, sans-serif font.

Действительно

```
input#user.classOne.classTwo[type="text"] {  
    ...  
}
```

```
input#user.classOne.classTwo[type="text"] {  
    ...  
}
```

- Тэг - `input`
- Идентификатор - `user`
- Классы - `classOne` и `classTwo`
- Атрибут `type` в значении `text`

```
input#user.classOne.classTwo[type="text"] {  
    ...  
}
```

- Тэг - **input**
- Идентификатор - user
- Классы - classOne и classTwo
- Атрибут type в значении text

```
input#user.classOne.classTwo[type="text"] {  
    ...  
}
```

- Тэг - input
- Идентификатор - user
- Классы - classOne и classTwo
- Атрибут type в значении text

```
input#user.classOne.classTwo[type="text"] {  
    ...  
}
```

- Тэг - input
- Идентификатор - user
- Классы - **classOne** и **classTwo**
- Атрибут type в значении text

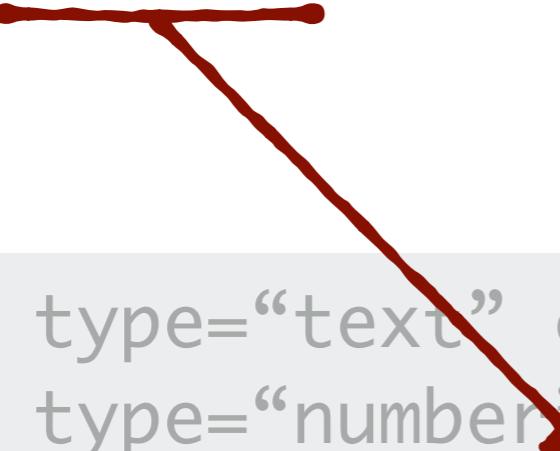
```
input#user.classOne.classTwo[type="text"] {  
    ...  
}
```

- Тэг - `input`
- Идентификатор - `user`
- Классы - `classOne` и `classTwo`
- Атрибут `type` в значении `text`

```
input#user.classOne.classTwo[type="text"]
```

```
01: <input type="text" class="classOne"></input>
02: <input type="number" i="age" class="classOne"></input>
03: <input type="text" id="user"
         class="classOne classTwo"></input>
04: <input type="text" class="classOne"></input>
05: <input type="text" name="first-name"></input>
06: <input class="classOne" placeholder="Hello"></input>
```

`input#user.classOne.classTwo[type="text"]`



```
01: <input type="text" class="classOne"></input>
02: <input type="number" i="age" class="classOne"></input>
03: <input type="text" id="user"
         class="classOne classTwo"></input>
04: <input type="text" class="classOne"></input>
05: <input type="text" name="first-name"></input>
06: <input class="classOne" placeholder="Hello"></input>
```

`input#user.classOne.classTwo[type="text"]`

```
01: <input type="text" class="classOne"></input>
02: <input type="number" id="age" class="classOne"></input>
03: <input type="text" id="user"
      class="classOne classTwo"></input>
04: <input type="text" class="classOne"></input>
05: <input type="text" name="first-name"></input>
06: <input class="classOne" placeholder="Hello"></input>
```

`input#user.classOne.classTwo[type="text"]`

The diagram illustrates the matching process of the CSS selector `input#user.classOne.classTwo[type="text"]` against a list of input elements. The selector consists of several parts: `input`, `#user`, `.classOne`, `.classTwo`, and `[type="text"]`. The matching logic is shown as follows:

- `input`: All elements in the list are `<input>` elements, so this part matches all of them.
- `#user`: A red arrow points from the `#user` part to the `id="user"` attribute in the third element.
- `.classOne`: A green arrow points from the `.classOne` part to the `class="classOne"` attribute in the third element.
- `.classTwo`: A green arrow points from the `.classTwo` part to the `class="classTwo"` attribute in the third element.
- `[type="text"]`: A gold arrow points from the `[type="text"]` part to the `type="text"` attribute in the first element.

```
01: <input type="text" class="classOne"></input>
02: <input type="number" id="age" class="classOne"></input>
03: <input type="text" id="user"
       class="classOne classTwo"></input>
04: <input type="text" class="classOne"></input>
05: <input type="text" name="first-name"></input>
06: <input class="classOne" placeholder="Hello"></input>
```

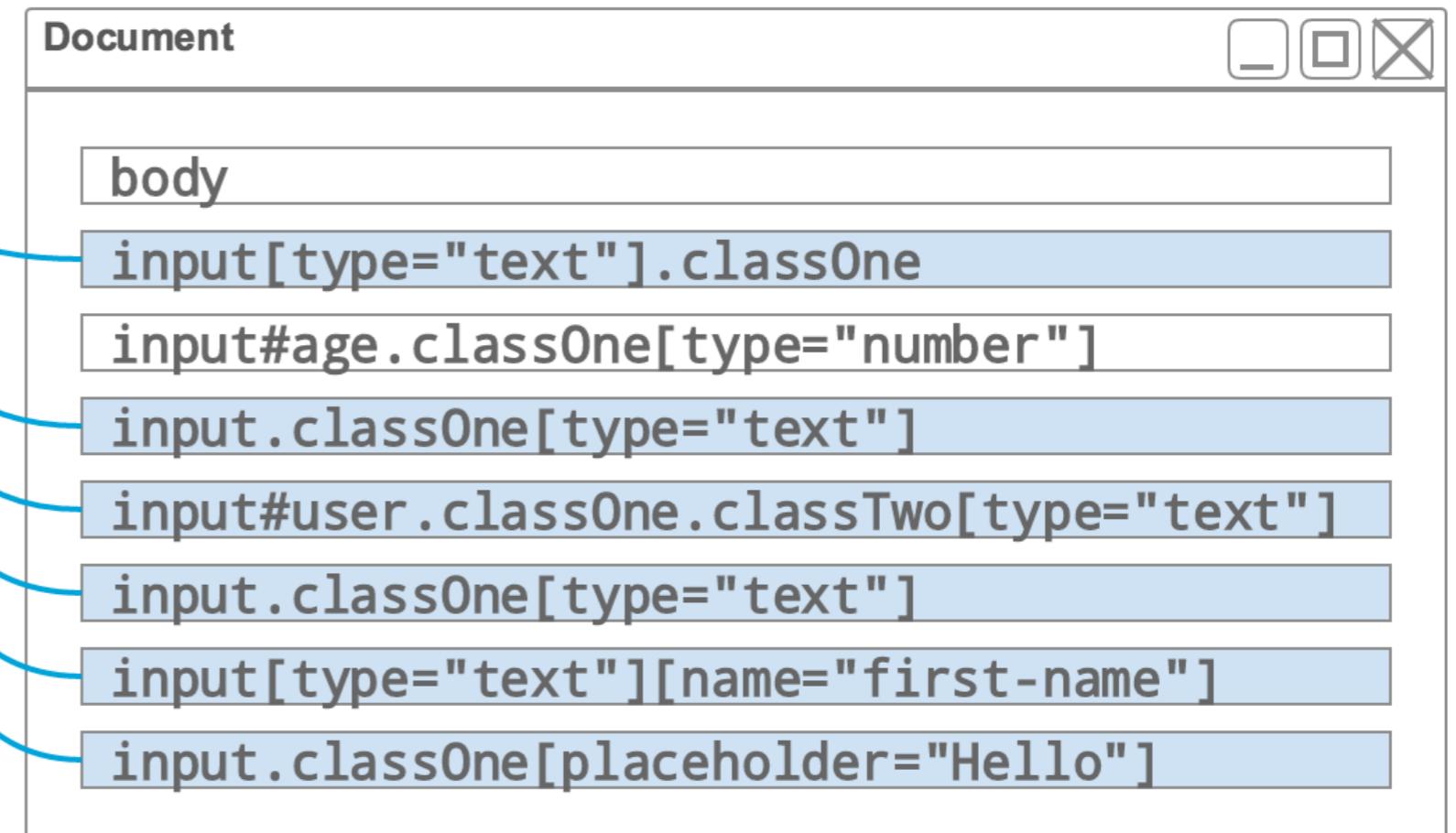
```
input#user.classOne.classTwo[type="text"]
```

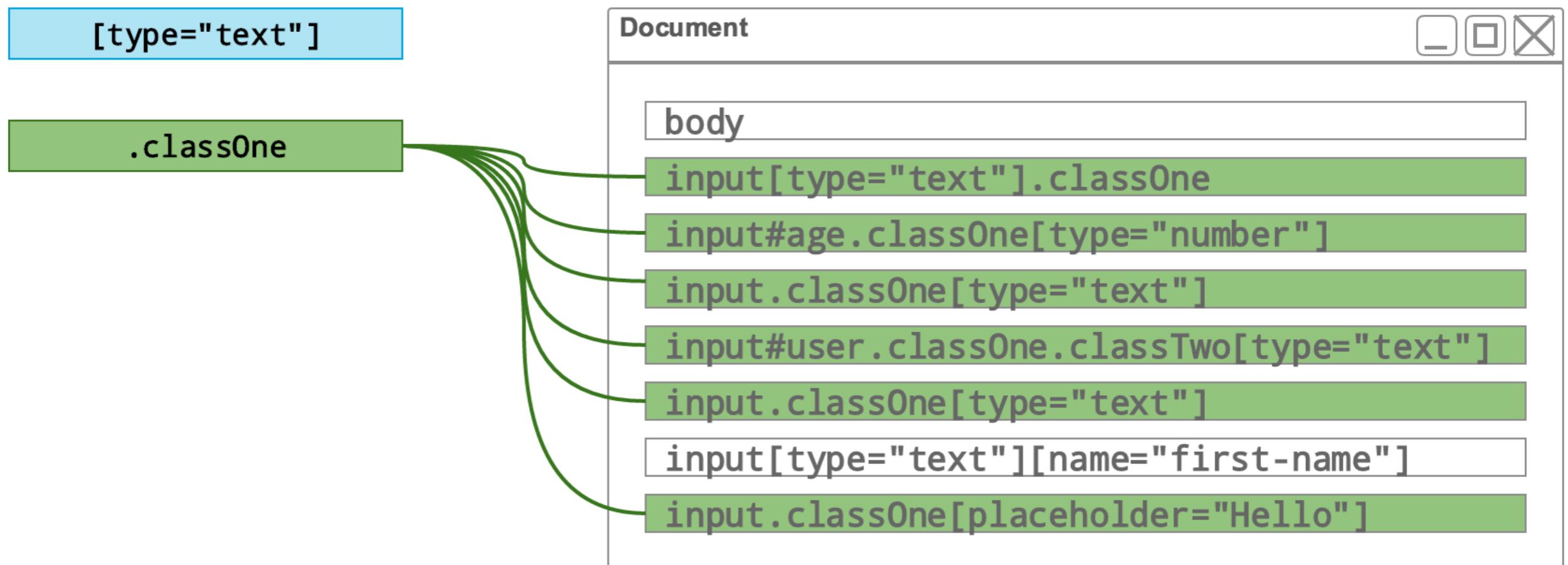
The screenshot shows a browser's developer tools with the "Elements" tab selected. On the left, there is a visual representation of the page containing several input fields. One input field in the middle contains the text "Hello". On the right, the DOM tree is displayed:

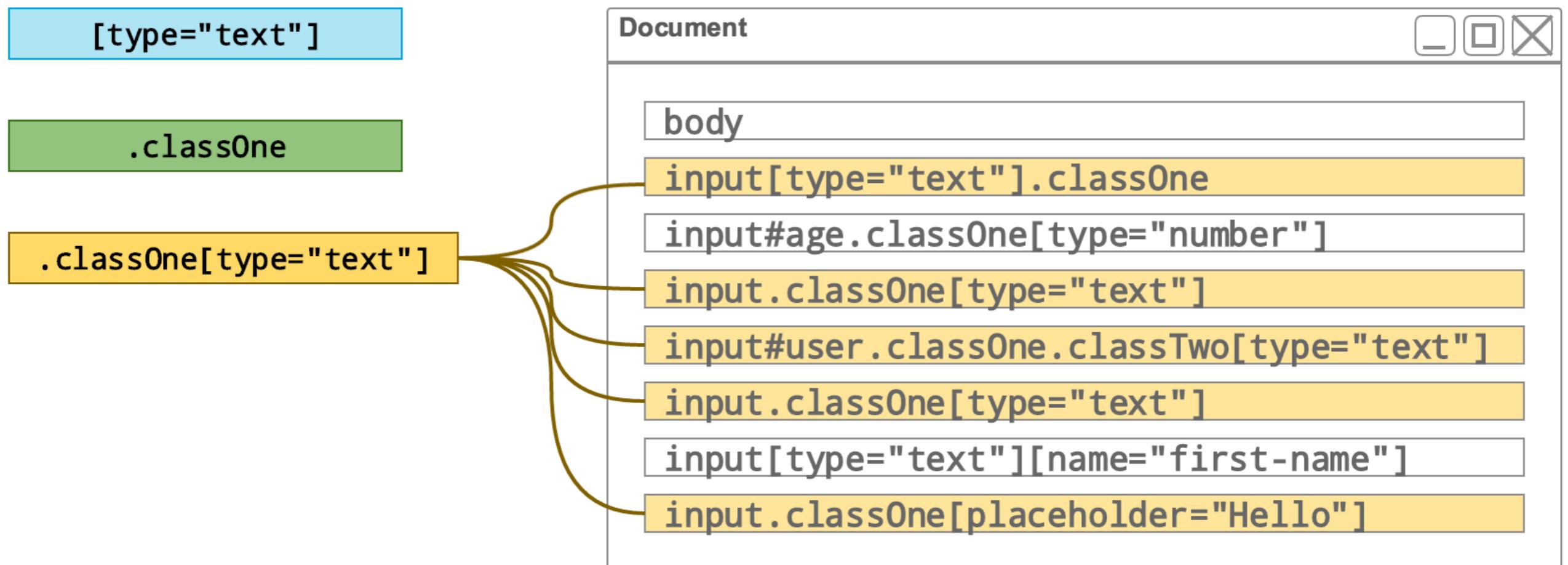
```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <input type="text" class="classOne">
    <input type="number" i="age" class="classOne">
    <input type="text" id="user" class="classOne classTwo"> // This node is highlighted with a gray background
    <input type="text" class="classOne">
    <input type="text" name="first-name">
    <input class="classOne" placeholder="Hello">
  </body>
</html>
```

The element `<input type="text" id="user" class="classOne classTwo">` is highlighted with a gray background. At the bottom of the developer tools interface, the path `html body input#user.classOne.classTwo` is shown, with the last part being highlighted in blue. A red horizontal line is drawn below the developer tools window.

[type="text"]







Каскадирование

body nav ul li.active a

Каскадирование

body nav ul li.active a



Каскадирование

body nav ul li.active a



Все элементы <a>

Каскадирование

body nav ul li.active a



Все элементы `<a>`, которые находятся внутри элемента `` с классом active

Каскадирование

body nav ul li.active a



Все элементы `<a>`, которые находятся внутри элемента `` с классом `active`, который находится внутри элемента ``, который находится в элементе `<nav>`, который находится внутри элемента `<body>`

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

*

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div.left

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div p

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div p

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div.left p

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div.left p

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

.title

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

.title

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

.left .title

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

.left .title

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div:first-child

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div:first-child

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

.left p:first-child

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

.left p:first-child

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div p:first-child

<body>

<div.left>

<p.title>

<p>

<p>

<p>

<div.right>

<p.title>

<p>

<p>

<p>

<footer>

div p:first-child

Каскадирование

вообще-то это важно

1. Переопределение

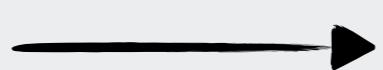
```
01: h1 {  
02:   color: #000000;  
03:   font-size: 18px;  
04:   font-weight: bold;  
05: }
```



Heading

1. Переопределение

```
01: h1 {  
02:   color: #000000;  
03:   font-size: 18px;  
04:   font-weight: bold;  
05: }
```

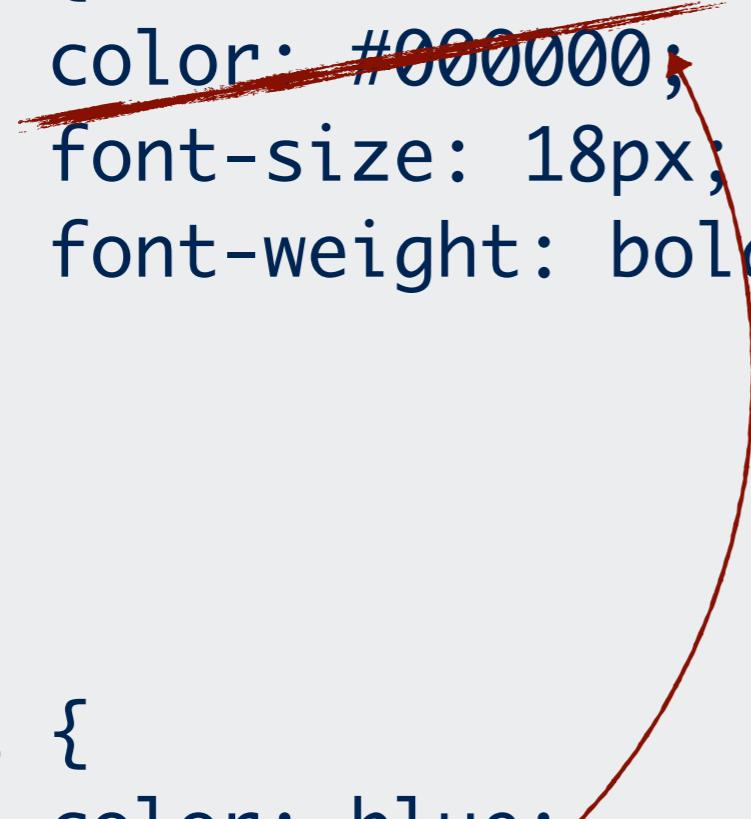


Heading

```
01: h1 {  
02:   color: blue;  
03: }
```

1. Переопределение

```
01: h1 {  
02:   color: #000000;  
03:   font-size: 18px;  
04:   font-weight: bold;  
05: }
```



→ Heading

```
01: h1 {  
02:   color: blue;  
03: }
```

1. Переопределение

```
01: h1 {  
02:   color: #000000;  
03:   font-size: 18px;  
04:   font-weight: bold;  
05: }
```

A diagram illustrating CSS inheritance. A red arrow points from the crossed-out 'color: #000000;' declaration in the first code block to the 'color' declaration in the second code block. Another red arrow points from the crossed-out 'font-size: 18px;' and 'font-weight: bold;' declarations in the first code block to their respective counterparts in the third code block. To the right of the second code block is a blue arrow pointing to the word 'Heading'.

→ Heading

```
01: h1 {  
02:   color: blue !important;  
03: }
```

```
01: h1 {  
02:   color: red;  
03: }
```

2. Специфичность

```
01: #h1Id {  
02:   color: #000000;           → Heading  
03: }
```

2. Специфичность

```
01: #h1Id {  
02:   color: #000000;  
03: }
```

...

```
01: h1 {  
02:   color: blue;  
03: }
```



Heading

2. Специфичность

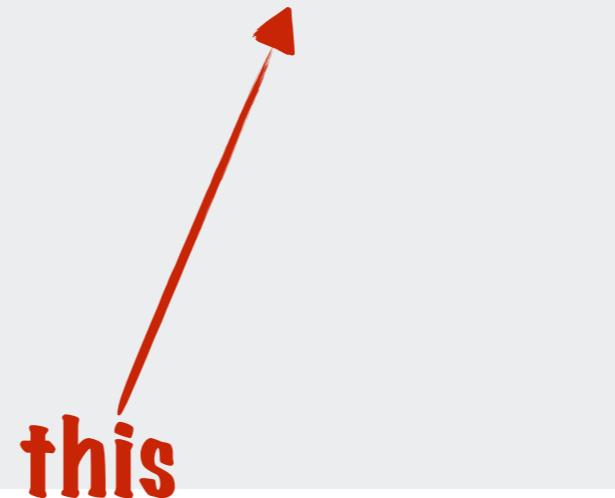
селектор	A	B	C	Специфичность
*	0	0	0	0
div	0	0	1	1
div:first	0	0	2	2
div span	0	0	2	2
div.head span	0	1	2	12
.head.title	0	2	0	20
#nav	1	0	0	100
#nav div#title	2	0	1	201

Как всем этим пользоваться?

```
01: <!DOCTYPE html>
02: <html>
03:   <head>
04:     <title> Привет, Moscow Coding School! </title>
05:     <link rel="stylesheet" href="cssfile.css">
06:   </head>
07:   <body>
08:     ...
09:   </body>
10: </html>
```

Как всем этим пользоваться?

```
01: <!DOCTYPE html>
02: <html>
03:   <head>
04:     <title> Привет, Moscow Coding School! </title>
05:     <link rel="stylesheet" href="cssfile.css">
06:   </head>
07:   <body>
08:     ...
09:   </body>
10: </html>
```



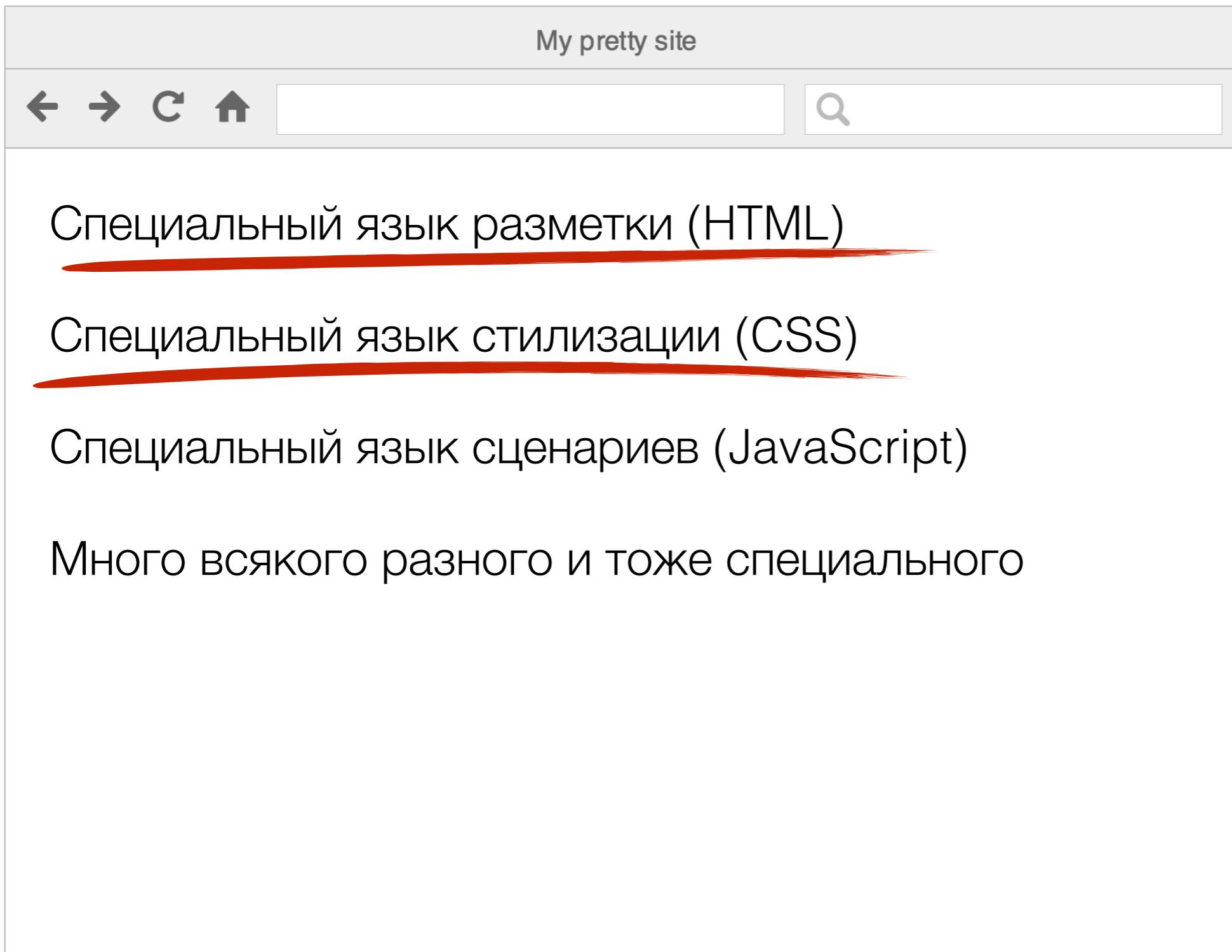
Как всем этим пользоваться?

```
01: <!DOCTYPE html>
02: <html>
03:   <head>
04:     <title> Привет, Moscow Coding School! </title>
05:     <style>
06:       h1 {
07:         color: black;
08:       }
09:     </style>
10:   </head>
11:   <body>
12:     ...
13:   </body>
14: </html>
```

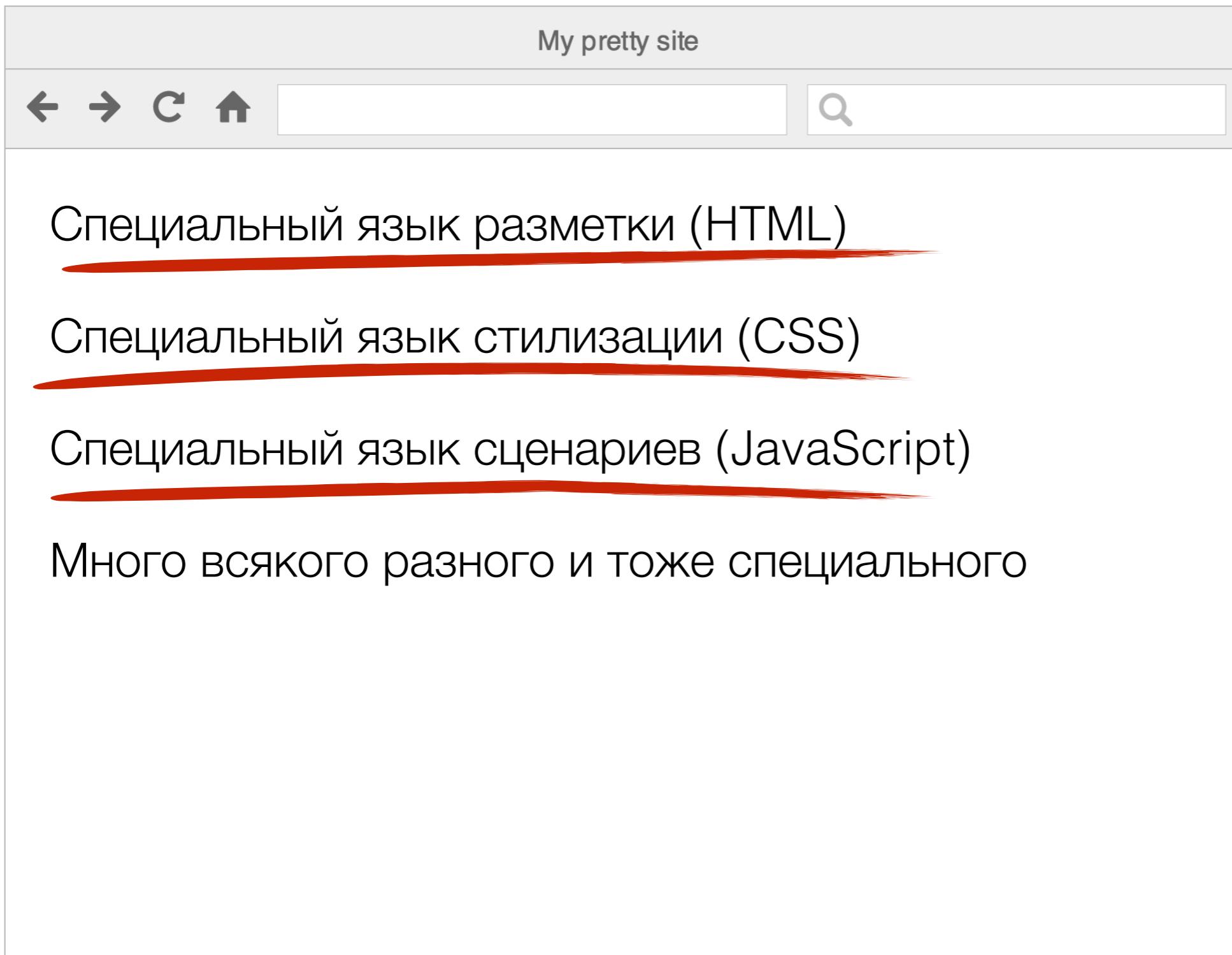
or this

Table Heading Cell	Table Heading Cell
Table Body Cell	Table Body Cell
Table Body Cell	Table Body Cell

Как работает браузер



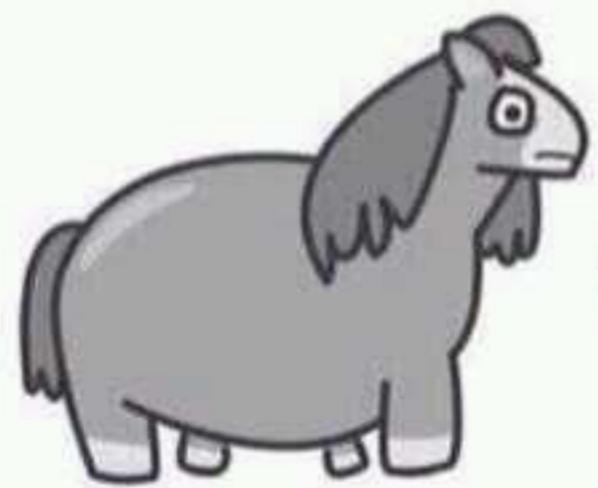
Как работает браузер



JS

the web

without js



with js



Интерпретация кода

```
01: var a = 10;  
02: var b = 20;  
03:  
04: var c = a + b;  
05:  
06: alert(c);
```

Интерпретация кода

```
01: var a = 10;  
02: var b = 20;  
03:  
04: var c = a + b;  
05:  
06: alert(c);
```



Процесс интерпретации

```
01: var a = 10;  
02: var b = 20;  
03:  
04: var c = a + b;  
05:  
06: alert(c);
```

Процесс интерпретации

```
01: var a = 10;           10
02: var b = 20;
03:
04: var c = a + b;
05:
06: alert(c);
```

Процесс интерпретации

```
01: var a = 10;           10
02: var b = 20;           20
03:
04: var c = a + b;
05:
06: alert(c);
```

Процесс интерпретации

```
01: var a = 10;           10
02: var b = 20;           20
03:
04: var c = a + b;        a+b
05:
06: alert(c);
```

Процесс интерпретации

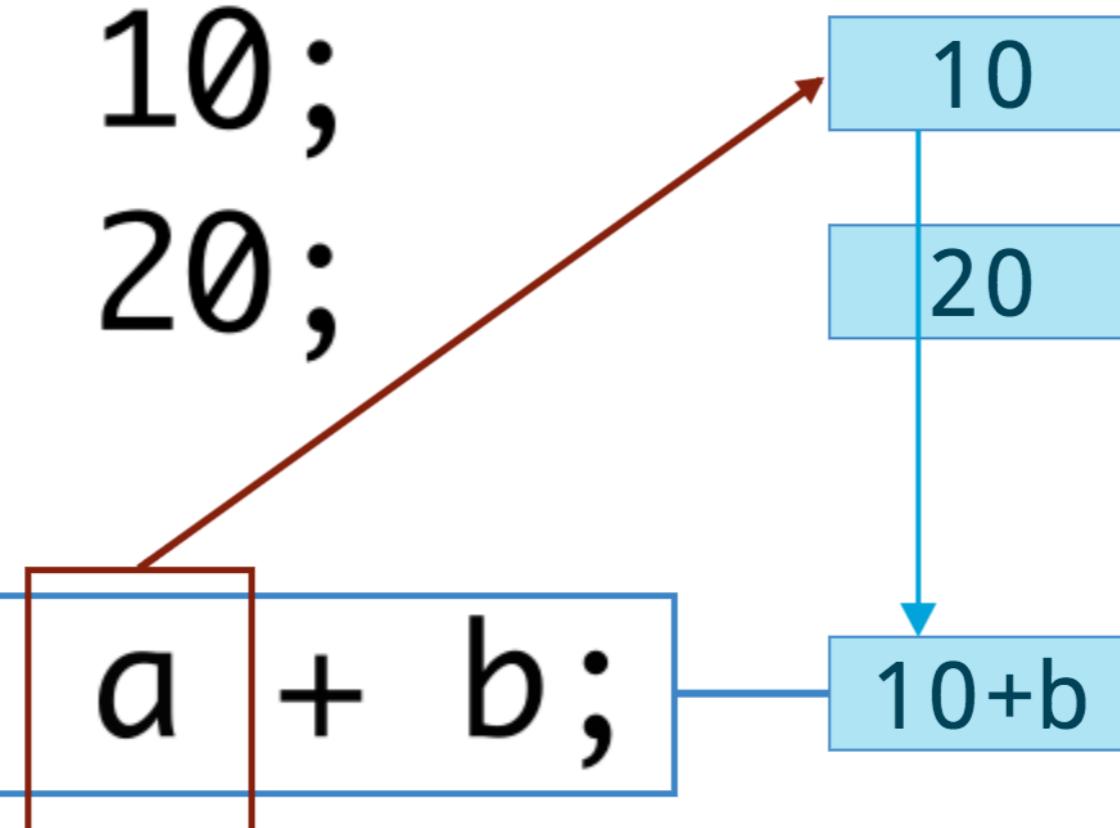
```
01: var a = 10;  
02: var b = 20;  
03:  
04: var c = a + b;  
05:  
06: alert(c);
```

The diagram illustrates the interpretation process of the provided JavaScript code. It shows variable declarations and their values being stored in boxes, and the execution of an addition operation.

- Line 01: `var a = 10;` - The value `10` is stored in a light blue box.
- Line 02: `var b = 20;` - The value `20` is stored in a light blue box.
- Line 04: `var c = a + b;` - The variable `a` is highlighted with a red box. A red arrow points from the `10` box to the `a` box. The expression `a + b` is highlighted with a blue box, and its result `a+b` is stored in a light blue box.
- Line 06: `alert(c);` - The variable `c` is highlighted with a blue box, and its value `a+b` is displayed in a light blue box.

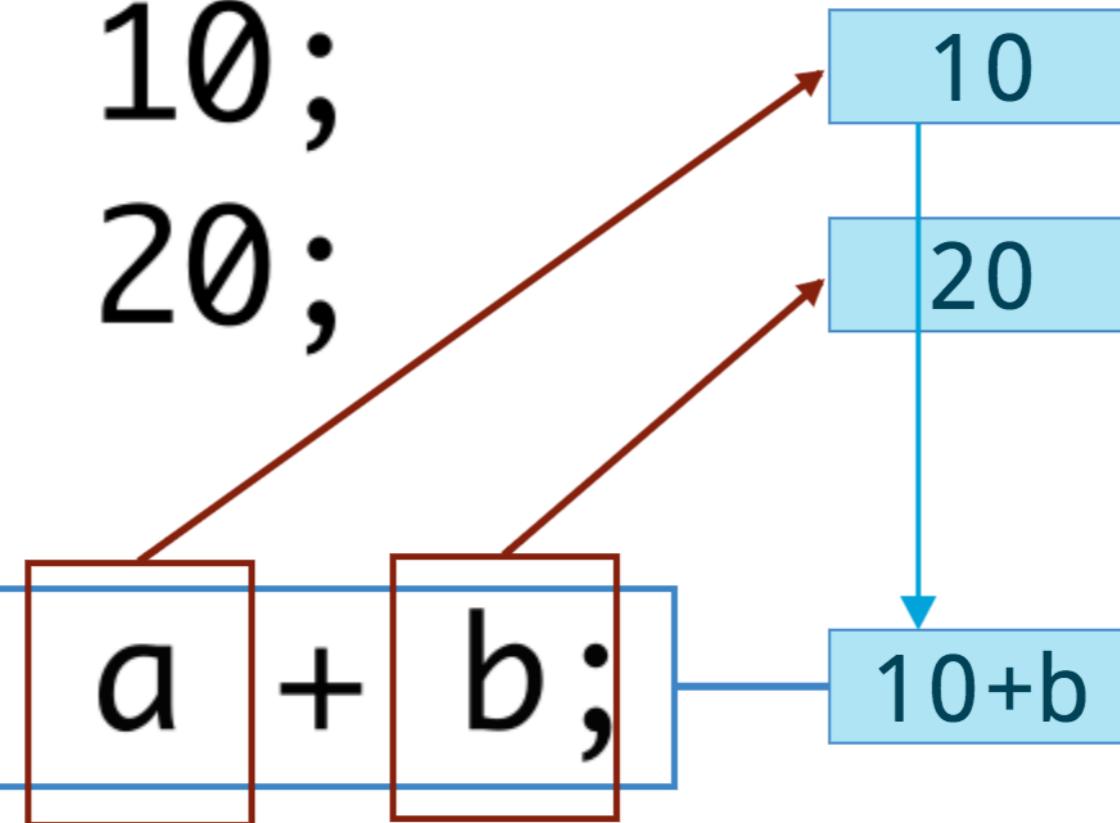
Процесс интерпретации

```
01: var a = 10;  
02: var b = 20;  
03:  
04: var c = a + b;  
05:  
06: alert(c);
```



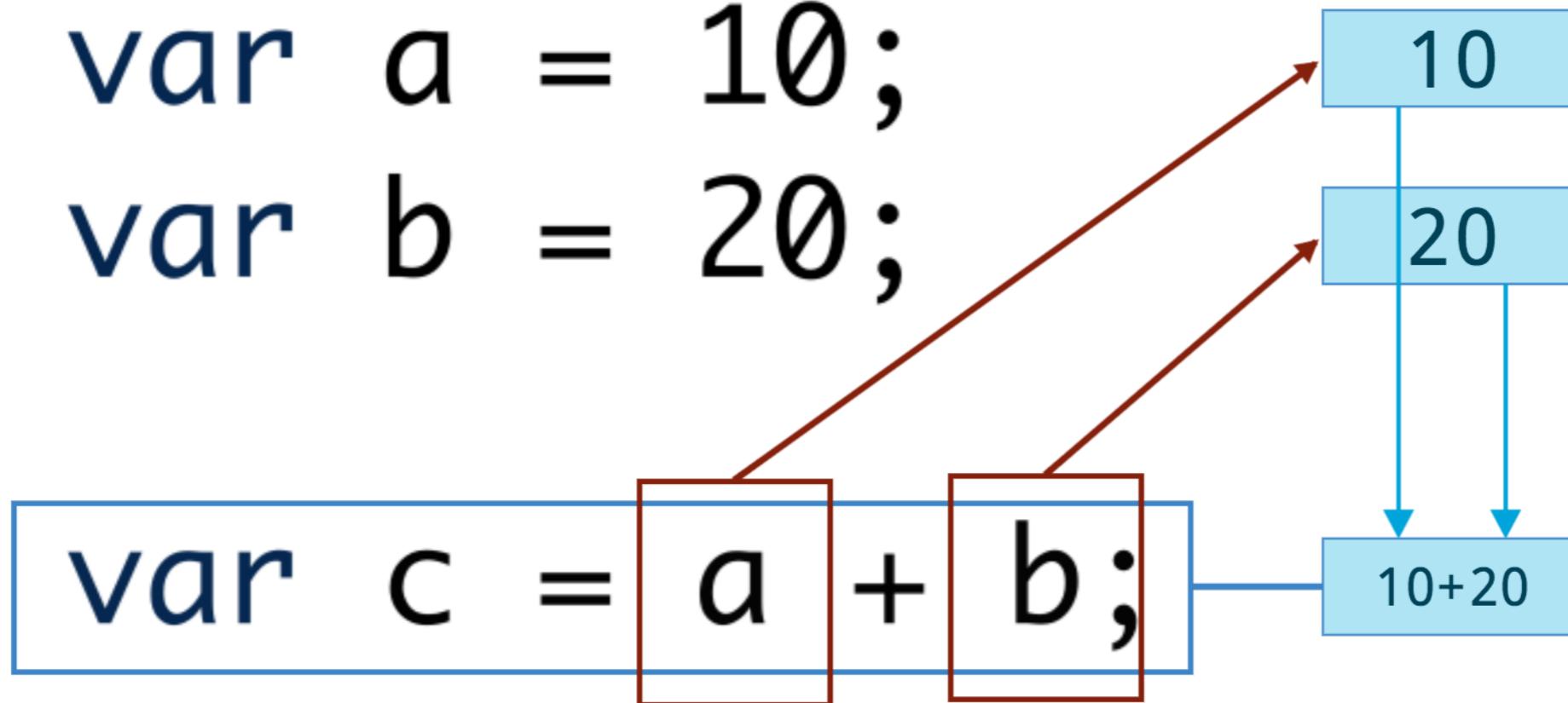
Процесс интерпретации

```
01: var a = 10;  
02: var b = 20;  
03:  
04: var c = a + b;  
05:  
06: alert(c);
```



Процесс интерпретации

```
01: var a = 10;  
02: var b = 20;  
03:  
04: var c = a + b;  
05:  
06: alert(c);
```



Процесс интерпретации

```
01: var a = 10;           10
02: var b = 20;           20
03:
04: var c = a + b;        30
05:
06: alert(c);
```

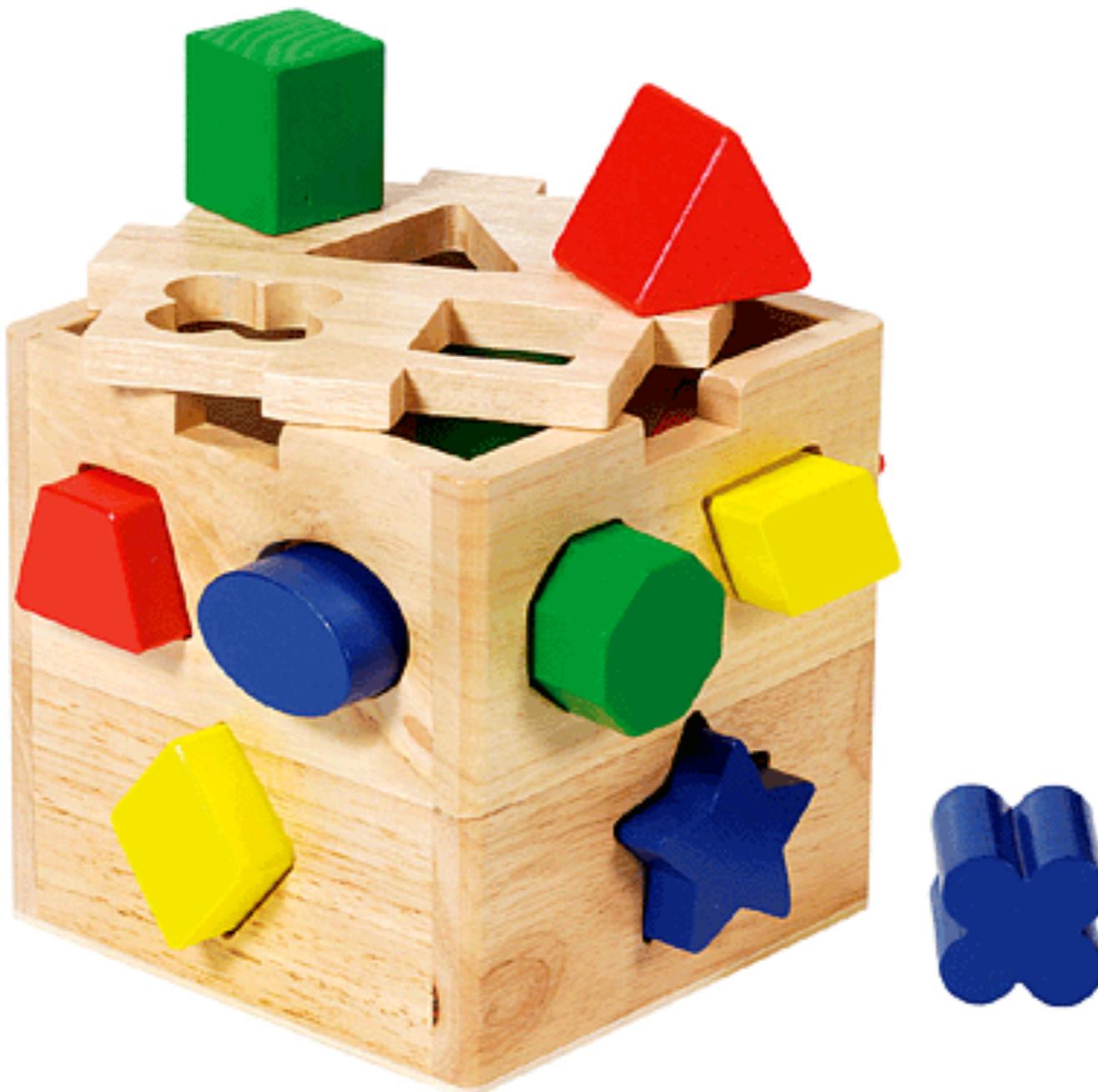


JavaScript Alert

30

OK

Типы данных



Типы данных

Значения	Тип
10, 010, -10, 12.5	number
“hello world”, “с”, ‘d’	string
true, false	boolean
function () { ... }	function
{x: 10, y: 20}	object
[10, 20]	object
undefined	undefined

Типы данных

`typeof variable`

вычисляет тип данных переменной `variable`

Statements (инструкции)

THE TRANSFORMERS[®]
MORE THAN MEETS THE EYE
INSTRUCTIONS: DEVASTATOR

Before assembling Devastator, transform each Constructicon as shown.

NOTE: Adult supervision may be necessary for younger children.

1. "Bonecrusher" Bulldozer: Attach launcher to post. Insert fist into launcher.

2. "Scavenger" Steam Shovel: Lift shovel. Attach launcher to post. Insert fist into launcher. Insert rifle into fist. Pull out hook from bottom.

3. "Scrapper" Payloader: Lift shovel. Swing front fenders down and around.

4. "Hook" Crane: Pull body halves apart and swing to sides. Swing pin out to sides. Flip robot head up. Slide robot head onto frame. Fit super wing on to frame.

5. "Long Haul" Dump Truck: Bend cab down. Lock into position. Lift roof. Turn dump truck over. Slide missile mount into frame between rear wheels. Insert wing into hole under cab.

6. "Mixmaster" Cement Mixer: Pull body halves apart. Grasp cab from underneath and swing up.

Assemble Devastator as shown.

NOTE: Missile launchers are designed only to hold missiles in place; therefore propulsion will be very weak, if at all.

A. To attach steam shovel to crane, rotate cab on crane down. Angle steam shovel, insert metal hook into hole in side of crane. Swing steam shovel down. Swing cab back up.
B. Press hole on bottom of bulldozer over post on side of crane.
C. Push rear of payloader onto missile mount on dumptruck.
D. Push rear of cement mixer onto missile mount on dumptruck.
E. Place post and hole on crane onto hole and post on dump truck.

PROOF OF AUTHENTICITY

Can you find the black square label on your TRANSFORMER? Rub the label — Watch robot face appear! It is your evidence that this robot is a true TRANSFORMER!

© 1984 Hasbro Bradley, Inc. Pawtucket, R.I. 02861 USA. All Rights Reserved. Made and printed in Japan.
Manufactured by Takara Co., Ltd. Tokyo, Japan. 5715 ASST.

Statements (инструкции)

```
01: // comment
02:
03: 666;
04:
05: 1 + 1;
06:
07: var x;
08:
09: var x = 10;
10:
11: var x = 10 + 20;
```

Операторы



Оператор +

01: 5 + 6 → 11 (число + число)

02:

03: 5 + '6' → '56' (число + строка)

Оператор +

01: 5 + 6 -> 11 (число + число)

02:

03: 5 + '6' -> '56' (число + строка)

04:

05: +'10' -> 10

Оператор +

01: 5 + 6 → 11 (число + число)

02:

03: 5 + '6' → '56' (число + строка)

04:

05: +'10' → 10

06:

07: {} + {} → NaN

Оператор +

01: 5 + 6 → 11 (число + число)

02:

03: 5 + '6' → '56' (число + строка)

04:

05: +'10' → 10

06:

07: {} + {} → NaN

08:

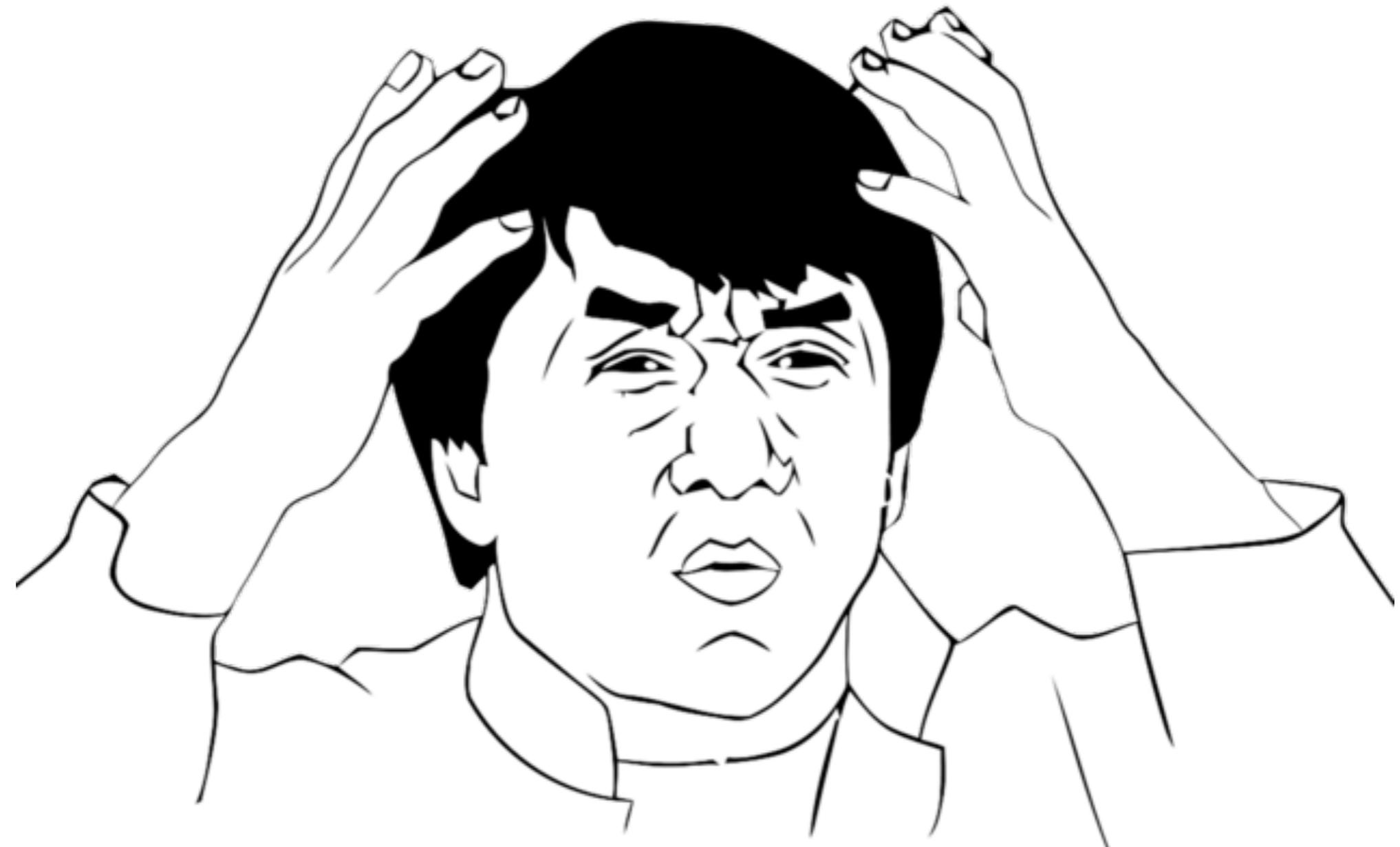
09: {} + '' → 0

Оператор +

```
01: 5 + 6      -> 11 (число + число)
02:
03: 5 + '6'    -> '56' (число + строка)
04:
05: +'10'      -> 10
06:
07: {} + {}    -> NaN
08:
09: {} + ''    -> 0
10:
11: '' + {}    -> '[object Object]'
```

Оператор +

```
01: 5 + 6      -> 11 (число + число)
02:
03: 5 + '6'    -> '56' (число + строка)
04:
05: +'10'      -> 10
06:
07: {} + {}    -> NaN
08:
09: {} + ''    -> 0
10:
11: '' + {}    -> '[object Object]'
12:
13: true + 1   -> 2
```



Операторы - , / , * , %

Тоже самое, что +, только всегда приводят к числу

Операторы сравнения

$a > b$ - а строго больше b

$a \geq b$ - а больше или ровно b

$a < b$ - а строго меньше b

$a \leq b$ - а меньше или ровно b

$a == b$ - а равен b

$a === b$ - а строго равен b

Логические операторы

`a && b` - а и b

`a || b` - а или b

`!a` - не a

Логические операторы

`a && b` - а и b

`a || b` - а или b

`!a` - не a

`!a && !b` - не a и не b

`a || !b` - а или не b

Тернарный оператор

a ? b : c

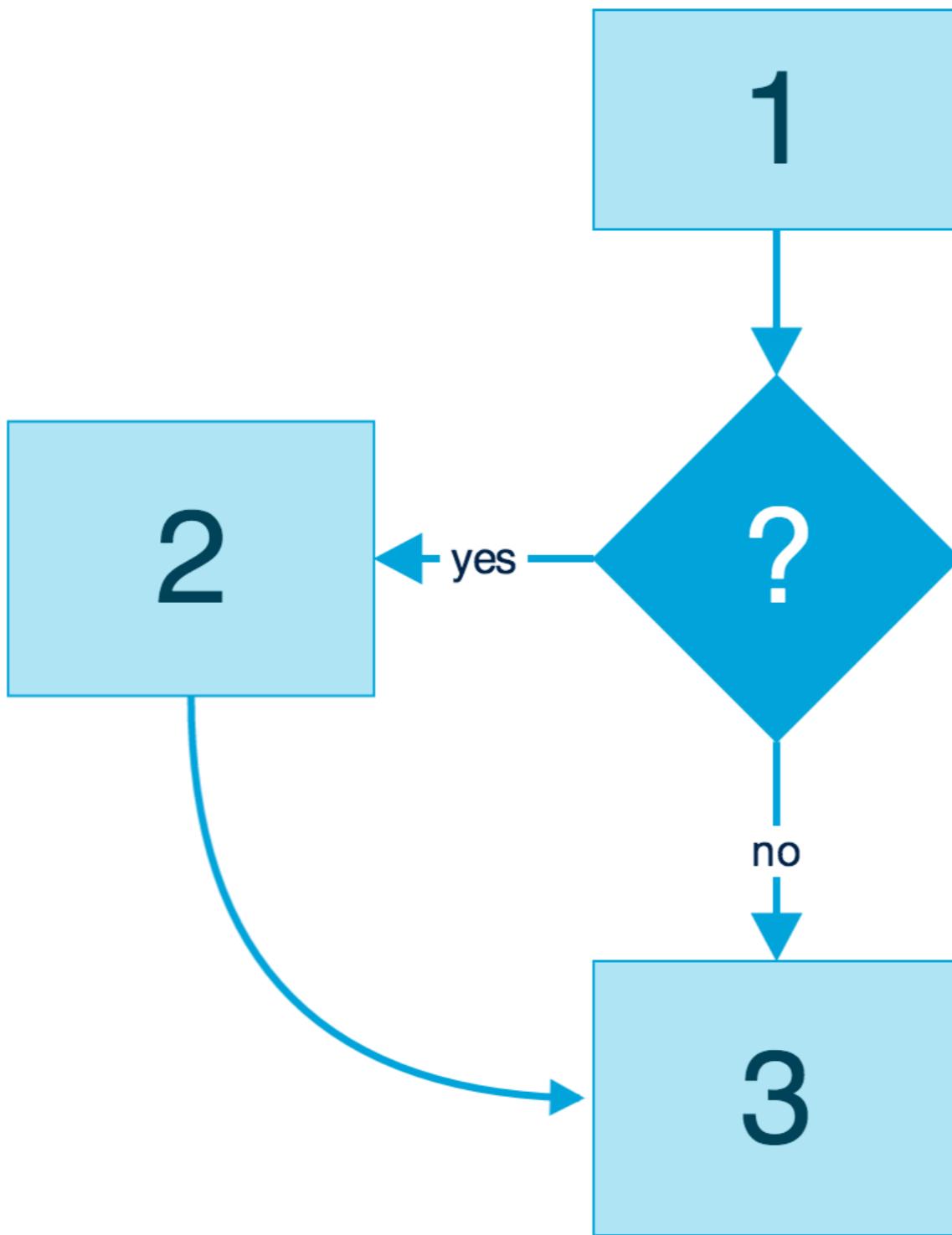
```
if (a) {  
    b  
} else {  
    c  
}
```



Управляющие конструкции

- Ветвления
- Циклы

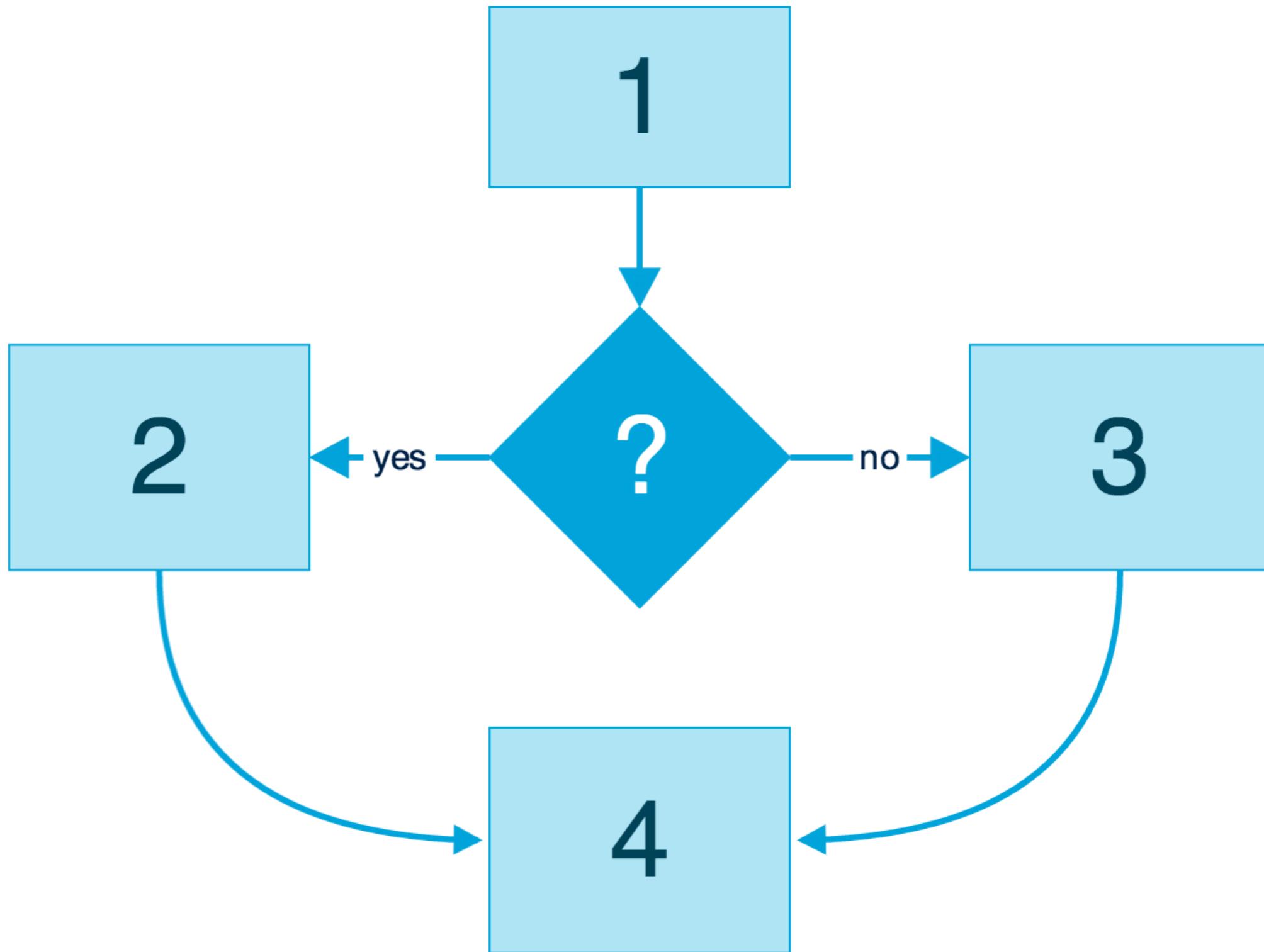
Ветвление



Ветвление

```
01: // 1
02: if (condition) {
03:     // 2
04: }
05: // 3
```

Ветвление



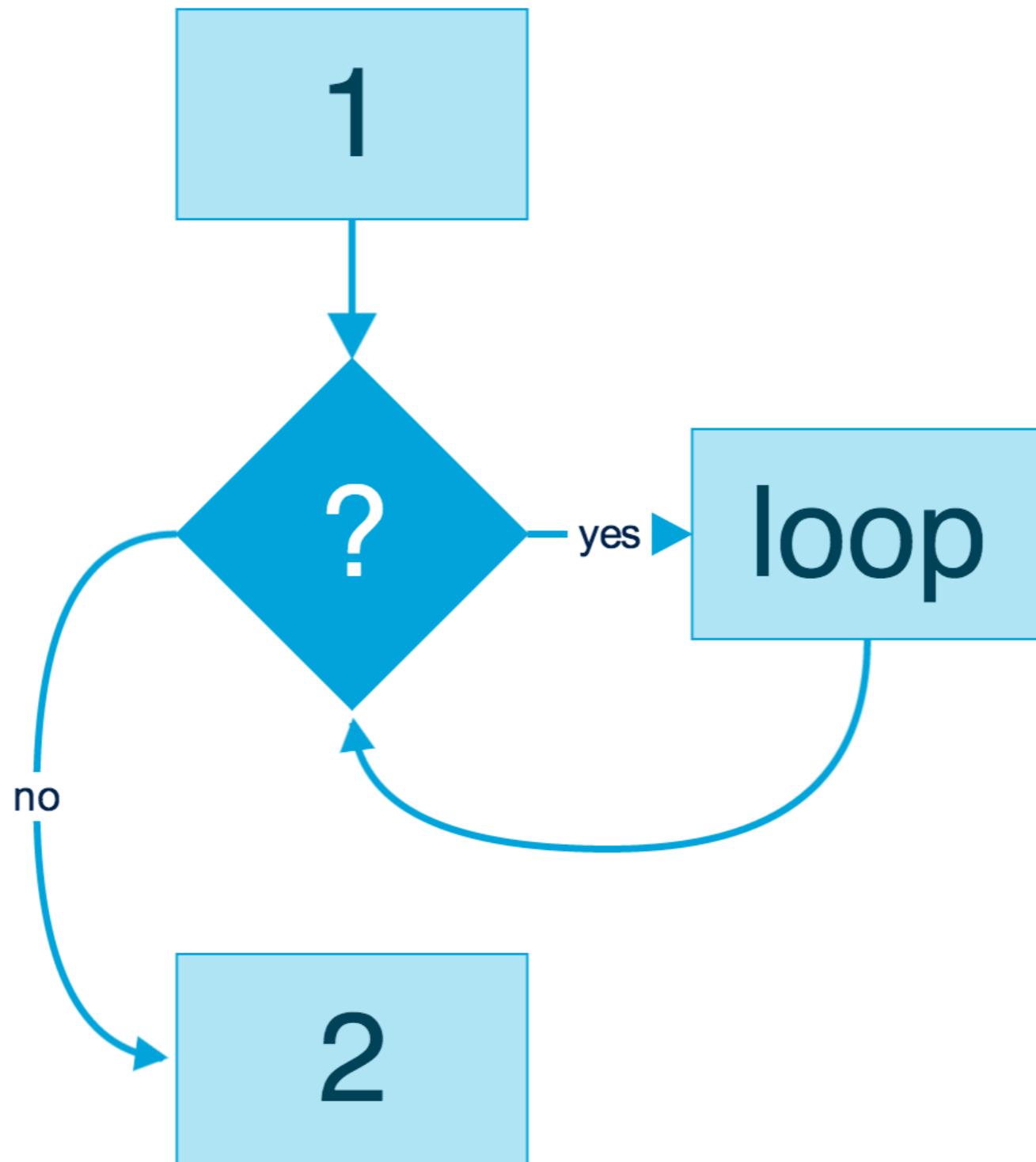
Ветвление

```
01: // 1
02: if (condition) {
03:     // 2
04: } else {
05:     // 3
06: }
07: // 4
```

Ветвление

```
01: // 1
02: if (condition) {
03:     // 2
04: } else
05: if (condition2) {
06:     // 3
07: } else {
08:     // 4
09: }
10: // 5
```

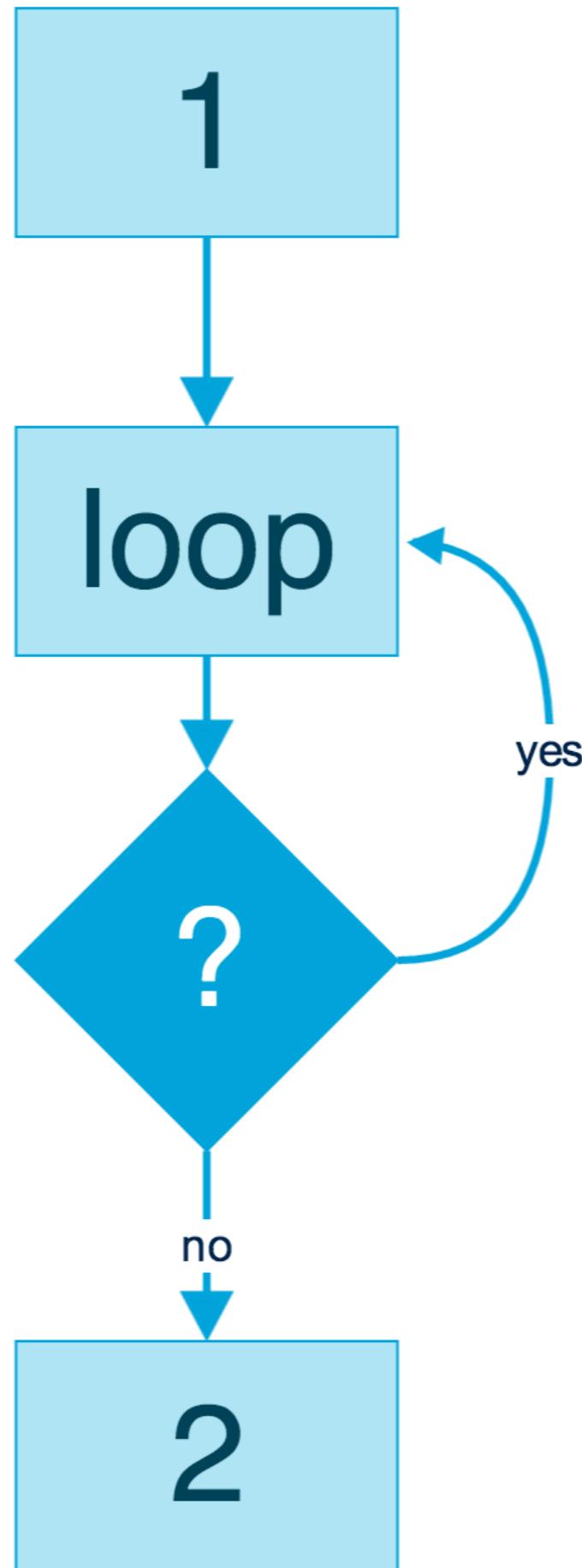
Цикл: while



Цикл: while

```
01: // 1
02: while (condition) {
03:     // loop body
04: }
05: // 2
```

Цикл: do-while



Цикл: do-while

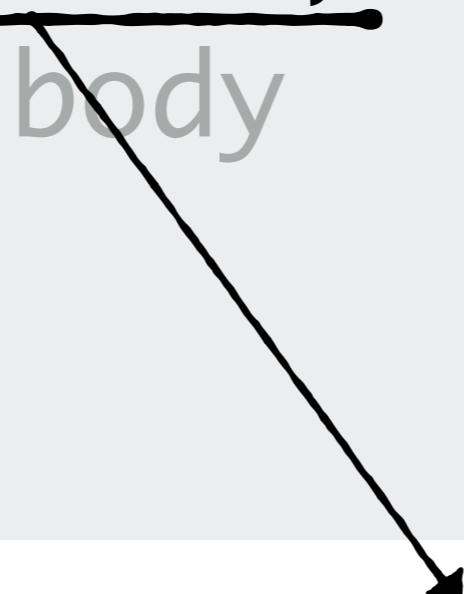
```
01: // 1
02: do {
03:     // loop body
04: } while (condition);
05: // 2
```

Цикл: for

```
01: // 1
02: for (var i = 0; i < 10; i++){
03:   // loop body
04: }
05: // 2
```

Цикл: for

```
01: // 1
02: for (var i = 0; i < 10; i++){
03:     // loop body
04: }
05: // 2
```



Инициализация

Цикл: for

```
01: // 1  
02: for (var i = 0; i < 10; i++){  
03:     // loop body  
04: }  
05: // 2
```

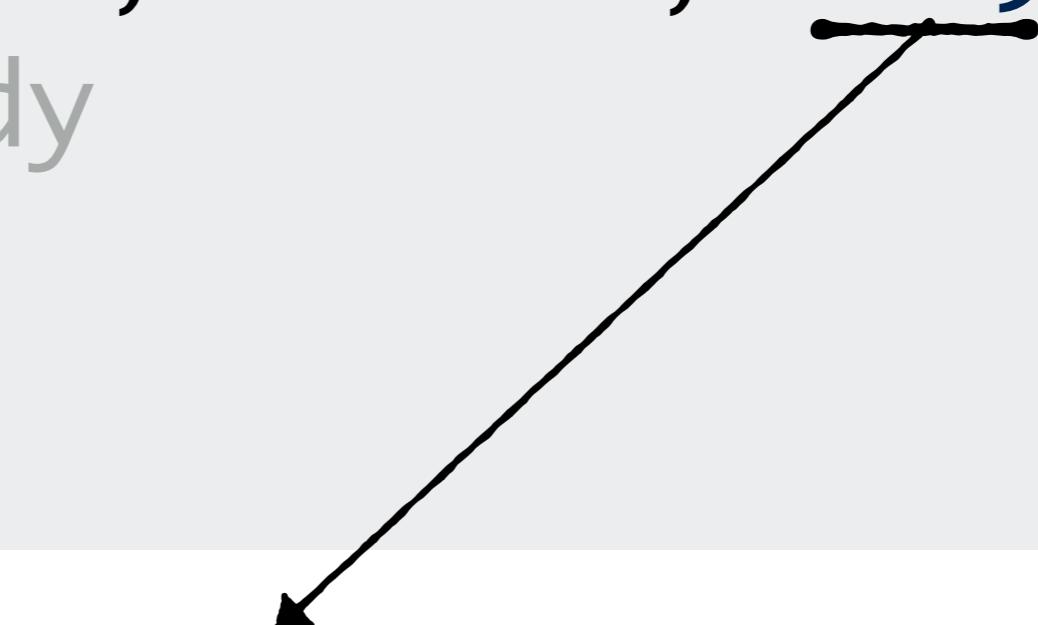


i < 10

Проверка

Цикл: for

```
01: // 1
02: for (var i = 0; i < 10; i++){
03:     // loop body
04: }
05: // 2
```



Обновление

Цикл: for

```
01: // 1
02: for (var i = 0; i < 10; i++){
03:   console.log(i);
04: }
05: // 2
```

ФУНКЦИИ

```
01: function greet(name) {  
02:     return 'Hello, ' + name;  
03: }  
04:  
05: var message = greet('World!');  
06:  
07: alert(message);
```

ФУНКЦИИ

определение

```
01: function greet(name) {  
02:     return 'Hello, ' + name;  
03: }  
04:  
05: var message = greet('World!');  
06:  
07: alert(message);
```

ФУНКЦИИ

```
01: function greet(name) {  
02:     return 'Hello, ' + name;  
03: }  
04:  
05: var message = greet('World!');  
06:  
07: alert(message);
```

ВЫЗОВ

ВЫЗОВ



JavaScript Alert

Hello, World!

OK

Параметры

Тело функции

Результат

Идеальный пример функции

Мясо

Мясорубка

Фарш

```
01: function inc(x) {  
02:     return x + 1;  
03: }  
04:  
05: var a = inc(10);  
06:  
07: alert(a);
```

```
01: function inc(x) {  
02:     return x + 1;  
03: }  
04:  
05: var a = inc(10);  
06:  
07: alert(a);
```

inc(10)

Число

Число + 1

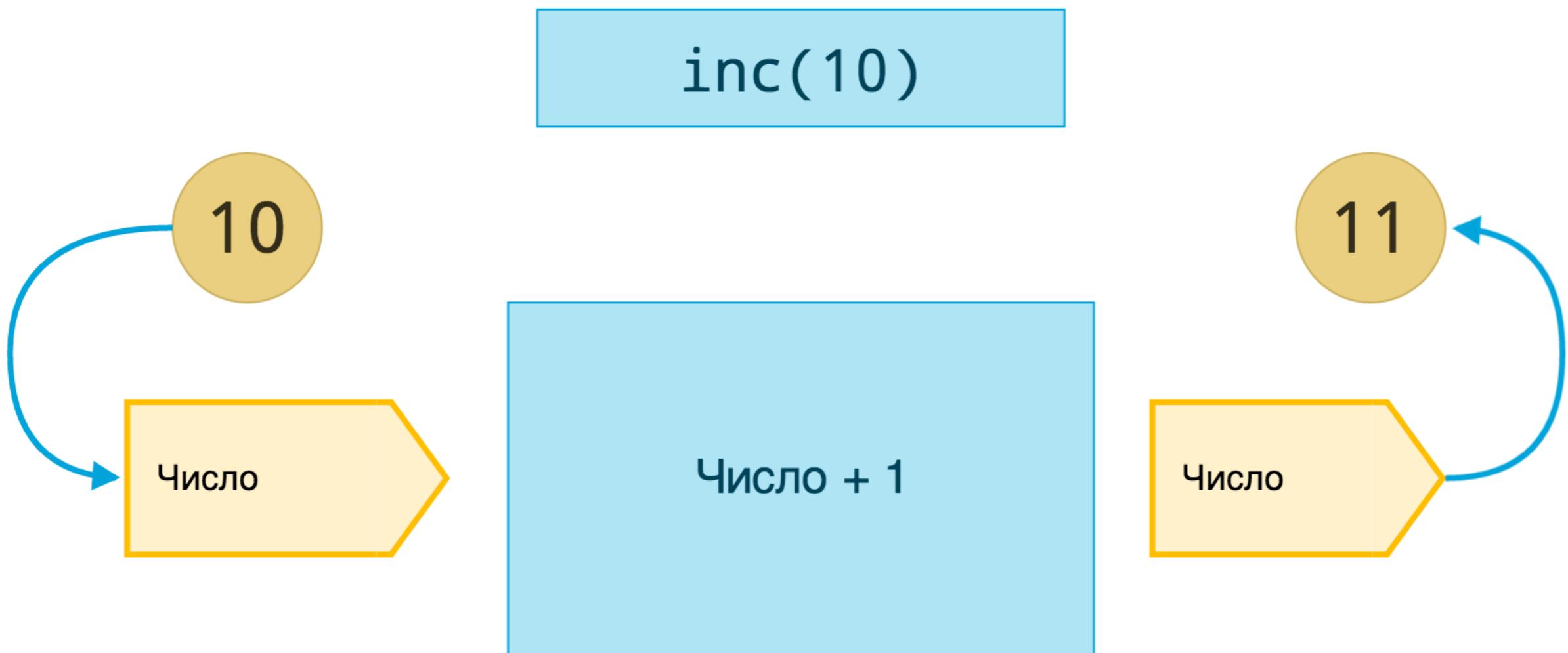
Число

```
01: function inc(x) {  
02:     return x + 1;  
03: }  
04:  
05: var a = inc(10);  
06:  
07: alert(a);
```

inc(10)



```
01: function inc(x) {  
02:     return x + 1;  
03: }  
04:  
05: var a = inc(10);  
06:  
07: alert(a);
```



ФУНКЦИИ - ЗНАЧЕНИЯ

```
01: var greet = function (name) {  
02:     return 'Hello, ' + name;  
03: }  
04:  
05: alert(greet('World!'));
```

ФУНКЦИИ - ЗНАЧЕНИЯ

```
01: var add = function (x, y) {  
02:     return x + y;  
03: }  
04:  
05: var add1 = function (x) {  
06:     return add(x, 1);  
07: }  
08:  
09: alert(add1(10));
```

ФУНКЦИИ - ЗНАЧЕНИЯ

```
01: var addN = function (x) {  
02:     return function(y){  
03:         return x + y;  
04:     };  
05: }  
06:  
07: var add1 = addN(1);  
08:  
09: alert(add1(10));
```

Область видимости (scope)

```
01: var a = 20;  
02:  
03: b = 30;  
04:  
05: function name(){  
06:     var a = 30;  
07: }  
08:  
09: if(true){  
10:     var a = 30;  
11: }
```

Область видимости (scope)

```
01: var a = 20; ← Объявление локальной
02:
03: b = 30;
04:
05: function name(){
06:     var a = 30;
07: }
08:
09: if(true){
10:     var a = 30;
11: }
```

Область видимости (scope)

```
01: var a = 20; ← Объявление локальной
02:
03: b = 30; ← Объявление глобальной
04:
05: function name(){}
06:   var a = 30;
07: }
08:
09: if(true){
10:   var a = 30;
11: }
```

Область видимости (scope)

```
01: var a = 20;                                     глобальная область
02:
03: b = 30;
04:
05: function name(){
06:   var a = 30;
07: }
08:
09: if(true){
10:   var a = 30;
11: }
```

Область видимости (scope)

```
01: var a = 20;                                     глобальная область
02:
03: b = 30;
04:
05: function name(){
06:   var a = 30;                                    локальная область
07: }
08:
09: if(true){
10:   var a = 30;
11: }
```

Область видимости (scope)

В JavaScript только функции создают новую область видимости

Замыкания (closures)

```
01: var cache = function (){  
02:     var store = {};  
03:  
04:     return function (key, value){  
05:         if (value){  
06:             return store[key] = value;  
07:         } else {  
08:             return store[key]  
09:         }  
10:     }  
11: };
```

Замыкания (closures)

```
01: var cache = function (){  
02:     var store = {};  
03:  
04:     return function (key, value){  
05:         if (value){  
06:             return store[key] = value;  
07:         } else {  
08:             return store[key]  
09:         }  
10:     }  
11: };
```

Замыкания (closures)

```
01: var cache = function (){  
02:     var store = {};  
03:  
04:     return function (key, value){  
05:         if (value){  
06:             return store[key] = value;  
07:         } else {  
08:             return store[key]  
09:         }  
10:     }  
11: };
```

Замыкания (closures)

```
01: var cache = function (){  
02:   var store = {};  
03:  
04:   return function (key, value){  
05:     if (value){  
06:       return store[key] = value;  
07:     } else {  
08:       return store[key]  
09:     }  
10:   }  
11: };
```

сохраняется ссылка
на store, но сама
переменная store
недоступна

Замыкания (closures)

В JavaScript - единственный способ создать приватные переменные (те, к которым нет доступа из вне) — это замыкания.

Хитрый кунштюк

```
01: var cache = function (){  
02:     var store = {};  
03:  
04:     return function (key, value){  
05:         if (value){  
06:             return store[key] = value;  
07:         } else {  
08:             return store[key]  
09:         }  
10:     }  
11: }
```

Хитрый кунштюк

```
01: var cache = (function (){  
02:     var store = {};  
03:  
04:     return function (key, value){  
05:         if (value){  
06:             return store[key] = value;  
07:         } else {  
08:             return store[key]  
09:         }  
10:     }  
11: })();
```

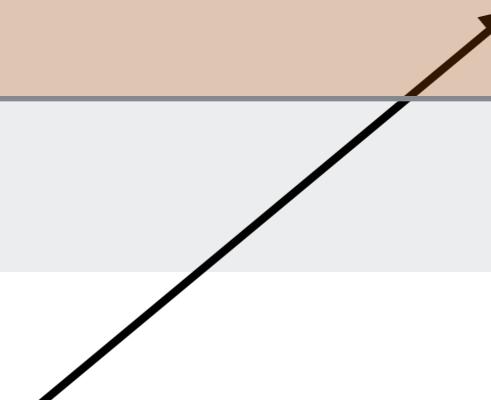
Self-invoking functions

Хитрый кунштюк

```
01: (function (){  
03  
03:     var a = 'Ничто не запретно';  
04:     var b = 'Все дозволено';  
05  
06: })();
```

Хитрый кунштюк

```
01: (function (){  
03:     var a = 'Ничто не запретно';  
04:     var b = 'Все дозволено';  
05:  
06: })();
```



Новый scope. Главное не забывать var.

Подъем (hoisting)

```
01: var a = 'Hello';
03: var b = a + c;
03:
04: var c = 'Wolrd';
```

Подъем (hoisting)

```
01: var a = undefined;  
02: var b = undefined;  
03: var c = undefined;  
04:  
05: a = 'Hello';  
06: b = a + c;  
07:  
08: c = 'Wolrd';
```

Подъем (hoisting)

```
01: var a = undefined;  
02: var b = undefined;  
03: var c = undefined;  
04:  
05: a = 'Hello';  
06: b = a + c;  
07:  
08: c = 'World';
```

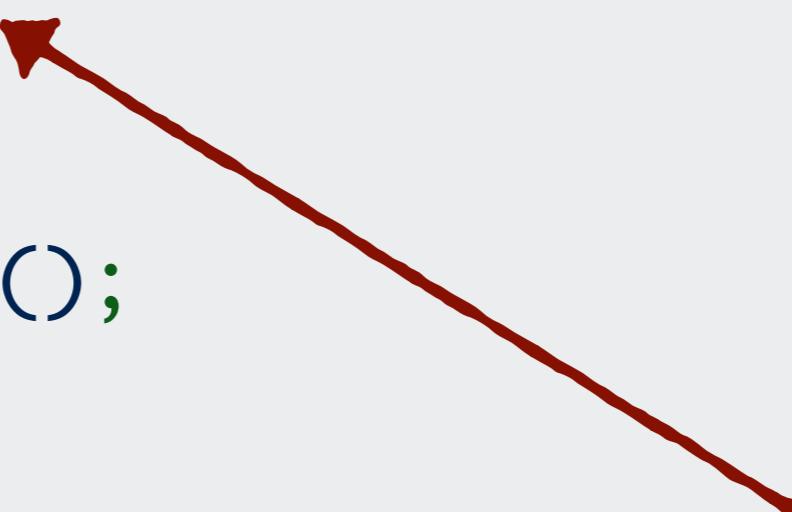
**хорошее решение - объявлять все
переменные в начале scope**

КОНТЕКСТ

```
01: function context() {  
02:     return this;  
03: }  
04:  
05: var ctx = context();  
06:  
07: alert(ctx);
```

КОНТЕКСТ

```
01: function context() {  
02:     return this;  
03: }  
04:  
05: var ctx = context();  
06:  
07: alert(ctx);
```



BOT OH



The page at <https://trello.com> says:

[object Window]

OK

КОНТЕКСТ

```
01: function context() {  
02:     return this;  
03: }  
04:  
05: var ctx = context.bind("Hello")();  
06:  
07: alert(ctx);
```



The page at <https://www.google.ru> says:

Hello

OK

Контекст

- bind - связывание контекста
- call или apply - вызов в определенном контексте

API языка

- таймеры

API языка

- таймеры

```
01: setInterval(function (){  
02:     console.log('Tick!');  
03: }, 1000);  
04:  
05: setTimeout(function (){  
06:     console.log('Опоздал на 5 секунд');  
07: }, 5000);
```

API языка

- таймеры
- глобальный this

API языка

- таймеры
- глобальный this
- конструкторы типов

API языка

- таймеры
- глобальный this
- конструкторы типов

```
01: new Function('a', 'b', 'return a + b')
02: new Number('666');
03: new String(666);
04: new Boolean(0);
05: new RegExp('^[a-b]*\d$');
06: new Array(1, 2, 3);
07: new Object();
```

API языка

- таймеры
- глобальный this
- конструкторы типов
- eval

API языка

- таймеры
- глобальный this
- конструкторы типов
- eval
- isNaN, isFinite, instanceof, isPrototypeOf

API языка

- таймеры
- глобальный this
- конструкторы типов
- eval
- isNaN, isFinite, instanceof, isPrototypeOf
- debug tools и объект console

Chrome Devtools

The screenshot shows the Chrome DevTools interface with the following sections:

- Elements:** Shows the DOM tree of the current page. A node under the body element is selected, highlighted in blue.
- Styles:** Displays the CSS styles applied to the selected element. It includes a list of rules and their corresponding properties and values. Some properties like font and color are shown with live preview swatches.
- Properties:** Shows the computed style for the selected element, including margin, border, padding, and width/height.
- Console:** A bottom panel for running JavaScript and viewing logs.

на этом все
спасибо!