

Web- developing

Vadym Savenko @2024

CHAPTER I. CREATION OF YOUR OWN WEB PAGES. (p. 4 – 19)

- 1.1. What does a web page consist of. (p. 4)
- 1.2. What is HTML? (p. 4)
- 1.3. Non-visual components. (p. 5 - 6)
- 1.4. Writing text. (p. 7 – 8)
- 1.5. Attributes. (p. 9 - 10)
- 1.6. Adding buttons, checkboxes, radio buttons, etc. (p. 10 – 12)
- 1.7. Adding images. (p. 13)
- 1.8. Creation of tables. (p. 13)
- 1.9. Creating numbered and bulleted lists. (p. 14)
- 1.10. Creating drop-down lists. (p. 15)
- 1.11. Adding video and audio content. (p. 15 - 16)
- 1.12. Using blocks: <div>; <footer>; <header>; etc. (p. 16 – 19)
- 1.13. Posting the site to the network. (p. 19)

CHAPTER II. STYLING OF PAGES. (p. 20 – 48)

- 2.1. What is CSS? (p. 20)
- 2.2. Using CSS on HTML. (p. 20 – 22)
- 2.3. Stylization of classes and id. (p. 23)
- 2.4. General options. (p. 23 - 29)
- 2.5. Display option. (p. 29 – 31)
- 2.6. Filters. (p. 32 – 34)
- 2.7. Flexbox. (p. 35 – 43)
- 2.8. Animation. (p. 44 – 47)
- 2.9. Adaptation. (p. 47 – 48)

CHAPTER III. SCRIPTS. (p. 49 – 69)

- 3.1. What is JavaScript? (p. 49)
- 3.2. Add and run scripts on a web page. (p. 49 - 50)
- 3.3. Events (p. 50 – 53)
- 3.4. The first scripts. (p. 54 - 57)
- 3.5. Operators, conditions, branching and loops. (p. 58 – 61)
- 3.6. Functions. (p. 62)
- 3.7. Random. (p. 63 - 64)
- 3.8. Web programming scripts and frameworks. (p. 66 – 69)

CHAPTER IV. DATABASE USE. (p. 70 – 87)

- 4.1. What is a database? (p. 70)
- 4.2. Relational and non-relational databases. (p. 70)
- 4.3. Using JSON with JavaScript. (p. 71)
- 4.4. What is SQL? (p. 72)
- 4.5. Data retrieval, sorting, conditions and operators. (p. 73 - 87)

This guide is related to web programming. In this book you will learn: creating web pages; stylization of pages; JavaScript programming; use the database using the SQL programming language. These languages are very popular in web development and the first thing web developers start learning is the use of HTML, which is found on all web pages.

CHAPTER I. CREATION OF YOUR OWN WEB PAGES.

Lesson 1.1. What does a web page consist of?

In general, pages consist of elements, styling and functionality. HTML hypertext markup language is responsible for objects, CSS style cascade language is responsible for styling, and programming language scripts are responsible for functionality.

Lesson 1.2. What is HTML?

HTML (Hyper Text Marker Language) is a hypertext markup language that is responsible for the presence of elements on the site. Here is the HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- <head> - the tag that is responsible for the
    presence elements of functions. Where lang="en" is
    an indication to the language of the page so that
    the browser understands which one language is
    written there.
    <!DOCTYPE html> - indication of use HTML version
    (In this case, it is: HTML 5.0) -->
  </head>
  <body>
    <!-- <body> - the tag that is responsible for the
    presence visual elements -->
  </body>
</html>
```

Where was it written?<!-- some text --> - this is a comment that does not affect the operation of the page.

It is also necessary to remember that: <html>; <body>; <head> is all tags. They can be with or without a closing tag. For example, tags: <body>; <head>; <html> - opening tags, and tags that have a "/" symbol at the beginning of their name, such as: </head>; </body>; </html> - closing tags. Do not forget about closing and opening tags.

Lesson 1.3. Non-visual components.

In general, non-visual components are those that we do not see on the web page, but perform a function, such as: import scripts; changing the general name of the page; saving information about the page; etc. To use non-visual components, they must be used in the <head> thesis. Here is the code using these components:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Setting the general
page names -->
    <title>Name of the page</title>

    <!-- Import styling -->
    <link rel="style sheet" href="PageStyle.css file">

    <!-- Setting the site icon -->
    <link rel=icon type="image/png" href="./assets/favicon.png">

    <!-- Information about the page that can be found by
keywords in the "content" attribute. Where is the attribute
"name" in which the value "keywords" is an indication of
performing the function of the <meta> tag. In this case -
the keyword search feature was mentioned
by search engine-->
    <meta name=keywords content="wikipedia, Metatag, article">

    <!-- Information about the name of the author of the page -->
    <meta name="author" content="Vadim Savenko">

    <!-- An analogue of "author" where you can also specify the language -->
    <meta name="copyright" lang="en" content="Vadim Savenko">

    <!-- Information about the page (Its description) -->
    <meta name="description" content="HTML Page">

    <!-- Information about the program used for
creation of this web page -->
    <meta name=generator content="Macromedia Dreamweaver 4.0">

    <!-- Page language information -->
    <meta http-equiv="content-language" content="en">

    <!-- Information about -->
    <meta http-equiv="Content-Style-Type" content="text/stylus-lang">
    <!--Continuation of the code on the next page -->
```

```

<!-- can be written like this -->
<meta http-equiv="Content-Style-Type" content="text/less">

<!-- Page encoding information -->
<meta charset="UTF-8">

<!-- Information that this page should be opened
in the frame -->
<meta http-equiv="Window-target" content="_top">

<!-- ICMB: allows you to bind the entire page to geos
coordinates -->
<meta name="ICBM" content="12.345, 67.890">

<!-- Mark the geo group. performs a similar task by allowing
clearly indicate the name of the settlement and the region -->
<meta name="geo.position" content="50.167958;-97.133185">
<meta name="geo.placename" content="Manitoba, Canada">
<meta name="geo.region" content="ca-mb">
</head>
<body>
<!-- <body> - the tag that is responsible for the presence
visual elements -->
</body>
</html>

```

These are all the non-visual components that everyone needs to know, but all the optional components have also been shown. Here is the code with the required components:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Name of the page</title>
    <link rel="style sheet" href="PageStyle.css file">
    <link rel=icon type="image/png" href="favicon.png">
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- <body> - the tag that is responsible for the
    presence visual elements -->
  </body>
</html>

```

Note! If <title> is not used, the general title of the page will be the name of the code file of that page.

Lesson 1.4. Writing text.

For a page, text is a visual component that has several characters (These are: letters; period; comma; semicolon; colon; etc.). The code with the text tags is on the next page.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- <head> - the tag that is responsible for the
         presence elements of functions (non-visual
         components).-->
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- Plain text -->
    <p>
      pppp p.
      pppp p.
    </p>

    <!-- Plain text -->
    <pre>
      pre pre pre.
      pre pre pre.
    </pre>

    <!-- 1st level header -->
    <h1>This page is about text.</h1>

    <!-- 2nd level header -->
    <h2>This page is about text.</h2>

    <!-- 3rd level header -->
    <h3>This page is about text.</h3>

    <!-- 4th level header -->
    <h4>This page is about text.</h4>

    <!-- Level 5 Title -->
    <h5>This page is about text.</h5>

    <!-- Level 6 Title -->
    <h6>This page is about text.</h6>
```

```

<!-- Hyperlink. Where is the data in "href",
there is a link to the --> page
<a href="https://website/">The page about text.</a>

<!-- Bold text -->
<b>This page is about text.</b>
<!-- Continuation of the code on the next page -->
<br> <!-- Moves elements and text to another line -->

<!-- Italic text -->
<i>This page is about text.</i>

<!-- Underlined text -->
<u>This page is about text.</u>

<hr> <!-- Creates a streak -->

<!-- Crossed out text -->
<s>This page is about text.</s>

<!-- Subline text -->
<sub>This page is about text.</sub>

<!-- Superscript -->
<sup>This page is about text.</sup>
</body>
</html>

```

If you check the code in the browser, there will be a difference between the tags:

ppppp.ppppp.

pre pre pre.
pre pre pre.

Ця сторінка про текст.

Ця сторінка про текст.

Ця сторінка про текст.

Ця сторінка про текст.

Ця сторінка про текст.

Ця сторінка про текст.

[Ця сторінка про текст.](#) **Ця сторінка про текст.**

Ця сторінка про текст. Ця сторінка про текст.

~~Ця сторінка про текст.~~ Ця сторінка про текст. Ця сторінка про текст.

Lesson 1.5. Attributes.

An attribute is a comprehensive property, its part, an appendix, necessary to ensure the integrity of an object, subject (subject). Attributes are data fields that relate to a file but are not part of it. They are not counted in the file size calculation and can be copied or deleted without changing the file itself. The system uses attributes to store, for example, file size, file type, last modified date, etc. It is the same in other operating systems. The difference is that you can add any attribute to any file, then display and make it editable in the Tracker window. You just need to determine what type of attribute you want to add to the file (for example: tape, integer and time), give a definition and description. The question is why the attribute is needed → it indicates the path or route of the browser, the attribute tells the tag where to place the text, what should be the list, etc. What are the attributes → specific, which is needed for a specific tag, and general or common, which is used for a paragraph, for a heading, for a table. Well, basically everything, there are attributes that can be added to any tag, and there are specific ones that are added to a specific tag, I will add that the "align" attribute is considered obsolete, but it can be said that the HTML hypertext markup language itself is obsolete, and not required when content management systems are present.

Attributes are always written inside a tag, followed by an equals sign and attribute details enclosed in double quotes. A semicolon after an attribute is used to separate commands of different styles. This is how a tag with attributes is written:

```
<tag attributeFirst="" attributeSecond=""></tag>
```

Note: A tag can have multiple attributes. Attributes are not required for use with tags, but there are some attributes that must be used for tag functionality or styling. Here are all the attributes we use in tags:

Attributes that all visual components have: class; id; style; width; . Take the <p> tag as an example. Here's how to write this tag using these attributes:

```
<p class="class_p" id="id_p" style="color: red;">Hello</p>
```

class— serves to specify the class for the tag.

id— serves to set the ID for the tag.

style— serves to specify the CSS style for the tag.

width— serves to set the width of the component as a percentage of the page size in the browser window or in pixels. For example: width="10px"; width="10%".

align— serves to display the component horizontally from the central (*align*="center"), right (*align*="right") or left side (*align*="left").

The src and href attributes: attributes that serve to link to a local file (including pages) or a network file:

```

```

Difference between src and href attributes:

- *src* used in tags: ; <iframe>; <video>; <source>.
- *href* used in tags: <a>; <meta>; <link>.

To refer to the address, it is necessary to specify as follows:

- **On the web page:**

```
href="https://on.web.page/"
```

- **To e-mail:**

```
href="mailto: mailuser2000@mail.com "
```

- **To phone number:**

```
href="tel:+380931112233"
```

Alt attributes: attribute that inserts text instead of an image if the image failed to load. This tag is only used with the tag. Here's his write-up with that tag:

```

```

Target attributes: attribute that is only used in the <a> tag and opens the link in a new tab if *target*=" _blank". Example:

```
<a href="google.com" target="_blank">Link</a>
```

Download attributes: attribute that is used only in the <a> tag and instructs the browser that the file link can only be downloaded. Example:

```
<a href="google.com/file.txt" download="file.txt">Link</a>
```

Attributes type and others: these attributes will be discussed in others later.

Lesson 1.6. Adding buttons, flags, radio buttons, etc.

The <button> tag is responsible for creating a button that performs a function when clicked. Here is the code writing with this tag:

```
<button>Button</button>
```

There is also an <input> tag, which is responsible for creating an input field. Here is the code writing with this tag:

```
<input title=Hello placeholder="Text" type=text>
```

In this case, the page will have:

But with this tag you can make not only buttons, but also other components using the type attribute. Here are examples of usage with the type attribute:

type	Components
text	Text field
"email"	Postal address field
"phone"	Phone number field
button	Button
check box	Flag
"number"	Numeric field
date	Date field
time	Time field
"datetime-local"	Date and time field
"password"	Password field
color	Color selection button
"file"	File selection button
radio	Switch (Radio Button)
"range"	Slider

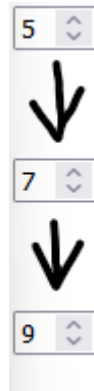
Also, the <input> tag has optional attributes:

- **min**—the minimum number of characters or values allowed in a component
- **max**—the maximum number of characters or values allowed in a component
- **step**—how many steps the slider or number field will move.

Using them in the page code:

```
<input min="5" max="9" step="2" type="number">
```

The result of the code in the browser:

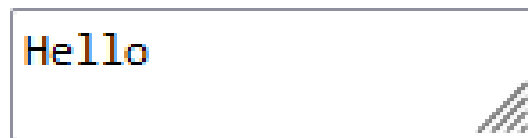


These attributes can be used with types: "time"; "range"; "number".

There is also a <textarea> tag that is responsible for creating an input field that can be resized. Here is its writing in the code:

```
<textarea title=Hi>Hello</textarea>
```

When opening the page in the browser:



Also, this tag has the "rows" attribute, which is responsible for the height of this tag by terms. If you do not specify this attribute, its height will be equal to two lines. Let there be a case when it is necessary to make the height of this component for 5 terms. Then, here is an example using this attribute:

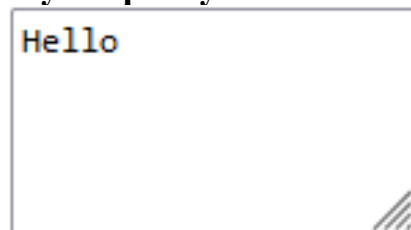
```
<textarea rows="5">Hello</textarea>
```

When opening the page in the browser:

If this attribute is not specified



If you specify this attribute



Lesson 1.7. Adding images.

To add images to an HTML page, you must use the tag, which has the alt and src attributes. Here is the code with this tag:

```
  

```

- *alt* serves to insert text instead of an image in the event that an attempt to load an image fails.
- *src* serves to set the image for the tag.

Lesson 1.8. Creation of tables.

Here is the code to create the table on the page:

```
<table>  
  <tbody>  
    <tr>  
      <th>1 column</th>  
      <th>2 column</th>  
      <th>3 column</th>  
    </tr>  
    <tr>  
      <th>4</th>  
      <th>5</th>  
      <th>6</th>  
    </tr>  
    <tr>  
      <th>7</th>  
      <th>8</th>  
      <th>9</th>  
    </tr>  
  </tbody>  
</table>
```

This is what will be on the page:

1 стовбець 2 стовбець 3 стовбець

4

5

6

7

8

9

Lesson 1.9. Creating numbered and bulleted lists.

To add images to an HTML page, you need to use the `` tag for numbered lists and `` to create bulleted lists:

```
<ol>
  <li><p>1 element</p></li>
  <li><h3>2 element</h3></li>
  <li>3 element</li>
</ol>
<ul>
  <li><p>1 element</p></li>
  <li><h3>2 element</h3></li>
  <li>3 element</li>
</ul>
```

This is what will be on the page:

1. 1 элемент
 2. **2 элемент**
 3. 3 элемент
- 1 элемент
 - **2 элемент**
 - 3 элемент

You can also set the start attribute for numbered lists, which corresponds to the number from which the element begins. Example:

```
<ol start="7">
  <li>1 element</li>
  <li>2 element</li>
  <li>3 element</li>
</ol>
```

This is what will be on the page:

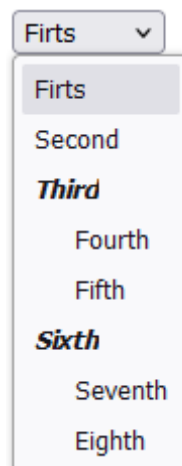
7. 1 элемент
8. 2 элемент
9. 3 элемент

Lesson 1.10. Creating drop-down lists.

To add images to an HTML page, use the <select> tag:

```
<select name="name" id="id">
  <option>Firts</option>
  <option>Second</option>
  <optgroup label="Third">
    <option>Fourth</option>
    <option>Fifth</option>
  </optgroup>
  <optgroup label="Sixth">
    <option>Seventh</option>
    <option>Eighth</option>
  </optgroup>
</select>
```

This is what will be on the page:



Lesson 1.11. Adding video and audio content.

To add audio files, use the <audio controls> tag, and to add video, use the <video controls> tag. Here is an example in this code:

```
<!-- Adding audio files -->

<!-- Adding .mp3 format -->
<audio controls>
  <source src="audio.mp3" type="audio/mp3">
</audio>

<!-- Continuation of the code on the next page -->
```

```

<!-- Adding .wav format -->
<audio controls>
  <source src="audio.wav" type="audio/wav">
</audio>

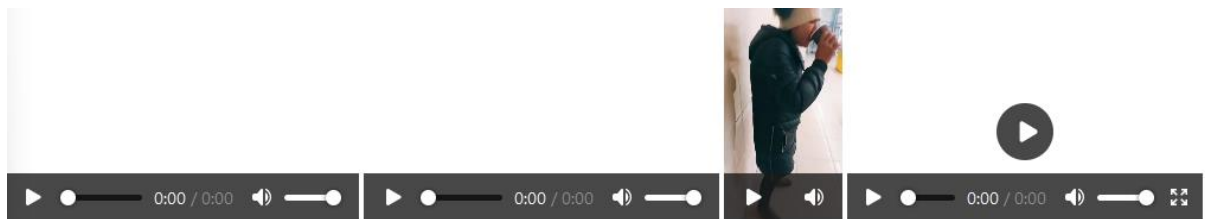
<!-- Adding video -->

<!-- Adding .mp4 format -->
<videocontrols>
  <source src="video.mp4" type="video/mp4">
</video>

<!-- Adding .avi format -->
<videocontrols>
  <source src="video.avi" type="video/avi">
</video>

```

This is what will be on the page:



You can also add videos from the YouTube service. You need to go to the desired video and click on the "Share" button and choose "Insert", where there will be an `<iframe>` tag with attributes and this code should be inserted into our HTML code.

Lesson 1.12. Using blocks: `<div>`; `<footer>`; `<header>`; etc.

HTML elements `<div>`, `<footer>`, and `<header>` are widely used to structure and design web pages. `<div>` (from "division") is a universal container used to group other HTML elements. It does not have any special meaning, but is often used to organize the page structure and apply CSS styles. Here is an example in this code:

```

<div style="width: 200px;">
  <h1>Header Header Header</h1>
  Main Content
</div>

<!-- Continuation of the code on the next page -->

```



```
<div style="width: 200px;">
  <h1>Header Header Header</h1>
  Main Content
</div>
```

This is what will be on the page:

Header
Header
Header

Main Content

Header
Header
Header

Main Content

But `<div>` is the only container in HTML, because there are also: `<nav>`; `<footer>`; `<header>`; `<section>`; `<article>`; `<aside>`; `<main>`; `<figure>`; `<figcaption>`; `<dialog>`.

- `<nav>`: Used to define navigation links. For example, the site menu.

```
<navigation>
  <street>
    <li><a href="#home">Main</a></li>
    <li><a href="#about">About us</a></li>
    <li><a href="#contact">Contacts</a></li>
  </street>
</navigation>
```

- `<footer>`: Represents the footer of a section or page. Often contains copyright, links to privacy policies, and other information.

```
<footer>
  <p>@2024 Company. All rights reserved.</p>
</footer>
```

- `<header>`: Used to represent an entry or group of navigation links. Usually includes a logo, header, or navigation links.

```
<header>
  <h1>Site title</h1>
  <navigation>
    <street>
      <li><a href="#home">Main</a></li>
      <li><a href="#about">About us</a></li>
      <li><a href="#contact">Contacts</a></li>
    </street>
  </navigation>
</header>
```

- `<section>`: Used to define sections in a document, such as chapters, tabs, etc.

```
<section>
  <h2>About us</h2>
  <p>Here you can add information about your company.</p>
</section>
```

- `<article>`: Represents a standalone piece of content that can be distributed independently. For example, a blog post or a news article.

```
<article>
  <h2>Title of the Article</h2>
  <p>The text of the article...</p>
</article>
```

- `<aside>`: Used for content that is indirectly related to the main content. For example, a sidebar with links or advertisements.

```
<aside>
  <h2>Side Panel</h2>
  <p>Additional content or advertising here.</p>
</aside>
```

- `<main>`: Used for the main content of the document. There should only be one `<main>` per page.

```
<main>
  <h1>Main Title</h1>
  <p>The main content of your page.</p>
</main>
```

- `<figure>`: Used to indicate independent content, such as an illustration, diagram, or photo, along with a description.

```
<figure> <!-- Continuation of the code on the next page -->
  
```

```
<figcaption>Image caption</figcaption>
</figure>
```

- <figcaption>: Used to add a caption to a <figure>.

```
<figure>
  
  <figcaption>Image caption</figcaption>
</figure>
```

- <dialog>: Used to create a dialog box or modal window.

```
<dialogue>
  <p>This is a dialog box.</p>
  <button>close</button>
</dialogue>
```

Lesson 1.13. Posting the site to the network.

The best and free service for uploading a website built from HTML pages is GitHub. First you need to register. After registering in this service, you need to create a public repository with a README file. After creating the repository, you need to click on "Add file" and on "Upload files", where there will be a transition to adding files. After going to the download page, you need to download all the files that are interconnected with the HTML pages. Next, you need to go to "Setting" and go to "", where there will be a drop-down list in which "None" is selected. It is necessary to choose "main" instead of "None" and press the "Save" button and wait for some time (Approximately, it is 5 minutes). After waiting for some time, you need to reload the site, in which a new link to the created site should appear.

But this is only uploading the site to the network, and if it is necessary for the user to be able to find the site through the search engine (), then it is necessary to connect the page to the search engine. Here is a guide on how to add a site link to the Google search engine (Link:<https://support.google.com/webmasters/>) and for Bing (Link:<https://www.bing.com/webmasters/help>).

These are not all the HTML tags that have been shown in this add-on, but these are the basic tags to know. All tags are available on the site [css.in.ua](https://css.in.ua/html/tags), where there are all tags, JavaScript scripts and CSS components for tag styling. It also shows all the attributes for each tag. Here is the link to the site:<https://css.in.ua/html/tags>. It also shows which tags can only be used in HTML 5.0, which can be used in all versions of HTML (Except 5.0, because they are not supported in this version), and which are supported in all versions of HTML.

II CHAPTER. STYLING OF PAGES.

Lesson 2.1. What is CSS?

If HTML serves to place elements on the page, then CSS (Cascade Style Sheet) is responsible for their styling. The CSS language is not a programming language either.

Lesson 2.2. Using CSS on HTML.

There are 3 ways to use styling in HTML pages:

1. **Built-in style (Using the style attribute):** in the style attribute, you can enter options for the tag. Example:

```
<div style="tag style"></div>
```

2. **Built-in style sheet (Using the <style> tag):** in the <style> tag, you can enter selectors and their options for tags, ids, and classes on the page. It can be in <body> or <head>. Example:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color:blue;
      }
    </style>
  </head>
  <body>
    <p>Hello</p>
  </body>
</html>
```

- 3.
4. **External style sheet (Import “.css” file):** we use the <link> tag. Example:

```
<link rel="style sheet" href="PageStyle.css file">
```

The style assignment consists of:

```
selector {
  option: value;
}
```

You can also make values through variables:

```
selector {  
  --change: value  
  option:var(--change);  
  option_two:var(--change);  
  
  /* At the same time, the two options will have the  
  same value */  
}
```

By the way! Where in the code: "/* At the same time, the two options will have the same value */", this line is a comment.

Let's do some styling for this code:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <link rel="style sheet" href="style.css">  
  </head>  
  <body>  
    <p>Hello</p>  
  </body>  
</html>
```

Now we need to make our <p> tag have a blue font of 14 pixels. Then we need to tell us the selector "p", because we need to do it for the <p> tag. This is how we will do it in the "style.css" code:

```
p {  
  /* Set color */  
  color:blue;  
  
  /* Set font size in pixels */  
  font-size:14px;  
}
```

Now on the page we will see:

Hello

You can also do it through variables, a method that will also produce the same result as the first:

```
p {
  --color:blue;
  --size:blue;
  color:var(--color);
  font-size:var(--size);
}
```

You can also not specify the color name in the value, but use the RGB function.

```
p {
  color:rgb(xxx, yyy, zzz);

  /* Or like this */
  color: #XXYYZZ;
  /*
  XX, xxx - Red shade
  YY, yyy - Green shade
  ZZ, zzz - Blue shade
  */
}
```

There is also an understanding of pseudo-classes. Here is their use in the style sheet:

```
selector:pseudo-class {
  option: value;
}
```

A pseudo-class is a state of a selector.

A selector is basically tags, id (#id) and classes (.class).

Here is a table of all base pseudo-classes:

Pseudo-class	Condition
hover	When the user pressed the cursor on the selector with the left mouse button
focus	When the user clicked on the selector with the right mouse button
active	When the user pressed and did not release the left mouse button on the selector
visited	When the user clicked on this selector and went to the page (Generally use this in the <a> tag)

Here is an example of using pseudo-classes:

```
p:hover{
  color:red;
}
```

Lesson 2.3. Stylization of classes and id.

We have a code that has tags: <h1>; <h2>; <p> which have class "cls" and id "i". Here he is:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="style sheet" href="style.css">
  </head>
  <body>
    <h1 class="cls" id="i">Hello</h1>
    <h2 class="cls" id="i">Hello</h2>
    <p class="cls" id="i">Hello</p>
  </body>
</html>
```

Now you need to make all the text tags red. You can do this:

```
h1, h2, p{
  color:red;
}
```

It is also possible in this way, but if they have a common class or id, then you can do it like this:

```
/* For classes */
.cls{
  color:red;
}

/* For id */
#i{
  color:red;
}
```

Lesson 2.4. General options.

Here is the code from the general options in CSS:

```
selector {
  /* Elements styling basics */

  /* Sets the text color */
  color:red;

  /* Continuation of the code on the next page */
}
```

```
/* Defines the font family */
font-family:'Segoe UI';

/* Font size (In pixels) */
font-size:ed;

/* Font thickness (Number) */
font-weight:100;/* 100 - Thin, the smallest thickness */
font-weight:400;/* 400 - Standard */
font-weight:600;/* 600 - Bold */

/* Font style (italic, normal or bold) */
font-style:italic;/* Italic font style */
font-style:normal;/* Normal font style */
font-style:bold;/* Bold font style */

/* Text decoration (underline, strikethrough) */
text-align:ed;

/* Text decoration (underline, strikethrough) */
text-decoration:underlined;/* underscore */
text-decoration:overline;/* strike out */

/* Change letter case */
text-transform:uppercase;/* All letters will be uppercase */
text-transform:capitalize;/* The first letter of each word will
                           be big */
text-transform:lowercase;/* All letters will become lowercase*/
text-transform:none;/* Leave the text unchanged as is default*/
text-transform:inherit;/* Inherits the property from parent
                        element */
text-transform:initial;/* Default value which defined by the
                        browser */
text-transform:unset;/* Resets the property to the value which
                        depends on the context */

/* Height of text line (In pixels) */
line-height:1.5px;

/* Spacing between characters (In pixels) */
letter-spacing:0.5px;

/* Interval between words (In pixels) */
word-spacing:2px;

/* Transparency */
opacity:0.7;/* 70% transparency */
/* Continuation of the code on the next page */
```



```
/* Blur level */
filter:blur(5px);/* Blur by 5 pixels */

/* Layout basics */

/* Defines the block type for the HTML element (block, inline,
flex, grid) */
display:block;/* Block element */
display:inline;/* String element */
display:flex;/* Flexible container */
display:grid;/* Mesh container */

/* Sets the width of the element (In pixels) */
width:10px;

/* Sets the height of the element (In pixels) */
height:10px;

/* Outer padding around the element (In pixels) */
margin:10px;

/* Internal padding around element content (In pixels) */
padding:10px;

/* Sets the border of the element (width, style, color) */
/* Color */
border:ed;
/* Width (In pixels) */
border:10px;
/* Style */
border:none;/* No border */
border:currentColor;/* The color of the border is equal text
                        color */
border:dashed;/* The border consists of short strokes */
border:dotted;/* The border consists of points */
border:double;/* Double border line */
border:groove;/* Appears to be cut out in the background */
border:hidden;/* The border is hidden (equivalent 'none') */
border:inherit;/* Inherits the border style */
border:inset;/* Appears to be drowned in the background */
border:initial;/* Initialize to value by default */
border:medium;/*Average border size (equivalent 3px by
                default)*/
border:output;/* Appears as convex over the background */
border:ridge;/* Looks like a crest on the background */
border:solid;/* Straight continuous border line */
/* Continuation of the code on the next page */
```

```
border:thick;/* Thick border */
border:thin;/* Thinner border */
border:unset;/* Reset to default */
/* Indents from the edges of the parent element for of
positioned elements (In pixels) */
top:10px;/* Indent from top */
right:10px;/* Indent from the right part */
bottom:10px;/* Indent from bottom */
left:10px;/* Indent from the left part */

/* Selector position */
position:absolutely;/* Element is positioned relative
nearest ancestor with non-static */
position:fixed;/* Element stays in one place
while scrolling */
position:relative;/* Element is positioned relative
his usual place */
position:static;/* The element is placed in the default
document threads */
position:sticky;/* Element is positioned relative
scrolls */
position:inherit;/* Element inherits position from
parent element */
position:initial;/* The element has a default value
(static) */
position:unset;/* Reset to a value that depends on
context */
/* Background basics */

/* Sets the text color */
background:ed;/* You can choose there
image or color */

/* Background color */
background-color:blue;

/* Background image */
background-image:url('image.png');

/* Method of repeating the background image */
background-repeat:no-repeat;/* No repetition */
background-repeat:inherit;/* Inherit */
background-repeat:initial;/* Initializes */
background-repeat:repeat;/* Repeats over the entire area */
background-repeat:repeat-x;/* Iterates only after
horizontal */
background-repeat:repeat-y;/* Iterates only after vertical */
```

```
background-repeat:round;/* Stretches to fit
into container */
background-repeat:space;/* Placed with spaces between
repetitions */
background-repeat:unset;/* Reset */

/* Background image position */
background-position:top left;/* Sets the position
image in the upper left corner */
background-position:top center;/* Positions the image in
top center */
background-position:top right;/* Positions the image in
upper right corner */
background-position:center left;/* Positions the image in
center left */
background-position:center center;/* Centers the image in
in the middle of the element */
background-position:center right;/* Positions the image in
center right */
background-position:bottom left;/* Positions the image in
lower left corner */
background-position:bottom center;/* Positions the image in
bottom center */
background-position:bottom right;/* Positions the image in
lower right corner */

/* Continuation of the code on the next page */

background-position:10px 20px;/* Positions the image on
distance 10px left and 20px
from above */
/* Default value: top left */

/* Background image size */
background-size:auto;
background-size:cover;
background-size:contains;
background-size:100px 200px;/* Sets the width and
image height.
100px - width,
200px height */
background-size:50% 40%;/* Sets the width and height
as a percentage of size
element */

/* Basics of working with text */

/* Continuation of the code on the next page */
```

```
/* Horizontal text alignment */
text-align:left;/* Aligns the text to the left edge */
text-align:center;/* Aligns the text to the center */
text-align:right;/* Aligns the text to the right edge */
text-align:justify;/* Justifies the text on both sides,
creating an even interval */

/* Text shadow */
text-shadow:none;/* No shadow */

/* Shadow size (X, Y, Blur, Color)
X - Move from the left side of the selector
Y - Offset from the top of the selector
Blur - Blur
Color - Shadow color*/
text-shadow:2px 2px 4px rgba(0,0,0,0.5);

/* Trim text that goes beyond the element */
text-overflow:clip;/* Trims the text that goes beyond
container bounds */
text-overflow:ellipsis;/* Displays the ellipsis "..."
at the end of overflow text */

/* How spaces and line breaks are handled */
white-space:normal;/* Trims spaces and carriage returns
default strings */
white-space:nowrap;/* Does not allow to wrap strings,
text continues in one line */
white-space:pre;/* Keeps all spaces and line breaks */

white-space:pre-wrap;/* Saves spaces,
but allows line breaks */
white-space:pre-line;/* Keeps spaces, but removes extra ones
spaces and saves line breaks */

/* Basics of working with a box model */

/* Algorithm for calculating the width and height of the element */
box-sizing:content-box;/* Calculates width and height
for content only */
box-sizing:border-box;/* Includes width and height
boundaries and indents in
total size */

/* Behavior of an overflow element
(visible, hidden, scroll, auto) */

/* Continuation of the code on the next page */
```

```

overflow:visible;/* Overflowed content is visible
outside element */
overflow:hidden;/* Overflowed content is truncated
and not visible */
overflow:scroll;/* Adds scrolling if content
goes beyond element */
overflow:auto;/* Adds scrolling only when
it is necessary */

/* Use gradient */
/* linear-gradient(to XXX, from_color, to_color)
XXX is the direction where the gradient should go:
right - to the right;
left - to the left;
top - up;
bottom - down;
right - to the right;
right - to the right; */
background:linear-gradient(to top,ed,yellow);
background:linear-gradient(yellow,ed,yellow,blue,green);
/*
to top - direction (Directions are always at the beginning,
and then color)

yellow, red, yellow, blue, green - set of colors,
which should be in the gradient */
}

```

Lesson 2.5. Display option.

Here are the codes for common options in CSS:

```

selector {
    display:block;/* Stands in the container and
the width of the selector depends on
the width of the container in which
be this selector */
}

```

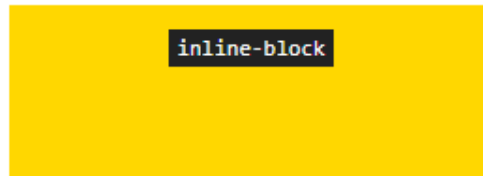
The browser will have:



The following code:

```
selector {  
    display:inline-block; /* Stands in the container and  
    the width of the selector depends on  
    selector width */  
}
```

The browser will have:



The following code:

```
selector {  
    display:none; /* The selector will be hidden */  
}
```

The browser will have:



The display property can receive 20 values:

block- The element is displayed as a block. By specifying this value for an element- it will start to behave as a block, that is, its content will always start with a new line.

inline- The element is displayed as a string (). The inline value cancels the feature of block elements always starting on a new line.

inline-block- This value creates a block element that behaves like a string element. Its inner part is formatted as a block element, and the element itself - as a line element. This is how the element behaves.

inline-table- Specifies that the element is a table<table>, but at the same time the table behaves as a line element and flows around other elements.

inline-flex- The element behaves as a string and displays the content according to the flex model. Appeared only in CSS3.

flex- The element behaves as a block and lays out the content according to the flex model

list-item- An element behaves the same as an element.

none- Hides an element. The layout is formed as if the element never existed.

You can make an element visible using scripts. At the same time, the data on the page is reformatted taking into account the newly added element.

run-in- Sets the element as block or string depending on the context.

table- Gives the table element to the element element<table>.

table-caption- Sets the table title like an application<caption>

table-cell- Indicates that the element is a table element (<td>or<th>)

table-column-group- An element behaves as if it were an element<colgroup>.

table-column- An element behaves as if it were an element<col>.

table-footer-group- Used to store one or more rows of cells that are displayed at the very bottom of the table. Its action is similar to work<tfoot>

table-header-group- The element is designed to store one or more rows of cells that are presented at the top of the page. Its action is similar to work<thead>

table-row- The element is displayed as a table row<tr>

table-row-group- The element is similar to the action of the element<tbody>.

[initial](#)- Sets the property to an unassigned value

[inherit](#)- Indicates the inheritance of a property from its parent element

Value without assignment: *inline*

Followed by: No

Animated: So

JavaScript syntax: object.style.display="none"

Lesson 2.6. Filters.

The filter CSS property specifies which visual effects to apply to the element, such as blur or contrast. It is most often applied to the element).

The filter property can receive 14 values:

`none`- There are no effects. No assignment.

`blur(px)` - Sets blur. The larger the value, the more blur. If no value is specified, 0 is used. A negative value is not allowed.

`brightness (%)` - Image brightness:

- 0% will make the image completely black.
- 100% (1) Image will remain unchanged. No assignment.
- A value greater than 100% will make the image lighter.

`contrast (%)` - Image contrast:

- A value less than 100% or 1 will decrease the contrast.
- 100% or 1 Image will remain unchanged. No assignment.
- A value greater than 100% or 1 will increase the contrast.

A negative value is not allowed. An empty value is treated as 1.

`grayscale (%)` - Will convert the image to black and white.

- 100% or 1 - Converts the image to black and white.
- 0% or 0- The image does not change.
- From 0% to 100% (or from 0 to 1) change the color scale of the picture.

Negative values are not allowed. An empty value is treated as 0.

`hue-rotate(deg)` - Changes the color of the image by rotating the shade on the color wheel.

- 0 or 360 degrees leaves the image unchanged.
- Any other values will change the color scheme.

Negative values are allowed. The maximum value is 360deg. An empty value is treated as 0 deg.

`invert (%)` - Inverts the colors in the image. In its action, it is similar to turning a photo into a negative.

- 0% or 0 - The image will be unchanged.
- 100% or 1 - Completely inverts the colors of the picture.
- 0% to 100% or 0 to 1 - Partially inverts the colors.

Negative values are not allowed. An empty value is treated as 0.

`opacity (%)` - Sets the degree of transparency of images. It works similar to the opacity property, but some browsers use hardware acceleration for filters to improve their performance.

- 100% or 1 Image will remain unchanged.
- 0% or 0 The image will become completely transparent.
- 0% to 100% or 0 to 1 sets the degree of transparency of the image.

A negative value is not allowed. An empty value is treated as 1.

`saturate (%)` - Saturation of colors in the image.

- 0% or 0 removes the saturation of colors in the image, turning it into black and white.
- 100% or 1 leaves the image unchanged.
- 100% and more makes images more saturated.

A negative value is not allowed. An empty value is treated as 1.

`sepia (%)` - Converts the image to sepia - the so-called black and white image with a brown tint. Sepia gives photos a vintage look:

- 0% or 0 The image does not change.
- 100% or 1 Sepia at maximum strength.
- 0% to 100% or 0 to 1 linearly applies sepia.

Negative values are not allowed. An empty value is treated as 0.

`url ()` - Accepts the location of the XML file that defines the SVG filter. A link can also include an anchor for a specific filter. Example:`filter: url(svg-url#element-id)`

[initial](#) - Sets the property to an unassigned value

[inherit](#) - Indicates the inheritance of a property from its parent element

`drop-shadow(h-shadow v-shadow blur spread color)`

A shadow effect is applied to the image.

Possible values:

- `h-shadow`- Definitely. Specifies the width of the horizontal shadow. Negative values will display a shadow to the left of the image.
- `v-shadow`- Definitely. Sets the pixel value for the vertical drop shadow. Negative values will display a shadow over the image.
- `blur`- Not necessarily. This is the third value and must be in pixels. Adds a shadow blur effect. A higher value will create more blur (the shadow becomes bigger and lighter). Negative values are not allowed. If no value is specified, 0 is used (no shadow will be visible).
- `spread`- Not necessarily. This is the fourth value and must be in pixels. A positive value will cause the shadow to expand, while a negative value will cause the shadow to contract. If this value is not specified, it will be 0 (the shadow will be the same size as the element). Note: Chrome, Safari, Opera and possibly other browsers do not support this value.
- `color` is optional. Adds color to the shadow. If not specified, the color depends on the browser (most often, the shadow will be black).

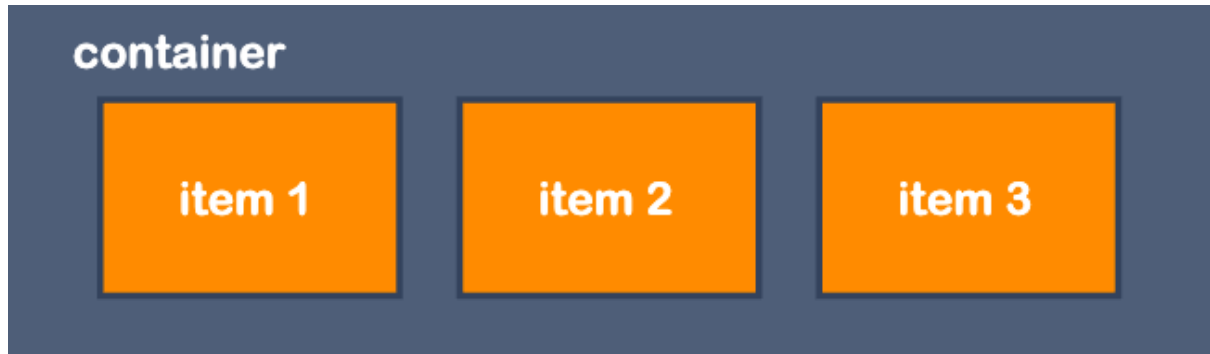
An example of creating a red shadow that is 8px wide and 8px high with a 10px blur effect:

```
filter: drop-shadow(8px 8px 10px red);
```

Hint: This filter is similar to a `propertybox-shadow`.

Lesson 2.7. Flexbox.

Flexbox, short for Flexible Box Layout, is a powerful CSS layout module designed to optimize the alignment, distribution, and organization of elements inside a container.



The flexbox layout model works on a parent-child basis. The parent, called the flex container, is responsible for dictating the layout of its child elements, known as flex elements. Using a number of flexbox properties, we can precisely control the size, position, and order of flex elements in a flex container.

Note! Inside a flex container, many of the normal positioning rules don't apply as expected. Let's look at the following list:

- Working with flex elements is different from working with inline or block-level elements.
- Unlike block-level elements, flex elements do not overlap other elements and do not overlap each other.
- Margins applied to the edges of a parent element do not extend beyond its borders.

flex model

Since we are not dealing with inline or block elements, document flow in the flex model is defined by the axes of the flex container along which all elements are aligned. To trigger flexbox behavior, we apply the `display: flex` property; to the parent containing all elements.

We can influence the direction of flex elements. By default, flex elements are arranged in a row. However, we can change this behavior using the `flex-direction` property. Let's compare document flow without flex and with flex:

CSS code:

```
/* Apply flex to the parent element.
/* In this situation, it is the <ul> element with the
"list" class name.
*/
.list {
    display: flex;
}

/* Add border to the items of the flex container */
.list li {
    border: 1px solid forestgreen;
}

/* Remove default decoration */
Street {
    list-style: none;
}
```

HTML code:

```
<html lang="en">
  <head>
    <link rel="style sheet" href="index.css" />
  </head>
  <body>
    <!-- Default document flow -->
    <h4>Default</h4>
    <street>
      <li>Lorem ipsum dolor sit amet.</li>

      <li>Lorem ipsum dolor sit amet consectetur.</li>
      <li>Lorem ipsum dolor sit amet.</li>
    </street>

    <!-- Applied flexbox -->
    <h4>Flex</h4>
    <street class="list">
      <li>Lorem ipsum dolor sit amet.</li>
      <li>Lorem ipsum dolor sit amet consectetur.</li>
      <li>Lorem ipsum dolor sit amet.</li>
    </street>
  </body>
</html>
```

Result on the page:

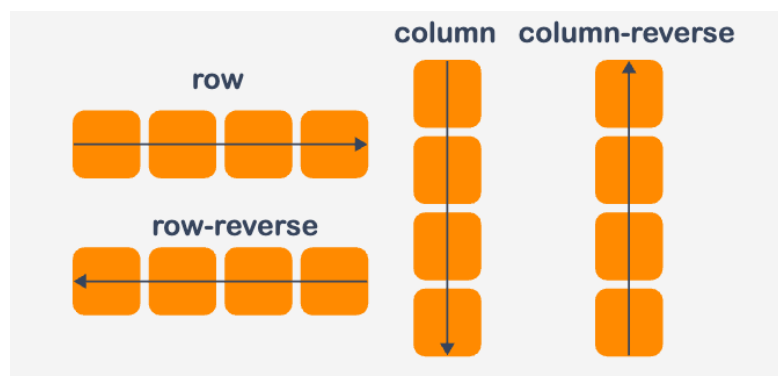
Default

- Lorem ipsum dolor sit amet.
- Lorem ipsum dolor sit amet consectetur.
- Lorem ipsum dolor sit amet.

Flex

The flex-direction property allows us to define the direction in which flex elements are located. By default, this property is set to row, which results in left-to-right layout in browsers that default to English. Here are the values you can insert to set the value: flex-direction: row | column | row-reverse | column-reverse;

Here's how elements are positioned using flex-direction:



Consider the following site navigation as an example

Here is the HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="style sheet" href="index.css" />
  </head>
  <body>
    <street class="navigation-list">

<!-- Continuation of the code on the next page -->
```

```

        <li class="navigation-item">
        <a href="#" class="navigation-link">Home</a>
        </li>
        <li class="navigation-item">
        <a href="#" class="navigation-link">About</a>
        </li>
        <li class="navigation-item">
        <a href="#" class="navigation-link">Price</a>
        </li>
        <li class="navigation-item">
        <a href="#" class="navigation-link">Team</a>
        </li>
        <li class="navigation-item">
        <a href="#" class="navigation-link">Support</a>
        </li>
    </street>
</body>
</html>

```

And this is the CSS code:

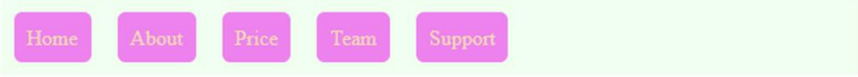

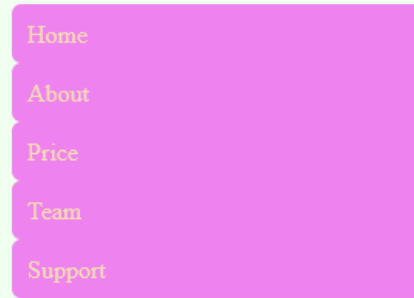

```

.navigation-list{
    background-color:honeydew;
    display:flex;
    padding:10px;
    flex-direction:XXX;
}
.navigation-item{
    padding:10px;
    border-radius:6px;
    background-color:violet;
}
.navigation-item:not(:last-child) {
    margin-right:20px;
}
.navigation-link{
    color:wheat;
}

street{
    list-style:none;
}
a{
    text-decoration:none;
}

```

Where there was XXX, we will indicate the value there. Here is the table when XXX is equal to some value:

When XXX is:	Result
row	
row-reverse	
column	
column-reverse	

align-items

The align-items property in CSS is used to vertically align items in a display: flex or display: grid container. For tables, this property is also not directly applicable, but we can use Flexbox or Grid inside the table cells to achieve the desired alignment: align-items: stretch | center | flex-start | flex-end | baseline;

This is how items are positioned using align-items:

stretch



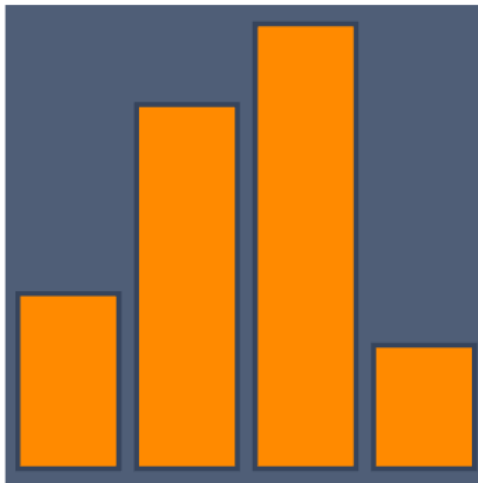
center



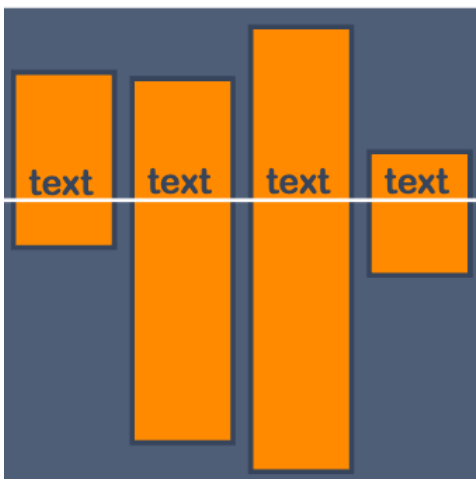
flex-start



flex-end



baseline



Now let's make the codes:

HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="style sheet" href="style.css" />
  </head>
  <body>
    <street class="cards-list">
      <li class="card">
        
        <p>Lorem, ipsum dolor.</p>
      </li>
      <li class="card">
        
        <p>
          Lorem ipsum dolor sit, amet consectetur.
        </p>
      </li>
      <li class="card">
        
        <p>Lorem ipsum dolor sit amet.</p>
      </li>
      <li class="card">
        
        <p>
          Lorem ipsum dolor sit, amet consectetur.
        </p>
      </li>
      <li class="card">
        
        <p>
          Lorem ipsum dolor sit, amet consectetur.
        </p>
      </li>
    </street>
  </body>
</html>
```

CSS:


























```
.cards-list{
  display:flex;
  border:1px solid khaki;
  align-items:XXX;
  width:660px;
}

.card{
  border:1px dashed navy;
  padding:10px;
  width:120px;
}

.card img{
  display:block;
  margin:0 auto;
}

/* Remove default decorations */
street{
  list-style:none;
  padding-left:0;
}
```

Where there was XXX, we will indicate the value there. Here is the table when XXX is equal to some value:

When XXX is:	Result				
scratch	 <p>Lorem, ipsum dolor.</p>	 <p>Lorem ipsum dolor sit amet consectetur adipiscing elit.</p>	 <p>Lorem ipsum dolor sit amet.</p>	 <p>Lorem ipsum dolor sit, amet consectetur adipiscing elit.</p>	 <p>Lorem ipsum dolor sit amet consectetur adipiscing elit.</p>
center	 <p>Lorem, ipsum dolor.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>	 <p>Lorem ipsum dolor sit amet.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>
flex-start	 <p>Lorem, ipsum dolor.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>	 <p>Lorem ipsum dolor sit amet.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>
flex-end	 <p>Lorem, ipsum dolor.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>	 <p>Lorem ipsum dolor sit amet.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>
baseline	 <p>Lorem, ipsum dolor.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>	 <p>Lorem ipsum dolor sit amet.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>	 <p>Lorem ipsum dolor sit, amet consectetur.</p>

Lesson 2.8. Animation.

Sometimes we need to create animations for changing elements on a web page with controlling factors such as speed, delay time, and duration. In such cases, the "transition" property can be used to perform this task. An element always has two states: initial and final. We can control the change behavior of an element using the following properties:

```
.transitionClass{
    /* The name of the CSS property that we
    must animate (The value must
    have the name of the option that should
    change when status changes
    selector) */
    transition-property:color;

    /* Time for which we need
    change the state of an element. It is indicated
    in seconds (s) or milliseconds (ms) */
    transition-duration:300 ms;
    /* It sets the speed curve of the effect
    transition Typical values are:
    ease; linear; ease-in; ease-out; ease-in-out.*/
    transition-timing-function:ease;

    /* Time delay before transition effect starts.
    It is specified in seconds (s) or
    milliseconds (ms). */
    transition-delay:2s;

    /* 1 second = 1000 milliseconds */
}
```

Here is an example of a page that has a container (Tag <div>), and when the mouse cursor approaches it, its color will change:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <style>
            div{
                width:400px;
                height:100px;
                /* Continuation of the code on the next page */
```

```

        border-radius:10px;
        border:2px solid orange;
        background-color:yellow;
        transition-property: background-color;
        transition-duration:300 ms;
        transition-timing-function:ease-in-out;
        transition-delay:100ms;
    }

    div:hover{
        background-color:darkblue;
    }
</style>
</head>
<body>
    <div></div>
</body>
</html>

```

If it is necessary to change the value of all options, then it is necessary to specify:

```
transition-property:all;
```

In CSS, the transition-timing-function property defines how the animation speed changes over time. Here is an explanation of the main values:

1. **ease:**

- This is the default value.
- The animation speed first increases, then slows down.
- Creates the effect of speeding up at the beginning and slowing down at the end.
- The speed graph is S-shaped.

2. **linear:**

- The animation speed is constant throughout.
- There is no acceleration or deceleration.
- The speed graph is a straight line.

3. ease-in:

- The animation speed increases at the beginning, then reaches a constant value.
- Creates an acceleration effect at the start.
- The speed graph starts from zero and accelerates.

4. ease-out:

- The animation speed is constant at first, then slows down to the end.
- Creates a slow-down effect at the end.
- The speed graph starts at speed and slows down to the end.

5. ease-in-out:

- The animation speed first increases, then slows down.
- Creates the effect of speeding up at the beginning and slowing down at the end.
- The speed graph is S-shaped, but more pronounced than that of ease.

You can also simply write transition in the style table:

```
transition:all 1200ms ease-in-out 250 ms;  
/* 1200ms - the time duration; 250ms - the delay */
```

There's also an animation that tells the selector to spin along the time arrow:

```
/* Do this 5 times */  
animation-iteration-count:5;  
  
/* Make it infinite */  
animation-iteration-count:infinite;
```

There is also an animation-direction option, which is applied like this:

```
/* element moves from left to right,  
repeats again from the beginning. */  
animation-direction:normal;  
/* element moves from right to left,  
repeats again from the end */  
/* Continuation of the code on the next page */
```

```
animation-direction:reverse;

/* element moves from left to right,
then right to left, and so on */
animation-direction:alternate;

/* element moves from right to left,
then from left to right, and so on */
animation-direction:alternate-reverse;
```

Lesson 2.9. Adaptation.

Site adaptation can include various aspects such as design, functionality, performance and compliance with modern standards. Responsive design ensures the correct display of the site on various devices, from mobile phones to desktop computers. Optimizing site loading speed includes using caching, reducing image size, minifying CSS and JavaScript files. Making the site accessible to people with disabilities includes using appropriate ARIA attributes, ensuring high contrast text, and supporting keyboard navigation. Adaptation of the site to support different languages includes the use of frameworks for localization and content translation.

Here is the syntax when using adaptation:

```
@media <media_type> <operator> (<media_feature>) {
    /* CSS rules */
}
```

1. <media_type> - declares the device type. Possible values:
 - all is the default value. If nothing is specified, the media rule works for all devices;
 - print - the media rule works for devices that produce printed documents, for example, printers;
 - screen - the media rule works for devices that produce printed documents, for example, printers.
2. <media_feature> - declares the characteristics of the device. The most common application:
 - min-width: - the minimum width of the viewing window;
 - max-width: - the maximum width of the viewing window.

3. <operator> - media-type and media-feature are separated by an optional logical operator. Its values can be: and or or,.

```
/* 1st example */
@media screen and(max-width:1200px) {
    .container{
        color:aliceblue;
    }
}

/* 2nd example */
@media screen and(min-width:1680px) {
    .container{
        color:gainsboro;
    }
}

/* In the first example, we told the browser
that for any screen width devices
less than or equal to 1200 pixels
apply the color property with a value
aliceblue to an element with a class name container.
(Continuation of the code on the next page)
In the second example, we told the browser
that if the screen width of any device exceeds
1680px or equal to 1680px property
of color with the value of gainsboro should be applied
to an element with the name of the container class. */
```


III CHAPTER. SCRIPTS.

Lesson 3.1. What is JavaScript?

JavaScript (JS) is a dynamic, object-oriented, prototype programming language. Implementation of the ECMAScript standard. It is most often used to create web page scripts, which provides the ability on the client side (end user device) to interact with the user, control the browser, asynchronously exchange data with the server, change the structure and appearance of the web page.

JavaScript is classified as a prototype (a subset of object-oriented), scripting programming language with dynamic typing. In addition to prototyping, JavaScript also partially supports other programming paradigms (imperative and partially functional) and some corresponding architectural properties, including: dynamic and weak typing, automatic memory management, prototypical inheritance, functions as first-class objects.

If we studied the languages HTML and CSS, which are not programming languages, then JavaScript belongs to programming languages, and HTML and CSS belong to languages that build graphics, because they cannot perform loops and in these languages it is not possible to make functions that only a programming language such as JavaScript can do.

Lesson 3.2. Add and run scripts on a web page.

There are 3 ways to add JS scripts:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- 1 way - adding <script> --> tag
    <script>
      console.log("Hello World");
    </script>

    <!-- Method 2 - adding a link to script to the
    <script> --> tag
    <script src="path/to/script.js"></script>
    <!--Continuation of the code is on the next page -->
    <!-- Method 3 - adding a link to script to the
    <script> tag from the Internet -->
```

```
<script
  type="application/javascript"
  src="http://page.with.script/"
></script>
</body>
</html>
```

Lesson 3.3. Events

Events are attributes in tags that execute a script when the tag state changes. Here's the code with the basic event attributes you need to know:

```
<!-- When clicking on the --> tag
<a onclick="console.log('Hello World');"></a>

<!-- Called after element loading complete -->
<a onload="console.log('Hello World');"></a>

<!-- Called when the page is unloaded or the browser window has
been closed. -->
<a onunload="console.log('Hello World');"></a>

<!-- Script executed when element loses focus. -->
<a onblur="console.log('Hello World');"></a>

<!-- Called when an element receives focus. -->
<a onfocus="console.log('Hello World');"></a>

<!-- Called when the user types something into the search field
(for <input type="search">) -->
<a onsearch="console.log('Hello World');"></a>

<!-- Called after any text has been selected in the element.-->
<a onselect="console.log('Hello World');"></a>

<!-- Called when the form is sent -->
<a onsubmit="console.log('Hello World');"></a>

<!-- The event is called when the user clicks
(presses and does not release) the key. -->
<a onkeydown="console.log('Hello World');"></a>
<!-- Called when the user pressed the --> key
<a onkeypress="console.log('Hello World');"></a>

<!--Continuation of the code is on the next page -->
```

```
<!-- Called when the user releases the --> key
<a onkeyup="console.log('Hello World');"></a>

<!-- Occurs when double-clicking with the left button
mouse on the element -->
<a ondblclick="console.log('Hello World');"></a>

<!-- Called when the user clicks the left mouse button on the
element. -->
<a onmousedown="console.log('Hello World');"></a>

<!-- Called when the mouse cursor moves over the --> element
<a onmousemove="console.log('Hello World');"></a>

<!-- Called when the cursor moves outside the --> element
<a onmouseout="console.log('Hello World');"></a>

<!-- Executed when the cursor hovers over the --> element
<a onmouseover="console.log('Hello World');"></a>

<!-- Called when the user releases the mouse button -->
<a onmouseup="console.log('Hello World');"></a>

<!-- Called when the user copies the contents of an --> element
<a oncopy="console.log('Hello World');"></a>

<!-- Called when the user cuts the contents of an --> element
<a oncut="console.log('Hello World');"></a>

<!-- Called when the user inserts content into the --> element
<a onpaste="console.log('Hello World');"></a>

<!-- Executed when interrupted by some event -->
<a onabort="console.log('Hello World');"></a>
```

These are the events you need to know. They work on all versions of HTML. But there are HTML events that only work on HTML 5.0. A table of events that only work in HTML 5.0 is on the next page.

Events	Tag status
onplay	The script is called when the media data is ready to start playing.
onafterprint	The script is executed only after the document is printed.
onbeforeprint	The script is executed before the document is printed.
onbeforeunload	The script is executed when the document is about to be downloaded
onhashchange	The script is executed when there have been changes to the anchor part of the URL
onmessage	The script is executed when the message is called.
onoffline	Fires when the browser starts working in offline mode
ononline	Triggers when the browser starts working in online mode.
onpagehide	The script is executed when the user navigates to another page page.
onpageshow	The script is executed when the user enters the page.
onpopstate	The script is executed when the history of one window is changed.
onresize	The script is executed when the browser window is resized.
onstorage	The script is executed when the Web Storage content is updated.
oncontextmenu	The script is executed when the context menu is called.
oninput	The script is called when the user enters data in the field.
oninvalid	The script is executed when the element is invalid.
onreset	Called when the Reset button is clicked on the form.
ondrag	Periodically called during the drag operation.
ondragend	Called when the user releases the relegated element.
ondragenter	Called when a draggable element enters the target area.
ondragleave	Called when the draggable element leaves the destination area.
ondragover	Called when the draggable element is in the destination area.
ondragstart	Called when the user starts dragging an element or selected text.
ondrop	Called when a draggable element drops to the destination area.
onscroll	Called when scrolling the content of an element (or web page).
onwheel	Called when the user scrolls the mouse wheel.

oncanplay	The script is executed when the file is ready to start playing (when it is buffered enough to start playing)
oncanplaythrough	The script is executed when the content can already be played without interrupting for buffering.
oncuechange	The script is executed when the cue in the <track> element changes
ondurationchange	Called when the length of the media file changes.
onemptied	Called when access to media content is interrupted (connection to the network is lost).
onended	Called when the media element has completely reproduced its content.
onshow	Called when the <menu> element will be displayed as a context menu.
loadedmetadata	The script is executed when the metadata (sizes or duration) is loaded.
loadeddata	Called when media data is loaded.
onloadstart	Called when the browser just starts downloading media data from the server.
onpause	Called when playback of media data is paused.
onplaying	Called when playback of media data is started.
onprogress	The onprogress event occurs when the browser loads the specified audio / video.
onratechange	Called when media data playback speed changes.
onseeked	Called when the sought attribute of the audio or video tag changes from true to false.
onseeking	Called when the seeking attribute of the audio or video tags changes from false to true
installed	The script is executed when the browser cannot receive the media data for any reason.
onsuspend	The script is executed when, for any reason, the data download is paused until it is fully loaded.
ontimeupdate	Called when the playback position of the <audio> or <video> element has changed.
onvolumechange	Called when the sound volume changes.
onwaiting	Called when the next frame in media playback is not available, but the browser expects it to load soon.
ontoggle	Called when the user opens or closes the <details> element.
onerror	Called if an error occurred while loading the element.

Lesson 3.4. The first scripts.

Where the team was mentioned: `console.log("Hello World");`, this command is used to display the value in the console. Here are all the commands you can use:

```
console.log("Hello World");// Display the value in the console
/*
Where "Hello World" is the value.
Where; - it means that the command is finished. The sign
";" necessarily
must be set after the command was written.
Where "/" or "/* text */" are comments.
"/" is a single-line comment, and "/* text */" is a
multi-line comment.
*/

alert("Hello World");// Display the message in the browser */
window.alert("Hello World");// Same as "alert("Hello World");"
return confirm("Hello World");// Display the message in the
browser with "Yes" or "No" buttons */
```

JavaScript, like all programming languages, has variables. Variables serve to store values in themselves. Values can be stored in a constant that never changes, or in a variable that can change. Here is an example of creating variables and constants:

```
const cns = "Hello World";// Created constants
let var = "Hello World";// Created variable
/*
const, var and let are declarations for creating
variables.
"Hello World" - variable values.
cns and var are the names of the variables
The name of the variables can be anything, but cannot
be called official words such as: let,
class, return and function. Non-English letters in names
variables are allowed but not recommended.
The variables named "apple" and "APPLE" are two different
variables
variables, because case matters. Neither are the variables
can have signs (Such as: ., ., ?)
*/

// You can do it like this:
```

```

let var2;
var2 = Hi;

// This is how you can change the values of variables
let var3 = Hello;// Creation of a new change
// setting value for new
// variable
var3 = Hi;// Change the value of the variable

```

All variable values of this code are text. JavaScript has the following value types:

The value type	Example
string (text)	"Hi"; "Text"; "Hello World"
number (numerical)	5; 6.24; 100.00; 3.6; 9; 10
boolean (True)	true or false
array (Lists)	[3, "Hi", "World", false]

Here is the code with variables of different value types:

```

let str = Hello;
let num = 5.4;
let bool = true;
let arr =[Hi, ["Hello World",4,true],3,false];

/* array (List) - the type of value that can fit in itself several
values (including variables, because a variable works as a reference
to a value). If we make a list of the variables that are in this
code, such as: str; num; bool; arr, then in the code we will write
as:
[str, num, bool, arr], then JavaScript will understand it,
like: ["Hello", 5.4, true, ["Hi", ["Hello World", 4, true], 3,
false]]

To find out what type of value a variable has, you can do it like
this: */
let typeOfStr = typeof str;/* typeof helps to find out the type of a
variable's value.

We may not know this because we stored the information in a
variable. In order to see this information, it is necessary to
display the value of this variable in the console: */
console.log(typeOfStr)

// Continuation of the code on the next page

```

```

/* It remains for us to learn about variables - it is about
selecting an element from the list. To select an element from the
list, we use such an understanding as an index. An index is an
element of an enumerated set that points to a specific array
element. This is usually a non-negative integer. In some languages,
negative subscripts are used to count elements backwards (from the
end of the array). To select the last item from the list, use index
[-1], use [-2] for penultimate, use [-3] for penultimate, etc. This
refers to choosing an element from the list from the last to the
first. To find elements from the beginning to the end of the list,
we subtract 1 from the element's normal ordering number. For
example: the first element is [0], the second element is [1], the
third element is [2], etc. Here is the list variable and the
variables that show the elements by index: */
let arrExample = ["First", "Second", "Third", "Fourth"];
let var1 = arrExample[0]; // Stores the value in a variable like:
"First"
let var2 = arrExample[2]; // Stores the value in a variable like:
"Second"
let var3 = arrExample[-1]; // Stores the value in a variable like:
"Fourth"
let var4 = arrExample[-3]; // Stores the value in a variable like:
"Second"
arrExample[2] = "3rd"; // Changes the element at index [2] to "3rd"
let var5 = arrExample.push("First"); // Creates the "First" element
// at the end of the list
let var8 = arrExample.unshift("Second"); // Creates the "Second"
element
// at the beginning of the list
let var6 = arrExample.pop(); // Stores the value of the last element
// and removes the last element of the list
let var7 = arrExample.shift(); // Stores the value of the first
element
// and removes the first element of the list

```

Now let's go to the dialog message with the "OK" and "Cancel" buttons that appears in the browser:

```

let var1 = confirm("Question"); // Creating a dialog message
// with buttons "OK" and "Cancel"
alert(var1); /* If the user clicked on "OK", then
var1 = true if user clicked on "Cancel" then
var1 = false */

```

Now let's analyze the use of elements with id. In general, classes are used for CSS, and ids for JavaScript. But ids can be used in CSS and in JavaScript, and

classes can be used in CSS and in JavaScript. Here's how you can get tags via id or via classes:

```
<!DOCTYPE html>
<html>
  <head><meta charset="UTF-8"></head>
  <body>
    <p id="blinking" class="cls">
      Text
    </p>
    <script>
      // Takes tags with the id specified in the function:
      const p=document.getElementById("blinking")

      // Takes tags from the class specified in the
      function:
      const p2=document.getElementsByClassName("cls")

      setInterval(function() {
        if(p.style.fontSize!="10px") {
          p.style.fontSize="10px" // Change the font
          p.style.color="Red" // Change the color
          p.textContent=hi // Change Text
        } else{
          p.style.fontSize="20px"
          p.style.color="Yellow"
          p.textContent=Hello
        }
      },XXX)/* setInterval executes a function or command
      constantly and after the completed command, waits
      XXX milliseconds, and then does this one
      function or command */
      p.remove();// Remove element with id="blinking"
    </script>
  </body>
</html>
```

Lesson 3.5. Operators, conditions, branching and loops.

Operators are divided into basic and logical operators. Boolean operators are usually used with Boolean (Boolean) values; however, the value they return is also a boolean. Here is a table with logical operators:

Operator	Audit
==	Is equal to
===	Strictly equal
!=	Not equal to
!==	Strictly not equal
>	Less
<	More
>=	Less than or equal to
<=	Greater than or equal to
	OR
&&	AND
!	NO

Here are the Boolean values in the variables:

```
let var1 = 3 > 5 || 6 >= 4; // true
let var2 = 5 === "5" || 6 === "6"; // false
let var3 = 5 == "5" || 6 == "6"; // true
let var4 = 3 > 5 && 6 >= 4; // false
let var5 = !(3 > 5 || 6 >= 4); // false
```

This is how variables can store values that can work as conditions.

By the way! As you have noticed, there is an operator "Equals" and "Strictly equals". The difference between these two operators is that the Equals operator can compare values without paying attention to the different types of values, while the Strictly Equals operator will also compare the type of values. A similar difference exists between the Not Equal and Strictly Not Equal operators.

Now consider branching on JavaScript. Branching is a scheme of algorithms in which, if the condition is true, then it is done according to the condition, and in the other case, another action will be performed. This is what branching looks like in JavaScript:

```
// Incomplete branching: IF (...) THEN {...}
if(/* Condition */) {
  // List of commands
}

// Continuation of the code on the next page
```

```

// Full branching:
// IF (...) THEN {...} ELSE {...}
if(/* Condition */) {
    // List of commands
} else{
    // List of commands
}

// Full branching:
// IF (...) THEN {...}, ELSE IF (...) THEN {...}, ..., ELSE {...}
if(/* Condition */) {
    // List of commands
} else if{
    // List of commands
} else{
    // List of commands
}

```

Now let's talk about basic operators. Basic operators are used to perform mathematical operations, such as: multiplication; division; to add; etc. Here's how to use them in code:

```

// When using text
var varText1 = Hello;
var varText2 = "World";
// var is the same as let

var TextSum = varText1 + varText2;
/* The value of the variable will be: "HelloWorld" */

var TextMultiply = varText1 * 3;
/* The value of the variable will be: "HelloHelloHello"
Where 3 was indicated - we indicated the number, because
text cannot be multiplied by text. But
the text can be multiplied by a positive integer.*/

// As for numeric
var varNum1 = 4;
var varNum2 = 3;

var plus = varNum1 + varNum2;// Adding
/* The value of the variable will be: 7 */

// Continuation of the code on the next page

```

```

var minus = varNum1 - varNum2;// Subtraction
/* The value in the variable will be: 1 */

var multiply = varNum1 * varNum2;// Multiplication
/* The value of the variable will be: 12 */

var divide = varNum1 / varNum2;// Division
/* The value in the variable will be: 1.3333333333333333
*/

var remainder= varNum1 % varNum2;// Obtaining the remainder of
the division
/* The value in the variable will be: 1 */

var degree= varNum1 ** varNum2;// Raise to power
/* The value in the variable will be: 64 */

```

JS also has applications. Let's have a variable varApply that stores the number 5. Here's how the apply will work:

Application name	1st method	2nd method	Result
Apply addition (+=)	<code>varApply += 5;</code>	<code>varApply = varApply + 5;</code> 5+5	10
Apply subtraction (-=)	<code>varApply-= 5;</code>	<code>varApply = varApply + 5;</code> 5-5	0
Apply multiplication (*=)	<code>varApply*= 5;</code>	<code>varApply = varApply + 5;</code> 5*5 or 5x5	25
Apply division (/=)	<code>varApply/= 5;</code>	<code>varApply = varApply + 5;</code> 5/5 or 5:5	1
Apply obtaining a division fraction (%=)	<code>varApply%= 5;</code>	<code>varApply = varApply + 5;</code> 5%5	0
Apply exponentiation (**=)	<code>varApply**= 5;</code>	<code>varApply = varApply + 5;</code> 5^5	3125
Apply unit addition (varApply++)	<code>varApply++;</code>	<code>varApply = varApply +1;</code>	6
Apply unit subtraction (varApply--)	<code>varApply--;</code>	<code>varApply = varApply - 1;</code>	4

In JavaScript, as in all programming languages, there are those that are repeated through a condition (This is while) or several times (This is for). To begin with, consider the for loop. Here is the code with this loop:

```
for(Initialization;Condition;Increment/Decrement) {
  // List of codes
}
/*
Initialization: Here you initialize a new counter which
used in a for loop. Runs only once.
Condition: An expression that is checked before each iteration,
similar to a while loop.
Increment/Decrement: Operations performed on a counter
at the end of each loop iteration.

An example of writing that command:
for (let i = 1; i < 5; i++) {
  console.log("Loop iteration:", i);
};

let i = 1: Initialization where we create variable i inside the loop
for. This operation is performed once.
i < 5: Condition checked before each iteration.
i++: Increment expression, executed after each iteration.
console.log("Loop iteration:", i);: The body of the for loop.
*/
```

Now about the while loop. Here is the code:

```
while(/* Condition */) {
  // List of codes
}

// You can also write like this:
do{
  // List of codes
} while(/* Condition */);
```

Now let's move on to converting values. For example: we have a variable that has a text value type, then we need to get its numeric value type. Here is the code:

```
var textNum = "5";// The text type of the value
var numNum = 5;// Numeric value type
var numNum2 = parseInt(textNum);/* Conversion
text to integer */
var numNum3 = parseFloat(textNum);/* Text conversion
to a decimal number */
var textNum2 = numNum.();/* Numeric conversion
to text value */
```

Lesson 3.6. Functions.

An example of how you can create functions on JS:

```
function name(a,b,c, ...) {  
  // List of commands  
};  
// Where name is the name of the function  
// Where a, b, c, ... are variables that can be accepted  
by the function
```

Now let's make a function that accepts a value and displays the result in the console:

```
function name(a,b,c) {  
  console.log(a + b * c);  
};  
name(5,10,4)// In the console it will be: 45, because 5 +  
10 * 4 = 45
```

But this is to show the result in the console, because you can still make the function return the result:

```
var varFun1;  
function name(a,b,c) {  
  return a + b * c;// Returns the value of the function  
};  
varFun1 = name(6,30,2)  
// The variable varFun1 will have: 66, because 6 + 30 * 2 = 66  
console.log((45,2,5))// The console will have: 55
```

And this is HTML code that has an input field and a button to display a dialog message that has the text from the input field:

```
<!DOCTYPE html>  
<html><head>  
  <meta charset="UTF-8">  
  <title>JavaScript</title>  
</head>  
<body>  
  <input id="input">  
  <button onclick="a()">Alert</button>  
  <script>  
    function a() {  
      const textFromInput=document.getElementById("input")  
      let text=textFromInput.value;  
      // textFromInput.value takes the value from <input>  
      alert(text);};  
  </script>  
</body></html>
```

Lesson 3.7. Random

Basics of random number generation

JavaScript uses the built-in Math object with the random() method to generate random numbers. The Math.random() method returns a random number between 0 (inclusive) and 1 (not inclusive).

```
let randomNumber = Math.random();
console.log(randomNumber); /* Random number between 0
(inclusive) and 1 (not inclusive) */
```

Generation of random integers

To generate a random integer in a certain range, you need to use Math.random() along with math operations.

Example: A random number from min to max inclusive:

```
function getRandomInt(min,max) {
    return Math.floor(Math.random()*(max - min + 1))+ min;
}

let randomInt = getRandomInt(1,10);
console.log(randomInt); /* Random number between 1 and 10
inclusive */
```

- Math.random() generates a number between 0 and <1.
- Math.random() * (max - min + 1) expands the range to [0, max - min + 1).
- Math.floor() rounds a number to the nearest smaller integer.
- Adding min shifts the range to [min, max].

Floating-point random number generation

To get a random floating point number in a specific range, you can use Math.random() without rounding.

Example: A random number from min to max

```
function getRandomFloat(min,max) {
    return Math.random()*(max - min)+ min;
}

let randomFloat = getRandomFloat(1.5,5.5);
console.log(randomFloat); /* Random number between 1.5
(inclusive) and 5.5 (not inclusive) */
```

Generation of random elements from an array

To select a random element from the array, use `Math.random()` together with `Math.floor()`.

Example: A random element from an array:

```
function getRandomElement(array) {  
  let index = Math.floor(Math.random()* array.length);  
  return array[index];  
}  
  
let fruits =['apple','banana','cherry','date'];  
let randomFruit = getRandomElement(fruits);  
console.log(randomFruit); // Random fruit from the array
```

Quick tips

Do not use `Math.random()` for cryptographically secure random numbers. For such cases, use `crypto.getRandomValues()`.

```
let array = new Uint32Array(1);  
window.crypto.getRandomValues(array);  
let cryptographicallySecureRandomInt = array[0];
```

Avoid using `Math.random()` for important security functions such as password or token generation. It is better to use specialized libraries.

Result

Generating random numbers in JavaScript is quite simple thanks to the `Math.random()` method. For more complex cases, such as random selection from an array or generating floating-point numbers, use a combination of math functions. For cryptographic needs, use the crypto API.

Lesson 3.8. Web programming scripts and frameworks.

There are several popular JavaScript frameworks that are widely used to build web applications. Here are some of them:

React

Developer: Facebook

React is a JavaScript library for building user interfaces. It allows you to create reusable components and manage application state.

- **Features:**
 - Uses JSX to describe UI components.
 - One-way data binding.
 - Virtual DOM to improve performance.
 - Support for extensions like React Router for routing.

His example:

```
import React from 'react';
import ReactDOM from 'react-dom';

function App() {
  return(
    <div>
      <h1>Hello, React!</h1>
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

2. Angular

Developer: Google

Angular is a platform for building web applications that includes a framework, libraries, and tools.

- **Features:**
 - Uses TypeScript as the main language.
 - Two-way data binding.
 - Modular architecture.
 - Built-in services for HTTP requests, routing and other tasks.

His example:

```
// app.module.ts
import{NgModule}from '@angular/core';
import{BrowserModule}from '@angular/platform-browser';
import{AppComponent}from './app.component';

@NgModule({
  statements:[AppComponent],
  imports:[BrowserModule],
  providers:[],
  bootstrap:[AppComponent]
})
export class AppModule{ }
```

```
// app.component.ts
import{Component}from '@angular/core';

@Component({
  selector: 'app-root',
  template: `<h1>Hello, Angular!</h1>`,
  styles:[]
})
export class AppComponent{ }
```

3. Vue.js

Developer: Evan You

Vue.js is a progressive framework for building user interfaces that is easily integrated into existing projects.

- **Features:**
 - Easy integration with other libraries.
 - One-way and two-way data binding.
 - Virtual DOM.
 - Support for components and directives.

His example:

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Vue App</title>
</head>
<body>
  <div id=app></div>
  <script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
  <script src="app.js"></script>
</body>
</html>

// app.js
new Vue({
  el: '#app',
  data:{
    message: 'Hello, Vue!'
  },
  template: '<h1>{{ message }}</h1>'
});
```

4. Svelte

Developer: Rich Harris

Svelte is a tool for building fast web applications that does most of the work at compile time rather than in the browser.

- **Features:**
 - No virtual DOM.
 - Easier syntax.
 - High productivity.
 - Low packet size.

His example:

```
<!-- App.svelte -->
<script>
  let name='Svelte';
</script>

<main>
  <h1>Hello{name}!</h1>
</main>

<style>
  h1{
    color:purple;
  }
</style>

<!-- main.js -->
import App from './App.svelte';

const app = new App({
  target:document.body,
});

export default app;
```

5. Ember.js

Developer: Ember Core Team

Ember.js is a framework for building ambitious web applications with a clear structure and conventions.

- **Features:**
 - Powerful routing system.
 - Two-way data binding.
 - A large number of built-in tools and support libraries.
 - Strict code organization.

His example:

```
<!-- templates/application.hbs -->
<h1>Hello, Ember!</h1>

// app/app.js
import Application from '@ember/application';
import Resolver from 'ember-resolver';
import loadInitializers from 'ember-load-initializers';
import config from 'my-app/config/environment';

export default class App extends Application{
  modulePrefix = config.modulePrefix;
  podModulePrefix = config.podModulePrefix;
  Resolver = Resolver;
}

loadInitializers(App,config.modulePrefix);
```

Each of these frameworks has its own advantages and characteristics, and the choice depends on the specific needs of your project.

There is also code for setting the value of the slider. An example of the code with setting the value of the slider is on the next page.

```
<!-- Sets the value of the slider -->
<!DOCTYPE html>
<html><head>
  <meta charset="UTF-8">
  <title>JavaScript</title></head>
  <body>
    <input type="range" id="input">
    <script>
      const textFromInput=document.getElementById("input")
      textFromInput.value=70;// Sets the value
      // slider at 70%
    </script></body></html>
```

IV CHAPTER. DATABASE USE.

Lesson 4.1. What is a database?

A database (DB) is an organized set of data stored and managed by electronic means. They allow you to store, search, change and retrieve information. The main components of the database include:

1. **Data:** Information stored in the database.
2. **Database management system (DBMS):** Software used to manage a database, such as MySQL, PostgreSQL, Oracle, SQL Server.
3. **Tables:** Basic structures in relational databases where data is stored in rows and columns.
4. **Requests:** Tools for extracting data from a database, often using query languages such as SQL.
5. **Scheme:** A structure that defines the organization of data in a database, including tables, fields, and relationships between them.

Databases are used to store a variety of information, from personal data to commercial transactions and scientific research. They are the basis for most information systems and applications that need to process large amounts of data.

Lesson 4.2. Relational and non-relational databases.

A relational database is a database that has a tabular view, while a non-relational database is a database that has a text view.

Relational databases		Non-relational databases								
<table><tr><th>First Name</th><th>Last Name</th></tr><tr><td>Joe</td><td>Biden</td></tr><tr><td>Emmanuel</td><td>Macron</td></tr><tr><td>Justin</td><td>Trudeau</td></tr></table>		First Name	Last Name	Joe	Biden	Emmanuel	Macron	Justin	Trudeau	<pre>{ "First Name":{ "1":"Joe", "2":"Emmanuel", "3":"Justin" }, "Last Name":{ "1":"Biden", "2":"Macron", "3":Trudeau } }</pre>
First Name	Last Name									
Joe	Biden									
Emmanuel	Macron									
Justin	Trudeau									

Lesson 4.3. Using JSON with JavaScript.

Let us have a JSON table in a page that has a JS script. Here is the page code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1.0">
  <title>Load JSON Data</title>
</head>
<body>
  <div id=output></div>

  <script>
    // Embedded JSON data
    const data= {
      "users":[
{"name": "John Doe", "age": 30},
{"name": "Jane Doe", "age": 25}
      ]
    };

    // Function to display data on the page
    function displayData(data) {
      const outputDiv=document.getElementById('output');
      data.users.forEach(user =>{
        const userDiv=document.createElement('div');
        userDiv.textContent=Name:${user.name}, Age:${user.age}`;
        outputDiv.appendChild(userDiv);
      });
    }

    // Display the data after the page is loaded
    window.onload= ()=> displayData(data);
  </script></body></html>
```

Lesson 4.4. What is SQL?

SQL (Structured Query Language) is a programming language used to manage and manipulate relational databases. SQL allows you to do the following:

1. **Data requests:** Retrieve data from one or more database tables.

```
SELECT * FROM table_name;
```

2. **Insert data:** Adding new records to the table.

```
INSERT INTO table_name (column1, column2) VALUES (value1, value2);
```

3. **Updating data:** Changing existing entries in the table.

```
UPDATE table_name SET column1 = value1 WHERE condition;
```

4. **Deleting data:** Delete records from the table.

```
DELETE FROM table_name WHERE condition;
```

5. **Creation and modification of the database structure:** Creating new tables, changing the structure of existing tables, etc.

```
CREATE TABLE table_name (column1 datatype, column2 datatype);  
ALTER TABLE table_name ADD column_name datatype;
```

6. **Data access control:** Setting access rights for users.

```
GRANT SELECT, INSERT ON table_name TO user_name;
```

SQL is a standard for working with relational databases such as MySQL, PostgreSQL, SQLite, Microsoft SQL Server, and others.

Lesson 4.5. Data retrieval, sorting, conditions and operators.

We will work with the country database, which contains the country table. Let's look at this table:

id	name	continent	region	surfacearea	capital	population
1	Japan	Asia	Eastern Asia	377829	Tokyo	126714000
2	Latvia	Europe	NULL	64589	Riga	2424200
3	Mexico	North America	Central America	1958201	Mexico City	98881000
4	Netherlands	Europe	Western Europe	41526	Amsterdam	15864000
5	Portugal	Europe	Southern Europe	91982	Lisbon	9997600
6	Sweden	Europe	Nordic Countries	449964	Stockholm	NULL
7	Ukraine	Europe	Eastern Europe	603700	Kyiv	50456000
8	United States	North America	NULL	9363520	Washington	278357000
9	Thailand	Asia	NULL	513115	Bangkok	61399000
10	Paraguay	South America	North America	406752	Asuncion	5496000
11	Philippines	Asia	Southeast Asia	300,000	Manila	75967000
12	New Zealand	Oceania	Australia and New Zealand	270534	Wellington	18886000
13	Norway	Europe	Nordic Countries	323877	Oslo	4478500
14	Monaco	Europe	Western Europe	1.50	Monaco	NULL
15	Malta	Europe	Southern Europe	316	Valletta	380200

SELECT is a statement that returns a set of data (a selection) from a database. In order to use the SELECT statement, we must specify what exactly we want to select and where we want to select from.

```
SELECT continent FROM country;
```

In the previous query, we used a SELECT statement to retrieve the single continent column from the country table. We need to specify the name of the

column immediately after the SELECT keyword, and after the FROM keyword we specify the name of the table from which we want to get data.

Let's learn a little more about this database. The country database contains one table named country. Let's look at this table.

This table has 15 rows. In other words, we have 15 different entries for different countries.

What about columns? Here we have 7 columns like id, name, continent, region, SurfaceArea, capital and population.

1. id - record number in this table;
2. name - name of the country;
3. continent - the name of the continent on which the country is located;
4. region - the name of the region of the country;
5. SurfaceArea - the surface area of the country;
6. capital - the capital of the country;
7. population - population of the country.

We can also retrieve multiple columns using the SELECT statement. The only difference is that after the SELECT word, we will need to specify multiple column names, which must be separated by commas. Consider an example where we get three columns from the countries table:

```
SELECT id, name, capital FROM country;
```

id	name	capital
1	Japan	Tokyo
2	Latvia	Riga
3	Mexico	Mexico City
4	Netherlands	Amsterdam
...
15	Malta	

Note that the SELECT query returns all rows in the specified column. However, what if we don't need all the values from a column (since they might be duplicates), but only the unique values? For such cases, it is convenient to use the DISTINCT keyword. Moreover, this keyword is used immediately before the column names. Consider an example:

```
SELECT DISTINCT region FROM country;
```

The result is on the next page.

region

null
Australia and New Zealand
Southeast Asia
Central America
Western Europe
Southern Europe
North America
Eastern Europe
Eastern Asia
Nordic Countries

We know that the SELECT statement selects all rows from a table of all or specific columns. However, what if we only need to retrieve a certain number of rows, for example?

Note. Different control systems implement this with different syntax. Since we are using PostgreSQL, we must use the LIMIT statement.

In the example below, we extract the first 7 rows (in our case - the capitals of the countries) from the column:

```
SELECT capital FROM country LIMIT 7;
```

capital
Tokyo
Riga
Mexico City
Amsterdam
Lisbon
Stockholm
Kyiv

The command to get all the data from the table:

```
SELECT * FROM country;
```

Let's understand what "data grouping" means with a simple example of the employee table:

department	employee_id	first_name	hire_date	last_name	salary
Engineering	1	John	2015-03-01T00:00:00Z	Doe	80000.00
Engineering	2	Jane	2017-08-15T00:00:00Z	Smith	90000.00
Marketing	3	Alice	2016-11-10T00:00:00Z	Johnson	75000.00
Marketing	4	Bob	2018-06-25T00:00:00Z	Brown	72000.00
...
Sales	10	James	2017-05-18T00:00:00Z	Clark	

Now let's imagine that we have the task "determine the number of employees in each department." To do this, we will group the data by the department column and use aggregation using the COUNT(*) function.

```
SELECT department, COUNT(*) AS number_of_employees
FROM employees
GROUP BY department;
```

So, as you can see, the syntax for grouping data looks like this:

```
SELECT column1, AGG_FUNC(column2)
FROM table
GROUP BY column1;
```

Note: AGG_FUNC stands for aggregate functions like MAX, MIN, COUNT, etc.

This syntax exists to find specific values using aggregate functions on specific columns.

Consider another example: our task is to find the department with the highest average salary.

To get the following data, we need to group the data by the department column and then use the AVG() function to calculate the average salary:

```
SELECT department, AVG(salary) as average_salary
FROM employees
GROUP BY department;
```

Note: Note that we will not be using this table in the exercises; the employees table will be used solely to demonstrate syntax examples and their application.

In this course, we will work with the Montreal metro system database, which contains the metro_travel_time table.

This table will contain information about the station's line (line_name), its name (station_name), and the amount of time it takes the train to travel from one station to the next (time_to_next_station).

Here's what that table looks like and a preview of the data in it:

id	line_name	station_name	time_to_next_station
1	Green	Angrignon	10
2	Green	Monk	16
3	Green	Verdun	9
4	Green	Charlevoix	17
...
21	Yellow	Longueuil	10

As you can see, this is not a complex table. Let's think about where we can apply grouping here.

The most obvious option is the grouping of subway lines by color. This means that we can aggregate the data by grouping it by subway line color.

Now let's practice grouping by doing the task.

For statistical analysis, we were tasked with counting the number of stations on each line and ranking them by increasing number of stations on each metro line.

To do this, we need to find the number of stations on each of the subway lines, and then sort them from least number of stations to most.

In this way, the construction company will understand which subway lines should be given priority attention for adding stations.

It is important for us to understand the order in which the clauses are written, in particular, where the GROUP BY clause should be placed.

So, the order looks like this:

1. SELECT operator;
2. FROM table;
3. WHERE construction;

4. GROUP BY construction;
5. ORDER BY construction;
6. LIMIT construction.

From this order, it is clear that the GROUP BY statement must be written AFTER the WHERE statement (or after the FROM table if your query does not filter using SELECT) and also BEFORE the ORDER BY statement.

Let's look at an example of this order of instructions using our employee table. Suppose we need to get the number of employees in each department, whose salary is higher than 70000, and sort them in ascending order:

```
SELECT department, COUNT(employee_id) AS number_of_employees
FROM employees
WHERE salary > 70000
GROUP BY department
ORDER BY number_of_employees;
```

Note:It is worth noting that the LIMIT clause is always written at the end. So you will easily remember its location in the request.

The construction company decided to increase the number of stations on the Yellow metro line.

Our next task is to find the turnaround time for each line. It is important for the company to be able to close the Yellow Line for maintenance and expansion by adding new metro stations, so it is critical not to cause undue inconvenience to passengers.

Therefore, we need to determine the total turnaround time of the train by summing the time to each station (using the SUM() function).

Note:If we simply add up the time to each station, it will be the time the train travels from one end station to the other. However, it is also important to know the total train turnaround time along the entire metro line. To achieve this, we must multiply the sum by 2.

To understand how to perform this task, consider an example with the employees table.

Suppose we need to find the department with the highest average monthly salary.

For this, we can use the following request:

```
SELECT department, AVG(salary) / 12 AS average_monthly_salary
FROM employees
GROUP BY department
ORDER BY average_monthly_salary DESC;
```

So, we get the necessary data as a result.

Here's the case: the school recommended you and the school is interested in you because they also have some assignments for you. But first, let's take a look at the table they provided:

id	student_surname	class_letter	subject_name	grade
1	Smith	A	Mathematics	85
2	Johnson	B	Physics	90
3	Williams	C	Chemistry	78
4	Brown	A	Biology	92
...
100	Gonzales	A	Biology	

As you can see, the school has a total of 100 students, whose information is provided in this table. The class_letter column contains information that can have 3 options: A, B or C. Also, the name of the subject (subject_name) and the student's grade (grade) are indicated here. The table is simple and contains marks for exams in various subjects.

Let's see how many students are in each class using the following query:

```
SELECT class_letter, COUNT(DISTINCT student_surname) AS anumber_of_students
FROM student_grades
GROUP BY class_letter;
```

The school is satisfied with our work and agrees to continue cooperation.

Now they have a new task for us. The top 10 students with the highest average score will receive a trip to the science center as a reward. One of the prerequisites is obtaining a score above 90 in mathematics on the exam. To find such students, they turned to you.

Let's see what we need to do using our employee table as an example.

Let's say we need to know in which departments there are employees who were hired until 2019 and the average salary in these departments. To perform such a task, you can use the following query:

```
SELECT department, AVG(salary) AS average_salary
FROM employees
WHERE hire_date < '2019-01-01'
GROUP BY department;
```

As you can see, there are only 3 such workers, and we used the necessary tools to achieve this result. Your task will be very similar, I'm sure you can handle it!

Here's a preview of the student_grades table we're working with:

id	student_surname	class_letter	subject_name	grade
1	Smith	A	Mathematics	85
2	Johnson	B	Physics	90
3	Williams	C	Chemistry	78
4	Brown	A	Biology	92
...
100	Gonzales	A	Biology	

The school is very grateful for your work, and now we have a new task.

It turned out that some students were taking additional exams when they were only supposed to take one. The school suspects them of dishonesty because each student is only supposed to have one grade.

We were instructed to find out the names of these students and pass them on to the school administration so that they could take the necessary measures.

Let's think together how we can do it. You can start by saying that we can do this with a WHERE clause and it will look something like this:

```
SELECT student_surname
FROM student_grades
WHERE COUNT(grade) > 1;
```

But as you can see, we get an error indicating that we cannot use aggregate functions inside the WHERE clause. This is where we will need the HAVING clause.

Let's understand what it is and how to use it, using an example from our employee table.

Let's say we need to get departments where the average employee salary is below \$70,000 per year.

For this we will need to use the aggregate function and the HAVING clause.

Let's see how we can do this:

```
SELECT department
FROM employees
GROUP BY department
HAVING AVG(salary) < 70000;
```

We got one department back using the HAVING class, where we set a condition for the column we were grouping the data by.

Note: To use data aggregation in the HAVING class, we need to have data grouping in our query. As in the query above, we grouped the data by the department column.

Let's look at the more general syntax of the HAVING class and when it's best to use it:

```
SELECT column1, column2... --(optional)
FROM table
GROUP BY column1
HAVING AGG(column_n) --(condition);
```

Let's also briefly review the main difference between WHERE and HAVING clauses and when you should use each one:

A WHERE clause is used before aggregating data, while a HAVING clause is used after aggregating data;

The WHERE clause is written before the GROUP BY, while the HAVING clause is written after the GROUP BY.

The school used to hold competitions for students who passed Mathematics, and some of them won awards. Now the school wants to make sure there are no students who cheat and take more than one exam, including the math exam.

Therefore, the school asked us to determine the names of those students who passed more than one exam, one of which was Mathematics.

Here's a query from our previous exercise that you can use as an example:

```
SELECT student_surname, AVG(grade) as average_grade
FROM student_grades
GROUP BY student_surname
HAVING COUNT(grade) > 1;
```

Where we indicated the ">" sign is an inequality sign. Here's how you can specify inequality signs in SQL if:

Sign	What means
=	Is equal to
<>	Not equal to
>	More
<	Less
>=	Greater than or equal to
<=	Less than or equal to

We used to work for different companies and execute SELECT queries for their needs. However, we need to learn how to create and modify tables.

Let's get straight to the point!

Tables are created using the CREATE statement, which has a similar structure to the SELECT statement, but instead of retrieving data, it creates data.

Let's look at the syntax:

```
CREATE TABLE example (  
    id INT PRIMARY KEY,  
    some_info VARCHAR(50)  
);
```

Note: When you run these examples, you will not get any output because these examples only create a new table. If you try to run the code again, you will get an error message saying that the table already exists. These code snippets are examples, and later in the task, data will be inserted into these newly created tables and displayed on the screen so you can see everything working.

Now let's understand what is written above.

This query will create an EMPTY table with two columns: id and some_info.

Pay attention to the data types used. The words INT or VARCHAR indicate the data type for each column. For example, INT represents integer data, while VARCHAR(50) represents text with a maximum of 50 characters.

We won't delve into all the data types right now, as there are quite a few of them. We'll focus on the main data types in this chapter, and we'll cover each one as we progress through the tutorial!

For example, let's create another table with different types of data:

```
CREATE TABLE users (  
    id INT PRIMARY KEY,  
    name VARCHAR(50),  
    birthdate DATE,  
    salary DECIMAL(10, 2),  
    is_active BOOLEAN  
);
```

With this query, we create an empty table that should contain information about users, including:

1. ID with data type integer;
2. Name information, with data type VARCHAR(50);
3. Date of birth information, with data type DATE.
4. Salary information, with a data type of floating-point number;
5. Whether the user is active, with a data type that accepts only true or false.

Limitation:

You may have noticed that next to each ID value we put the words PRIMARY KEY. This is called a constraint and refers to the restriction placed on that column.

For example, PRIMARY KEY provides uniqueness and identifies each row in a table. The table can have only one such column.

There are also other restrictions, such as:

- NOT NULL: Guarantees that the column will not contain NULL values;
- UNIQUE: Ensures the uniqueness of all values in a column or combination of columns;
- DEFAULT: Sets the default value for a column if no value is specified for that column when inserting data.

These aren't all the restrictions in use, but these are the ones we need for now.

Consider an example where we modify the previous table:

```
CREATE TABLE users_2 (  
    id INT PRIMARY KEY,  
    name VARCHAR(50) NOT NULL,  
    birthdate DATE,  
    salary DECIMAL(10, 2) DEFAULT 50000, is_active BOOLEAN);
```

Now the name column cannot have empty or null values and the salary column defaults to 50000.

This way, you can use constraints to control the columns of a table when you create it.

In the previous chapter, we learned how to create tables.

But imagine a situation where we need to add a column to an existing table. It would be quite pointless to delete a table (especially if it already contains some data) and then create a new one, repopulating it with data.

Therefore, in this section we will look at the ALTER operation.

Let's see how to use this operation:

```
CREATE TABLE library (  
    id INT PRIMARY KEY,  
    title VARCHAR(50) NOT NULL,  
    author VARCHAR(50),  
    pages INT  
);  
ALTER TABLE library ADD price DECIMAL DEFAULT 300;  
ALTER TABLE library DROP COLUMN price;
```

As you can see, this is the script to create the table from the previous section.

Two ALTER operations follow. The first operation adds a price column to the table, setting the default value of 300 for this column. The second operation deletes this column.

The syntax is extremely simple:

```
ALTER TABLE table_name ADD/REMOVE column_name DATA_TYPE;
```

The syntax is actually quite simple.

Note! With the ALTER statement, you can perform various schema-level operations on a table, such as adding or removing constraints, renaming, changing data types, and adding or removing indexes.

Let's move on to another operation, namely the insertion operation. You can use the INSERT statement to insert data into SQL. To use INSERT, we have to specify which columns we want to add values to.

Here is the syntax of this statement:

```
INSERT INTO library (id, title, author, pages) VALUES
    (1, 'CAMINO GHOSTS', 'John Grisham', '213'),
    (2, 'FUNNY STORY', 'Emily Henry', '341');
```

You probably correctly noticed that this is an excerpt of the script from the previous section, where data is inserted into the library table.

Let's break down what's going on here:

1. First, the keywords **INSERT INTO** are written, followed by the `table_name` where the data will be inserted;
2. Then the parentheses are opened and the names of the columns where the data will be inserted are indicated; in our case there are 4 columns;
3. After that, the keyword **VALUES** is written, and the brackets are opened, where the data will be written;
4. The data should be written in the same order as the column names, while respecting the data types. For example, you cannot insert an integer value into a column with a **VARCHAR** data type;
5. **Brackets are closed**, and then a comma is placed, thus filling one line. You can fill in as many rows as you want using this method.

In summary, the general syntax of the **INSERT** statement looks like this:

```
INSERT INTO table_name (column1_name, column2_name) VALUES
    (column1_value, column2_value),
    (column1_value, column2_value),
    ...;
```

Don't forget the semicolon at the end!

It is worth mentioning two more operations in DDL: **DROP** and **TRUNCATE**.

Let's take a quick look at what each of these operations does:

DROP: Used to drop database objects such as tables, databases, and indexes.

TRUNCATE: Removes all rows from a table but preserves its structure.

I used these operations to clear or delete tables to test the tasks in the previous sections. Their syntax is quite simple; let's look at them:

```
ALTER TABLE table_name ADD/REMOVE column_name DATA_TYPE;
```

This code will delete the employees table from the database. In many DBMS, this operation requires certain permissions, and if you are working on a project, you may not have access to this operation. You'll learn about roles and how to configure them in the next course, which covers Advanced SQL Concepts.

```
ALTER TABLE table_name ADD/REMOVE column_name DATA_TYPE;
```

This code will delete all rows from the employees table, completely clearing it and making it empty. This operation will not affect the structure of the table, that is, it will not change the columns or constraints. You also need permissions in the DBMS for such an operation, as not everyone should be granted the ability to clear a table.

Use these operations carefully, because if you don't have database snapshots, you won't be able to roll back table deletions or row truncations.

Note! Often, developers use soft deletion by adding a new `is_deleted` column with data type `BOOLEAN`, and when some rows are deleted, the status is set to true (or 1). Thus, you can see the deleted data and not worry about its integrity.

You already know how to clear a table, add a column, insert data, and more. However, to properly interact with the database, we need to understand how to update and delete individual rows.

There are two types of statements and requests for this: `UPDATE` and `DELETE` requests.

UPDATE: Used to modify existing data in the table. With the help of such a query, we can change the data in the table without affecting other rows. Let's consider an example with the medications table, which is on the following page:

id	name	price
1	Aspirin	5.00
2	Ibuprofen	10.00
3	Paracetamol	8.00
4	Naproxen	12.00
...
10	Azithromycin	22.00

Let's imagine that we need to update the price of a certain type of medicine. For example, Ibuprofen is currently 50% off and we need to change the price of this product.

Our update request will look like this:

```
UPDATE medications
SET price = 4
WHERE id = 2;

SELECT *
FROM medications
ORDER BY id;
```

Here, we updated the medications table so that the price of the product with id 2 (Ibuprofen) was set to 4. We then selected all the columns from the table to verify that the price column was successfully updated. You can substitute any value and see how the update operation works in SQL.

In summary, the general syntax looks like this:

```
UPDATE table_name
SET column_name = value
WHERE some_condition;
```

The operation of the DELETE command is almost identical in principle. However, we don't use the SET keyword here, because we're not changing anything; we just delete the rows.

The syntax for deleting would look like this:

```
DELETE FROM table_name
WHERE some_condition;
```

But I will remind you that deleting lines should be done carefully, because you will not be able to recover them easily.

Note! If you do not include a WHERE condition, data will be updated or deleted for all rows.

