**Lecture 1. Introduction to web development. Basics of HTML, CSS, JavaScript, React.js, Django Framework**

**Table of contents**

[Introduction 2](#_Toc139893266)

[Main Stages of Web Development 2](#_Toc139893267)

[Code Editor 6](#_Toc139893268)

[Git 7](#_Toc139893269)

[HTML 12](#_Toc139893270)

[CSS 20](#_Toc139893271)

[Bootstrap Framework 29](#_Toc139893272)

[JavaScript programming language 32](#_Toc139893273)

[React.js Library. Creating reactive interfaces. 38](#_Toc139893274)

[Python basics 49](#_Toc139893275)

[Django Framework 57](#_Toc139893276)

[API 65](#_Toc139893277)

[REST API 67](#_Toc139893278)

# Introduction

Web development is one of the modern professions in the IT industry. Web development technologies in the hands of a specialist allow you to create websites, applications, work on other tasks in the field of business and marketing. This knowledge will be useful and needed by managers and specialists at all levels.

Web developers, web designers, when discussing projects, very often go into technical conversations and it becomes very difficult for people from other business areas to understand them, especially business owners and marketers.

At the same time, it is important for the project customer to understand some technical aspects of web design and development services. At least so that they can have informed discussions with their development teams or vendors. This can give them a conceptual idea of how the project will be built from a technical point of view.

Today we will consider the following points:

* + the main stages of web development;
  + tools for developing the client side of a web application - Frontend (html, CSS, javaScript);
  + Git version control system, basic commands;
  + Bootstrap framework;
  + library for creating reactive interfaces - React.js;
  + Python+Django stack and REST API.

In order to develop web applications or websites, you need to know several web technologies, without which web development is impossible. Any web developer should know the following technologies: HTML, CSS, JavaScript, one of the frameworks for developing web interfaces (React.js , Angular.js or Vue.js), basics of SQL language for working with databases, one or more server programming languages, as well as an IDE that uses this language. Of course, this is not a complete list. However, these technologies are mandatory for studying at least at a basic level.

I will give you links where you can learn in more detail how to work with these tools.

# Main Stages of Web Development

In today's digital era, web development has become an integral part of our lives, enabling businesses, organizations, and individuals to establish a strong online presence. Understanding the stages involved in web development is crucial for anyone interested in creating a web project or pursuing a career in this field. So let's consider the main stages of web development and explore their content.

**Planning Stage:** The first stage of web development is the planning stage. During this phase, it is essential to determine the purpose, goals, and target audience of the website. Key activities in this stage include:

* Defining the website's objectives and desired outcomes.
* Conducting market research and competitor analysis.
* Identifying the target audience and their needs.
* Developing a site map or structure to outline the website's pages and navigation.
* Creating a timeline and budget for the project.

**Technical Task Development Stage:** The Technical Task development stage typically take place after the planning stage and before the design and development stages. This stage involves creating a comprehensive Technical Task document that outlines the specific requirements, functionalities, and technical details of the website. The Technical Task serves as a blueprint for the development team and ensures that everyone involved in the project is aligned and understands the project's scope. Key aspects of the Technical Task development stage include:

* *Gathering requirements:* This involves collaborating with stakeholders, clients, and the development team to gather all the necessary requirements for the website. It includes understanding the desired features, functionalities, and any specific technical constraints or preferences.
* *Defining technical specifications:* Based on the requirements, the Technical Task stage focuses on specifying the technical details of the project. This may include selecting the appropriate programming languages, frameworks, and tools to be used, defining the database structure, outlining security measures, and determining integration requirements with other systems if applicable.
* *Creating wireframes or prototypes:* As part of the Technical Task development, low-fidelity wireframes or prototypes may be created to visualize the layout and user flow of the website. These visual representations can help validate the technical approach and gather feedback from stakeholders.
* *Documenting technical specifications:* The Technical Task document should include detailed information about the website's architecture, data structures, user roles and permissions, third-party integrations, APIs, and any other technical considerations. It should also outline any specific performance or scalability requirements.
* *Review and approval:* The Technical Task document should be reviewed by the project stakeholders, including clients, designers, developers, and quality assurance personnel, to ensure that it accurately reflects the project requirements. Any necessary revisions or clarifications should be made before proceeding to the design and development stages.

The Technical Task development stage is important for establishing a clear understanding of the project's technical aspects and facilitating effective communication and collaboration among the development team. It acts as a reference throughout the subsequent stages of design, development, and testing, ensuring that the final project aligns with the project's objectives and technical requirements.

**Design Stage:** Once the Technical task is created, we move on to the design stage. This phase focuses on the visual aspects and user experience of the website. The main elements of the design stage include:

* Creating wireframes or low-fidelity mockups to visualize the layout and structure of the website.
* Designing the user interface (UI) elements, such as buttons, forms, and menus.
* Selecting appropriate color schemes, typography, and images.
* Ensuring the website design is responsive and compatible with different devices and screen sizes.

**Development Stage:** The development stage is where the website starts taking shape. It involves turning the design concepts into a fully functional website. Key aspects of the development stage include:

* Writing HTML, CSS, and JavaScript code to build the website's structure, style, and interactivity.
* Integrating content management systems (CMS) or e-commerce platforms if required.
* Implementing database systems for dynamic content and data storage.
* Optimizing the website's performance, including loading speed and responsiveness.
* Conducting rigorous testing and debugging to ensure functionality and cross-browser compatibility.

**Content Creation Stage:** The content creation stage is often concurrent with the development stage. It focuses on creating compelling and relevant content for the website. This includes:

* Creating text content for each page.
* Creating or selecting appropriate visuals, such as images, videos, and infographics.
* Optimizing content for search engines (SEO) by incorporating relevant keywords and metadata.
* Ensuring content is accessible and complies with online accessibility guidelines (WCAG).

**Deployment Stage:** Once the website has been developed and its content is ready, it is time for deployment. The deployment stage involves:

* Setting up a hosting environment and domain name registration.
* Uploading all the website files to the server.
* Configuring server settings and ensuring proper security measures are in place.
* Conducting final checks to ensure all website functionalities are working correctly.
* Launching the website for public access.

**Maintenance and Updates Stage:** After the website is live, the work doesn't end there. Regular maintenance and updates are crucial for its success. Key activities in this stage include:

* Monitoring website performance, uptime, and security.
* Implementing updates to the CMS, plugins, and frameworks used.
* Analyzing website analytics to gain insights into user behavior and make improvements.
* Adding new content, features, or sections as required.
* Conducting periodic backups and security audits to protect the website's data.

In conclusion, web development encompasses several stages, each with its unique content and objectives. By understanding these stages and their content, you'll be better equipped to plan, design, develop, and maintain a successful website. Remember that web development is a continuous process, and staying updated with the latest technologies and trends is essential for creating engaging and user-friendly websites.

# Code Editor

Next, let’s' look at code editors.

To write markup code, you can use various code editors: NotePad++, SubLime Text, MS Visual Studio Code. We will use the last one presented.

VS Code – is a free editor from Microsoft. It is quite "light". It is possible to install various plugins to speed up work.

To install it, go to the following address: https://code.visualstudio.com/. Download the distribution kit and install it on your computer.

After installing the code editor, you need to configure it to work. Let's move on to installing some plugins.

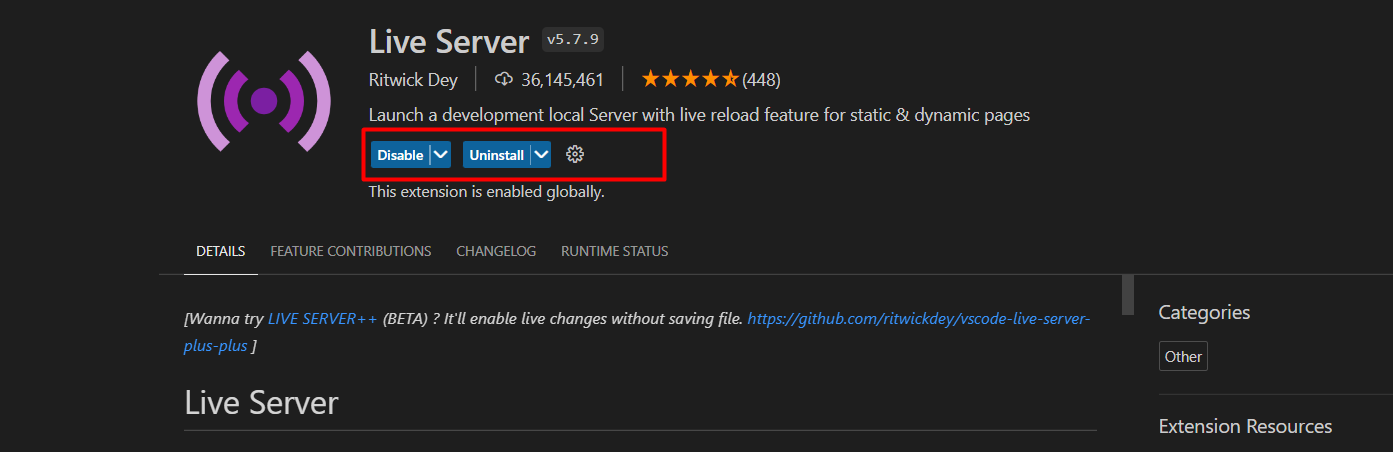
The following plugins are widely used:

1. Auto Close Tag.
2. Auto Complete Tag.
3. Auto Rename Tag.
4. ES7+ React/Redux/React-Native/JS snippets.
5. C# for Visual Studio.
6. HTML CSS Support.
7. Intellisense for CSS class names in HTML.
8. Live Server.

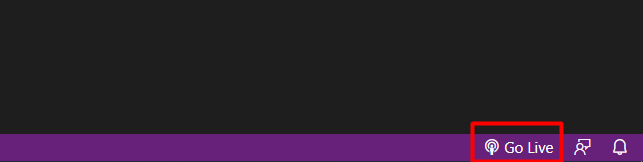
For synchronous updating of a web page in the browser, the Live Server plugin is used.

To launch the plugin, do the following:

1. Go to the «Extensions» tab. In the search bar, type the name of the required plugin. In addition, click the "Install" button.



1. Click on the "Go Live" icon on the bottom panel of the editor (on the right side).



1. "VS Code" will launch the embedded web server and open the html code in the browser at http://127.0.0.1:5500/index.html.

Now, if you make changes to the code and save it, the page will automatically overload and all changes will be reflected in the browser.

# Git

An equally important tool for a programmer to work in a team is a version control system.

**Git** – a widely used version control system that allows you to manage your projects efficiently. Whether you're a developer, a designer, or a content creator, Git can help you track changes, collaborate with others, and maintain a history of your work. In this lecture, we'll discuss what Git is, its advantages, and explore some of the basic commands to get you started.

**Understanding Git: Version Control Systems**

Version control systems are tools used to track and manage changes to files and projects over time. They enable collaboration, provide backup and restore capabilities, and facilitate easy experimentation.

**What is Git?**

Git is a distributed version control system, designed to handle everything from small to large projects with speed and efficiency. It was created by Linus Torvalds in 2005 for managing the Linux kernel development.

**Advantages of Git:**

1. Distributed nature: Each user has a complete copy of the project, enabling offline work and easy branching.
2. Speed and performance: Git is designed to be fast, even with large repositories.
3. Data integrity: Git uses a cryptographic hash function to ensure the integrity of the files.
4. Branching and merging: Git provides powerful branching and merging capabilities, allowing for parallel development and easy collaboration.
5. Traceability: Git maintains a complete history of all changes made to the project, allowing you to track who made the changes and why.

**Git** is a console utility for tracking and maintaining the history of file changes in projects. It is a software for tracking changes in any set of files. Usually, Git is used to coordinate the work of programmers who jointly develop source code during software development.

First of all: you should create a GitHub account on the official website and install the Git on your local PC. To download Git on your local PC you should use this link: <https://github.com/git-guides/install-git> .

**Installing Git**

To install Git on your computer, follow these steps:

1. Windows:

a. Download the Git installer from the official Git website: <https://git-scm.com/downloads>.

b. Run the downloaded installer and follow the instructions in the setup wizard.

c. Choose the desired installation options and complete the installation process.

1. macOS:

a. Git is usually pre-installed on macOS. Open the Terminal and type "git --version" to check if Git is already installed.

b. If Git is not installed, you can download the installer for macOS from: <https://git-scm.com/downloads>.

c. Run the downloaded installer and follow the instructions in the setup wizard.

1. Linux:

a. Git may already be available in your Linux distribution's package manager. You can use the package manager to install Git.

b. For example, on Ubuntu, you can use the command: "sudo apt-get install git"

c. If Git is not available in your package manager or you want the latest version, you can download it from: <https://git-scm.com/downloads>.

d. Follow the instructions provided on the website to install Git on your Linux distribution.

Working with any program always begins with its settings. Git can be set up once and changed only as needed.

Setting the name of the user from whom the changes will be sent.

*git config --global user.name "User Name"*

Setting an email address. Please note that the address must match the one to which the GitHub account is registered.

*git config --global user.email " mail@gmail.com "*

Installing a text editor that will open files to resolve conflicts:

*git config --global core.editor "editor"*

With the command

*git config --list*

You can see a list of all installed settings.

**Cloning a repository**

Let us say your Team lead gave you a new project. This project is located on a remote repository (in our case it will be GitHub). We need to copy this repository, make some changes in it and upload it back to the cloud. Let us see how it can all be done.

To clone the repository, enter the command:

*git clone*

and give his address. The repository is cloned to the currently selected folder in the console.

**Working with changes**

Any work with changes begins with getting the latest version of the remote repository. You can get the latest version with the command:

*git pull [<options>] [<repository> [<refspec>…]]*

Be careful: calling this command will erase all uncommitted changes.

After making any changes to the project, you can view the status of the files using the command:

*git status*

It will show files that have been modified, removed and new ones that need to be added.

To add tracking for new files, you must use the command:

*git add <filename> <filename>*

to add multiple files by name.

If you have many files to add, you can use the command:

*git add .*

which adds tracking for all new files from the current directory. And the command:

*git add -A*

adds also remote files, not only from the current directory, but from the entire local repository.

In addition to adding files, they also need to be deleted. There is a command for this:

*git rm <filename> <filename>*

which deletes files by their name.

After all new files have been added and old files have been removed, you can commit the changes. Committing changes, or commit, is very important, because until this command is executed, your local changes will not be written anywhere. To add a commit, you must enter the command:

*git commit -m "comment to commit"*

It is worth noting that it is necessary to properly break changes into commits and give full comments to commits.

If you have made changes and want to quickly undo them, then use the command:

*git reset*

which undoes all uncommitted changes.

By default, this command only removes from the index. And the command:

*git reset --hard*

permanently removes uncommitted current changes from the local repository and from the index.

Since all the above actions are performed in a local copy of the repository, this copy must be sent to the server so that other participants in the process can get the latest version. There is a command for this

*git push*

which sends all committed changes to the remote repository.

**Working with branches**

With Git, you have to constantly create and navigate branches.

Command:

*git checkout -b "branch-name"*

will create a branch with the specified name and automatically switch to it.

Once created, the branch can be pushed to the server using the command:

*git push origin "branch-name"*

Similarly, you can take a branch from a remote repository to your computer with the command

*git checkout origin/"branch-name" -b "branch-name"*

In order not to store branch names in memory or look up branch names, there are two special commands that allow you to view all existing branches of a local repository:

*git branch*

or all existing branches of the remote repository:

*git branch -r*

You can switch to any local branch with the command

*git checkout "branch-name"*

After work on the project, various unnecessary, untracked files and other garbage may remain in the repository. To remove all unnecessary files, use the command

*git clean -f -d*

Here are some additional resources to learn Git and the process of installing it on your computer:

1. Official Git Documentation: The official Git documentation is an excellent resource for learning Git from scratch. It covers everything from basic to advanced topics and provides detailed explanations of Git concepts and commands. You can find it at: <https://git-scm.com/doc>.
2. Pro Git Book: "Pro Git" is a comprehensive book by Scott Chacon and Ben Straub that covers Git in detail. It starts with the basics and progresses to advanced topics like branching, merging, and rebasing. The book is available for free online at: <https://git-scm.com/book/en/v2>.
3. Git Tutorials on Atlassian Bitbucket: Bitbucket provides a series of step-by-step tutorials that cover Git basics, branching, merging, and collaborating with others. These tutorials are beginner-friendly and include practical examples. You can access them at: <https://www.atlassian.com/git/tutorials>.
4. GitHub Learning Lab: GitHub Learning Lab offers interactive, hands-on courses to learn Git and GitHub. It provides a guided learning experience with real-world scenarios and exercises. You can access the courses at: <https://lab.github.com/>.
5. GitKraken Git Client: GitKraken is a popular Git client that offers a visually appealing and user-friendly interface. Their website provides tutorials and resources to help you learn Git and get started with their client: <https://www.gitkraken.com/learn-git>.

Remember to consult the official documentation or specific resources for detailed instructions on installing Git based on your operating system. These resources and installation instructions help you get started with Git.

# HTML

HTML is the standard language used for creating and structuring web pages on the World Wide Web. It forms the backbone of every website you visit, allowing us to present information in a structured and interactive manner.

HTML provides a set of tags (or elements) that define the structure and content of a web page. These tags are enclosed within angle brackets (< >) and are used to describe different parts of the document. By using a combination of these tags, we can format text, add images, create links, embed videos, and much more. The browser then interprets these tags to display the web page as intended.

Let's start by looking at some basic HTML tags and their usage:

1. The **<html>** tag: This tag is used to define the beginning and end of an HTML document. It wraps around all other HTML tags.
2. The **<head>** tag: Within the <html> tag, we have the <head> tag, which contains metadata about the document, such as the title, character encoding, and linked stylesheets.
3. The **<body>** tag: Inside the <html> tag, we have the <body> tag, which represents the main content of the web page. Everything we want to display to the user goes within this tag.
4. The **<h1>** to <h6> tags: These tags are used for headings of different sizes, with <h1> being the largest and <h6> being the smallest.
5. The **<p>** tag: This tag is used to define paragraphs. Text enclosed within <p> tags will be displayed as a separate paragraph.

Now that we have seen some basic tags, let's explore some common use cases for HTML:

1. Building a personal website: HTML provides the foundation for creating personal websites where you can display your skills, portfolio, or a blog. By using HTML tags, you can structure the content and add various elements to make your website visually appealing.
2. Developing online forms: HTML allows you to create forms that collect user input, such as contact information or survey responses. Form elements like text fields, checkboxes, radio buttons, and submit buttons can be used to design interactive forms.
3. Designing e-commerce websites: HTML plays a crucial role in designing online stores. You can create product listings, shopping carts, and checkout processes by utilizing HTML tags in conjunction with other web technologies like CSS (Cascading Style Sheets) and JavaScript.
4. Embedding multimedia: HTML enables you to embed multimedia elements like images, videos, and audio files into web pages. This way, you can enrich the user experience by incorporating visual and interactive content seamlessly.

To learn more about HTML and its various tags and attributes, there are several excellent online resources available. Here are a few links to get you started:

MDN Web Docs: HTML - <https://developer.mozilla.org/en-US/docs/Web/HTML>.

W3Schools HTML Tutorial - <https://www.w3schools.com/html/>.

HTML Dog: The Best-Practice Guide to XHTML and CSS - <http://www.htmldog.com/>.

I encourage you all to explore these resources, experiment with HTML tags, and practice building your own web pages. The possibilities with HTML are vast, and mastering this language will open doors to endless opportunities in web development.

Let's delve deeper into some of the basic HTML tags and understand how to use them effectively:

1. <html>: This tag serves as the root element of an HTML document and wraps around all other HTML tags. It indicates that the content inside is an HTML document.
2. <head>: This tag contains metadata about the document, such as the title, character encoding, linked stylesheets, and scripts. It doesn't appear on the web page itself but provides important information to the browser and search engines.
3. <body>: This tag represents the main content of the web page. It encapsulates all the visible elements that users will see when they visit the page. Text, images, links, headings, paragraphs, and other HTML elements are placed within the <body> tags.
4. <h1> to <h6>: These tags are used to create headings of different sizes, where <h1> represents the largest heading and <h6> represents the smallest. For example:

<h1>This is the largest heading</h1>

<h2>This is a subheading</h2>

<h3>Another subheading</h3>

1. <p>: This tag is used to define paragraphs of text. It creates a new line and adds spacing before and after the content. For example:

<p>This is a paragraph of text.</p>

1. <a>: This tag creates a hyperlink to another web page or a specific location within the same page. It is commonly used for navigation menus, linking to external websites, or creating internal links within a document. For example:

<a href="https://www.urfu.ru">Visit URFU website</a>

1. <img>: This tag is used to embed images into a web page. It requires the src attribute to specify the image source (URL or file path) and the alt attribute to provide alternative text for accessibility. For example:

<img src="image.jpg" alt="A beautiful image">

1. <ul> and <li>: These tags are used to create unordered lists. The <ul> tag represents the entire list, and each list item is defined using the <li> tag. For example:

<ul>

    <li>Item 1</li>

    <li>Item 2</li>

    <li>Item 3</li>

</ul>

1. <ol> and <li>: These tags are used to create ordered lists. Similar to unordered lists, the <ol> tag represents the list, and each item is defined using the <li> tag. The difference is that ordered lists are numbered. For example:

<ul>

    <li> First item </li>

    <li> Second item </li>

    <li> Third item </li>

</ul>

1. <div>: This tag is a generic container used to group and style content. It is often used for layout purposes or to apply CSS styles to a specific section of a web page. For example:

<div>

    <h2>Section Title</h2>

    <p>This is the content of the section.</p>

</div>

These are just a few examples of basic HTML tags and their usage. HTML provides a wide range of tags and attributes to structure and format content on web pages. By combining and nesting these tags creatively, you can create complex and visually appealing web pages.

**Semantic layout**

Semantic layout in HTML refers to the practice of using HTML tags that convey meaning and describe the structure of the content rather than just the appearance. By using semantic tags, you make your HTML more meaningful, accessible, and search engine friendly. Let's explore some examples of semantic layout elements:

1. <header>: The <header> tag represents the introductory content or the top section of a web page. It typically includes the website logo, navigation menu, or page title. For example:

<header>

    <h1>My Website</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">About</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

</header>

1. <nav>: The <nav> tag is used to define a section of navigation links. It represents a collection of links that allow users to navigate within the website or to different sections of a page. For example:

<nav>

    <ul>

      <li><a href="#">Home</a></li>

      <li><a href="#">About</a></li>

      <li><a href="#">Contact</a></li>

    </ul>

</nav>

1. <main>: The <main> tag represents the main content area of a web page. It typically contains the core content unique to that page. It should only be used once per page. For example:

<main>

    <h1>Welcome to My Website</h1>

    <p>Here is some important information.</p>

</main>

1. <article>: The <article> tag is used to define a self-contained, independent piece of content within a web page. It could represent a blog post, news article, forum post, or any other content that makes sense on its own. For example:

<article>

    <h2>Article Title</h2>

    <p>This is the content of the article.</p>

</article>

1. <section>: The <section> tag is used to divide the content of a web page into different sections or groups. It helps organize related content. For example:

<section>

    <h2>About Me</h2>

    <p>Some information about me goes here.</p>

 </section>

 <section>

    <h2>My Skills</h2>

    <ul>

      <li>HTML</li>

      <li>CSS</li>

      <li>JavaScript</li>

    </ul>

</section>

1. <aside>: The <aside> tag represents content that is tangentially related to the main content but can be considered separate from it. It is often used for sidebars, pull quotes, or advertisements. For example:

<aside>

    <h3>Related Articles</h3>

    <ul>

      <li><a href="#">Article 1</a></li>

      <li><a href="#">Article 2</a></li>

      <li><a href="#">Article 3</a></li>

    </ul>

</aside>

These are examples of semantic HTML tags that help structure and describe the content of a web page. By using these tags appropriately, you provide additional context to assist screen readers, search engines, and other devices in understanding the purpose and hierarchy of your content.

Semantic HTML not only improves accessibility and search engine optimization but also makes your code more maintainable and easier to understand for other developers.

Here is a sample code for a web page ("Pizza Order") with a semantic layout:

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Pizza Order</title>

  <style>

    /\* Some basic styling for demonstration purposes \*/

    body {

      font-family: Arial, sans-serif;

      margin: 0;

      padding: 20px;

    }

    header {

      background-color: #f2f2f2;

      padding: 10px;

      text-align: center;

    }

    nav {

      margin-bottom: 20px;

    }

    main {

      margin-bottom: 20px;

    }

    section {

      padding: 10px;

      background-color: #f9f9f9;

      margin-bottom: 10px;

    }

    footer {

      background-color: #f2f2f2;

      padding: 10px;

      text-align: center;

    }

  </style>

</head>

<body>

  <header>

    <h1>Pizza Order</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">Menu</a></li>

        <li><a href="#">Order</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

  </header>

  <main>

    <section>

      <h2>Choose Your Pizza</h2>

      <p>Select your favorite pizza from our menu:</p>

      <ul>

        <li>Margherita</li>

        <li>Pepperoni</li>

        <li>Hawaiian</li>

        <li>Veggie Supreme</li>

      </ul>

    </section>

    <section>

      <h2>Customize Your Pizza</h2>

      <p>Customize your pizza with the following options:</p>

      <ul>

        <li>Size: <select>

            <option>Small</option>

            <option>Medium</option>

            <option>Large</option>

          </select></li>

        <li>Toppings: <input type="checkbox"> Pepperoni <input type="checkbox"> Mushrooms <input type="checkbox"> Olives</li>

        <li>Crust: <input type="radio" name="crust" value="thin"> Thin <input type="radio" name="crust" value="thick"> Thick</li>

      </ul>

    </section>

    <section>

      <h2>Place Your Order</h2>

      <p>Enter your details and submit your order:</p>

      <form>

        <label for="name">Name:</label>

        <input type="text" id="name" name="name" required>

        <br>

        <label for="address">Address:</label>

        <textarea id="address" name="address" required></textarea>

        <br>

        <input type="submit" value="Submit Order">

      </form>

    </section>

  </main>

  <footer>

    &copy; 2023 Pizza Order. All rights reserved.

  </footer>

</body>

</html>

In this example, we have structured the web page using semantic HTML tags. The <header> contains the page title and navigation menu. The <nav> tag represents the navigation links. The main content is enclosed within the <main> tag, with each section of content wrapped in <section> tags. The footer is marked using the <footer> tag.

The provided CSS (page styles) is for demonstration purposes and can be customized further to achieve the desired visual style.

# CSS

CSS, short for Cascading Style Sheets, is a fundamental technology used for styling and formatting web pages. It allows us to control the visual appearance of HTML elements, such as fonts, colors, layout, and positioning. In this lecture, we will cover the basic CSS rules, element positioning, and the concept of web page adaptability.

1. **Basic CSS Rules**: CSS uses a set of rules to define how elements should be styled. Each rule consists of a selector and a declaration block. The selector targets one or more HTML elements, and the declaration block contains one or more property-value pairs.

Example: Let's say we have an HTML element with the class "box". We can style it using CSS as follows:

.box {

    background-color: blue;

    color: white;

    font-size: 18px;

}

In this example, the selector is .box, and the declaration block contains three property-value pairs. The background-color property sets the background color to blue, the color property sets the text color to white, and the font-size property sets the font size to 18 pixels.

1. **Element Positioning**: CSS provides several ways to position elements on a web page. The most commonly used positioning methods are "static," "relative," "absolute," and "fixed."

* *Static Positioning*: This is the default positioning for HTML elements, where they appear in the normal flow of the document.
* *Relative Positioning*: Elements positioned relatively are adjusted relative to their normal position. You can use properties like top, bottom, left, and right to offset the element from its normal position.
* *Absolute Positioning*: With absolute positioning, elements are positioned relative to their nearest positioned parent. If no parent has a defined position, it will be relative to the document itself.
* *Fixed Positioning*: Fixed positioning fixes an element's position relative to the browser window. It remains in the same position even when scrolling the page.

Example: Consider the following CSS code snippet:

.box {

    position: relative;

    top: 20px;

    left: 30px;

}

In this example, the .box element is positioned relatively, and it will be moved 20 pixels down and 30 pixels to the right from its normal position.

1. **Web Page Adaptability**: In today's multi-device world, it is crucial to create web pages that adapt to different screen sizes and devices. CSS provides several techniques to achieve web page adaptability.

* *Responsive Design*: Responsive design uses media queries to apply different styles based on the characteristics of the device, such as screen width, height, or orientation.

Example:

@media screen and (max-width: 600px) {

    body {

      font-size: 14px;

    }

}

In this example, when the screen width is 600 pixels or less, the font size of the body element will be reduced to 14 pixels.

* *Fluid Layouts*: Fluid layouts use relative units, such as percentages, instead of fixed units like pixels, to specify element dimensions. This allows elements to resize proportionally as the screen size changes.

Example:

.container {

    width: 80%;

}

.box {

    width: 50%;

}

In this example, the .container element will take up 80% of its parent's width, while the .box element will take up 50% of the .container element's width.

1. **Styling Text**: CSS allows you to apply various styles to text, such as changing the font, color, size, and alignment.

Example:

h1 {

    font-family: Arial, sans-serif;

    color: #FF0000;

    font-size: 24px;

    text-align: center;

}

In this example, the h1 heading element will have the Arial font family, red color (#FF0000), a font size of 24 pixels, and will be centered.

1. **Box Shadows:** CSS box shadows add depth and visual effects to elements.You can control the shadow's color, size, and position.

Example:

.box {

    box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3);

}

In this example, the .box element will have a box shadow with a 2-pixel horizontal offset, 2-pixel vertical offset, a blur radius of 5 pixels, and a slightly transparent black color (rgba(0, 0, 0, 0.3)).

1. **Hover Effects**: CSS allows you to add hover effects to elements, which change their appearance when the cursor hovers over them.

Example:

.button {

    background-color: #007BFF;

    color: #FFFFFF;

    padding: 10px 20px;

    transition: background-color 0.3s ease;

}

.button:hover {

    background-color: #0056b3;

}

In this example, the .button element will have a blue background color and white text. When hovered over, the background color will transition smoothly to a darker shade of blue.

1. **Flexbox Layout**: CSS Flexbox provides a flexible way to layout elements within a container, allowing for easy alignment and distribution.

Example:

.container {

    display: flex;

    justify-content: space-between;

}

.item {

    flex: 1;

    margin: 10px;

}

In this example, the .container element will arrange its child elements with equal space between them horizontally. The .item elements will have equal flex grow values, causing them to take up equal space within the container.

These examples demonstrate just a few of the many possibilities CSS offers for styling and layout on web pages. Experimenting with different properties and values will help you gain a deeper understanding of CSS and its capabilities.

**Adaptive layout**

Adaptive layout, also known as responsive layout, is the practice of designing web pages that adapt and respond to different screen sizes and devices. It ensures that the content and layout remain readable and usable across various devices, such as desktop computers, laptops, tablets, and smartphones.

To achieve adaptive layout, there are several key principles and techniques to consider:

1. **Fluid Grids**: One fundamental concept is the use of fluid grids. Instead of using fixed pixel-based measurements for element widths, fluid grids employ relative units such as percentages. This allows elements to resize proportionally based on the available screen width.

Example:

.container {

    width: 100%;

    display: grid;

    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

    gap: 20px;

}

In this example, the .container element uses a grid layout with columns that dynamically adjust based on the available width. The repeat(auto-fit, minmax(200px, 1fr)) rule ensures that each column will have a minimum width of 200 pixels but will expand to fill any additional space equally.

1. **Media Queries**: Media queries are a powerful CSS feature that allows you to apply different styles based on specific conditions, such as screen width, height, orientation, or device type. Media queries enable you to create breakpoints where the layout and styles can be adjusted accordingly.

Example:

@media screen and (max-width: 768px) {

    .container {

      grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));

    }

}

In this example, the media query targets screens with a maximum width of 768 pixels. Inside the query, the .container element's grid template columns are adjusted to have a minimum width of 150 pixels, providing a more suitable layout for smaller screens.

1. **Flexible Images and Media**: Adaptive layouts should also consider images and media elements. To prevent them from overflowing or becoming too small, CSS properties such as max-width: 100% and height: auto can be applied to ensure that images and media scale appropriately within their containers.

Example:

img {

    max-width: 100%;

    height: auto;

}

This CSS rule ensures that images will never exceed their container's width, maintaining their aspect ratio while scaling down if necessary.

1. **Mobile-First Approach**: One common strategy in adaptive layout design is adopting a mobile-first approach. This means initially designing and styling for smaller screens, and then progressively enhancing the layout as the screen size increases. By starting with a minimal, mobile-friendly layout, you ensure that the content remains accessible on all devices.

Adaptive layout is essential for creating user-friendly and accessible web pages across different devices. By utilizing fluid grids, media queries, flexible images, and adopting a mobile-first approach, you can design layouts that seamlessly adapt and provide optimal user experiences.

Here is an updated version of the code with gallery section added and additional CSS styles applied:

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Pizza Order</title>

  <style>

    /\* Basic styling \*/

    body {

      font-family: Arial, sans-serif;

      margin: 0;

      padding: 20px;

    }

    /\* Header styles \*/

    header {

      background-color: #f2f2f2;

      padding: 10px;

      text-align: center;

    }

    nav ul {

      list-style: none;

      padding: 0;

      margin: 0;

      display: flex;

      justify-content: center;

    }

    nav ul li {

      margin: 0 10px;

    }

    nav ul li a {

      text-decoration: none;

      color: #333;

    }

    /\* Main styles \*/

    main {

      margin-bottom: 20px;

    }

    section {

      padding: 10px;

      background-color: #f9f9f9;

      margin-bottom: 10px;

    }

    section h2 {

      color: #333;

    }

    section p {

      color: #666;

    }

    ul {

      list-style: disc;

      margin-left: 20px;

    }

    /\* Form styles \*/

    form label {

      display: block;

      margin-bottom: 5px;

      color: #333;

    }

    form input[type="text"],

    form textarea {

      width: 100%;

      padding: 5px;

      margin-bottom: 10px;

      border: 1px solid #ccc;

    }

    form input[type="submit"] {

      background-color: #333;

      color: #fff;

      padding: 10px 20px;

      border: none;

      cursor: pointer;

    }

    /\* Footer styles \*/

    footer {

      background-color: #f2f2f2;

      padding: 10px;

      text-align: center;

    }

    /\* Gallery styles \*/

    .gallery {

      display: flex;

      flex-wrap: wrap;

      justify-content: center;

    }

    .gallery-item {

      width: 200px;

      margin: 10px;

    }

    .gallery-item img {

      width: 100%;

      height: auto;

    }

    /\* Media queries \*/

    @media (max-width: 767px) {

      .gallery-item {

        width: 100%;

        max-width: 300px;

      }

    }

  </style>

</head>

<body>

  <header>

    <h1>Pizza Order</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">Menu</a></li>

        <li><a href="#">Order</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

  </header>

  <main>

    <section>

      <h2>Choose Your Pizza</h2>

      <p>Select your favorite pizza from our menu:</p>

      <div class="gallery">

        <div class="gallery-item">

          <img src="pizza1.jpg" alt="Margherita Pizza">

          <p>Margherita Pizza</p>

        </div>

        <div class="gallery-item">

          <img src="pizza2.jpg" alt="Pepperoni Pizza">

          <p>Pepperoni Pizza</p>

        </div>

        <div class="gallery-item">

          <img src="pizza3.jpg" alt="Hawaiian Pizza">

          <p>Hawaiian Pizza</p>

        </div>

        <div class="gallery-item">

          <img src="pizza4.jpg" alt="Veggie Supreme Pizza">

          <p>Veggie Supreme Pizza</p>

        </div>

      </div>

    </section>

    <section>

      <h2>Customize Your Pizza</h2>

      <p>Customize your pizza with the following options:</p>

      <ul>

        <li>Size:

          <select>

            <option>Small</option>

            <option>Medium</option>

            <option>Large</option>

          </select>

        </li>

        <li>Toppings:

          <label><input type="checkbox"> Pepperoni</label>

          <label><input type="checkbox"> Mushrooms</label>

          <label><input type="checkbox"> Olives</label>

        </li>

        <li>Crust:

          <label><input type="radio" name="crust" value="thin"> Thin</label>

          <label><input type="radio" name="crust" value="thick"> Thick</label>

        </li>

      </ul>

    </section>

    <section>

      <h2>Place Your Order</h2>

      <p>Enter your details and submit your order:</p>

      <form>

        <label for="name">Name:</label>

        <input type="text" id="name" name="name" required>

        <br>

        <label for="address">Address:</label>

        <textarea id="address" name="address" required></textarea>

        <br>

        <input type="submit" value="Submit Order">

      </form>

    </section>

  </main>

  <footer>

    &copy; 2023 Pizza Order. All rights reserved.

  </footer>

</body>

</html>

In this updated version, we have added a gallery section using the <div> element and applied CSS styles to create a responsive gallery layout. Each pizza item is represented by a <div> with the class "gallery-item" containing an <img> tag for the pizza image and a <p> tag for the pizza name. You can replace the image URLs (pizza1.jpg, pizza2.jpg, etc.) with your own image URLs or use appropriate image file paths.

Also we've made the following changes:

* Added Flexbox styles to the .gallery class to create a flexible gallery layout with items wrapping.
* Added the width of .gallery-item to 200px and added margin for spacing.
* Added a media query using @media to modify the .gallery-item width to 100% with a maximum width of 300px when the viewport width is 767px or less. This ensures that the gallery items stack vertically on smaller screens.

Feel free to customize the gallery styles further to achieve your desired visual presentation.

# Bootstrap Framework

Bootstrap, a popular front-end framework for building responsive websites. In this section, we will cover the basics of Bootstrap and explore some examples to help you get started.

**What is Bootstrap?**

* Bootstrap is an open-source CSS framework developed by Twitter.
* It provides a collection of pre-built responsive components, such as grids, typography, forms, buttons, and navigation.
* Bootstrap allows developers to create modern and visually appealing websites quickly.

**Key Features of Bootstrap:**

* Responsive Grid System: Bootstrap's grid system is based on a 12-column layout, allowing developers to create responsive designs for various screen sizes.
* CSS Components: Bootstrap includes a wide range of reusable CSS components, such as buttons, forms, navigation bars, and alerts.
* Typography: Bootstrap provides a set of typography styles and utilities for better text formatting.
* JavaScript Plugins: Bootstrap comes bundled with JavaScript plugins, such as carousels, modals, tooltips, and more, to enhance website functionality.
* Customization: Bootstrap can be customized to match specific design requirements by modifying variables or using custom stylesheets.

**Getting Started with Bootstrap:**

1. Downloading Bootstrap: Visit the official Bootstrap website (getbootstrap.com) and download the precompiled CSS and JavaScript files.
2. Linking Bootstrap Files: Include the Bootstrap CSS file in the head section and the JavaScript file just before the closing body tag in your HTML file.
3. Using Bootstrap CDN: Alternatively, you can use Bootstrap's Content Delivery Network (CDN) by adding the appropriate links to the CSS and JavaScript files in your HTML file.

**Bootstrap Grid System:**

* Bootstrap's grid system provides a flexible and responsive layout for creating web pages.
* The grid system is based on a 12-column layout, which allows you to divide the page into multiple columns.

Examples:

*<div class="container">*: Creates a container that houses the grid system.

*<div class="row">*: Defines a row to hold columns.

*<div class="col-md-6">*: Creates a column with a width of 6 out of 12 columns on medium-sized screens.

**Bootstrap CSS Components:**

Bootstrap offers a wide range of CSS components that can be easily integrated into your web pages.

Examples:

*Buttons*: <button class="btn btn-primary">Primary Button</button>

*Forms*: <input class="form-control" type="text" placeholder="Enter your name">

*Navigation*: <nav class="navbar navbar-expand-lg navbar-dark bg-dark">...</nav>

*Alerts*: <div class="alert alert-info" role="alert">...</div>

**Bootstrap Typography:**

Bootstrap provides a set of predefined typography styles and utilities.

Examples:

*Headings:* <h1>Heading 1</h1> to <h6>Heading 6</h6>

*Text Formatting*: <p class="lead">This is a lead paragraph.</p>

*Text Alignment*: <div class="text-center">Center-aligned text</div>

**Bootstrap JavaScript Plugins:**

Bootstrap includes a variety of JavaScript plugins that can enhance the functionality of your website.

Examples:

*Carousels:* <div id="myCarousel" class="carousel slide">...</div>

*Modals*: <div class="modal fade" tabindex="-1" role="dialog">...</div>

*Tooltips*: <button class="btn" data-toggle="tooltip" data-placement="top" title="Tooltip text">...</button>

**Customizing Bootstrap:**

Bootstrap can be customized to match your design requirements.

Customization options include:

* *Modifying variables*: Bootstrap provides a set of SCSS variables that can be overridden to change colors, spacing, and more.
* *Custom stylesheets*: You can create custom CSS stylesheets and override Bootstrap's default styles.

Bootstrap is a powerful framework that allows you to build responsive websites quickly. It provides a wide range of pre-built components and a responsive grid system. By using Bootstrap, you can save time and effort in designing and developing modern websites.

**Useful Resources:**

Official Bootstrap Website: <https://getbootstrap.com/>.

Bootstrap Documentation: <https://getbootstrap.com/docs/>.

Bootstrap Examples: <https://getbootstrap.com/docs/examples/>.

# JavaScript programming language

JavaScript is a popular programming language used for web development and creating interactive elements on websites. In this lecture, we will cover the fundamental concepts of JavaScript and provide examples to help you understand its usage.

1. Variables and Data Types: JavaScript allows you to store and manipulate data using variables. Here's an example:

// Declaration and assignment of variables

let name = "John";

let age = 25;

let isStudent = true;

In this example, we declare three variables: name, age, and isStudent. We assign the values "John" (a string), 25 (a number), and true (a boolean) to these variables, respectively. JavaScript has various data types, including strings, numbers, booleans, arrays, objects, and more.

1. Operators and Expressions: JavaScript provides a range of operators for performing arithmetic, comparison, and logical operations. Let's explore a few examples:

let a = 5;

let b = 10;

let sum = a + b; // Addition

let product = a \* b; // Multiplication

let isGreater = a > b; // Comparison

let logicalAnd = (a > 0) && (b > 0); // Logical AND

In this example, we perform addition, multiplication, comparison, and logical AND operations using the variables «a» and «b».

1. Control Flow: JavaScript allows you to control the flow of your program using conditional statements and loops. Here are two common examples:

Conditional Statements (if-else):

let age = 18;

if (age >= 18) {

console.log("You are eligible to vote.");

} else {

console.log("You are not eligible to vote yet.");

}

In this example, we check if the **age** is greater than or equal to 18. If it is, we print a message stating eligibility to vote; otherwise, we print a different message.

Loops (for loop):

for (let i = 0; i < 5; i++) {

console.log(i);

}

This loop will print the numbers from 0 to 4. The loop starts with i as 0, checks the condition (i < 5), executes the code inside the loop, and increments i by 1 until the condition is no longer true.

1. Functions: Functions are reusable blocks of code in JavaScript. They allow you to define a set of instructions that can be called multiple times. Here's an example:

function greet(name) {

console.log("Hello, " + name + "!");

}

greet("John");

In this example, we define a function called «greet» that takes a parameter «name» and prints a greeting message. We then call the function with the argument «John, » resulting in the output "Hello, John!".

JavaScript is a versatile programming language that forms an essential part of web development. We covered some of the basic concepts, including variables, data types, operators, control flow, and functions. Keep practicing and exploring JavaScript to enhance your programming skills further.

Here are some resources that can help you learn JavaScript:

1. Mozilla Developer Network (MDN) JavaScript Guide:
   * Website: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>
   * Description: The MDN JavaScript Guide is an excellent resource for beginners. It covers the basics of JavaScript, including data types, control flow, functions, and more. It also provides in-depth explanations and examples.
2. JavaScript.info:
   * Website: <https://javascript.info/>
   * Description: JavaScript.info is a comprehensive resource for learning JavaScript from scratch. It covers topics ranging from the basics to more advanced concepts, including object-oriented programming and asynchronous programming. The website offers interactive examples and exercises to reinforce your understanding.
3. Eloquent JavaScript by Marijn Haverbeke:
   * Book: "Eloquent JavaScript" by Marijn Haverbeke
   * Description: This book is highly regarded in the JavaScript community. It offers a beginner-friendly introduction to JavaScript, covering fundamental concepts and gradually progressing to more advanced topics. The book includes practical exercises to solidify your knowledge.
4. JavaScript: The Good Parts by Douglas Crockford:
   * Book: "JavaScript: The Good Parts" by Douglas Crockford
   * Description: This book focuses on the essential parts of JavaScript, providing guidance on best practices and avoiding common pitfalls. It's a concise and insightful resource for understanding the core concepts of JavaScript.
5. FreeCodeCamp:
   * Website: <https://www.freecodecamp.org/>
   * Description: FreeCodeCamp offers a free and interactive platform for learning web development, including JavaScript. It provides a structured curriculum with hands-on coding exercises and projects. The JavaScript section covers a wide range of topics and is suitable for beginners.
6. Codecademy:
   * Website: <https://www.codecademy.com/learn/introduction-to-javascript>
   * Description: Codecademy offers an interactive JavaScript course that teaches the basics of the language. The course provides a hands-on learning experience, allowing you to write code directly in the browser and receive instant feedback.

Remember, practicing coding and building projects is crucial to solidify your understanding of JavaScript. Don't hesitate to explore additional resources and engage in coding exercises to enhance your skills. Happy learning!

Here are a few examples of how JavaScript can be used:

1. Example: Changing HTML content dynamically JavaScript can be used to manipulate HTML elements and change their content dynamically. Here's an example that changes the text of a paragraph element when a button is clicked:

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Example</title>

</head>

<body>

<p id="demo">Click the button to change this text.</p>

<button onclick="changeText()">Click Me</button>

<script>

function changeText() {

document.getElementById("demo").innerHTML = "Text changed!";

}

</script>

</body>

</html>

When the button is clicked, the «changeText()» function is called, which selects the paragraph element with the ID "demo" and modifies its «innerHTML» property to display the new text.

1. Example: Handling form submission JavaScript can be used to validate form inputs and perform actions when a form is submitted. Here's an example that validates a simple form before submitting it:

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Example</title>

</head>

<body>

<form onsubmit="return validateForm()">

<input type="text" id="name" placeholder="Enter your name">

<input type="email" id="email" placeholder="Enter your email">

<input type="submit" value="Submit">

</form>

<script>

function validateForm() {

var name = document.getElementById("name").value;

var email = document.getElementById("email").value;

if (name === "" || email === "") {

alert("Please fill in all fields.");

return false;

}

// Additional validation or form processing code can be added here

// The form will only be submitted if all validations pass

alert("Form submitted successfully!");

return true;

}

</script>

</body>

</html>

The «validateForm()» function is called when the form is submitted. It retrieves the values of the name and email fields, checks if they are empty, and displays an alert message if any field is missing. If all validations pass, an alert is shown indicating successful form submission.

1. Example: Creating a simple slideshow JavaScript can be used to create interactive elements, such as slideshows. Here's an example that implements a basic image slideshow:

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Example</title>

<style>

.slideshow {

max-width: 500px;

height: 300px;

position: relative;

margin: 0 auto;

}

.slideshow img {

width: 100%;

height: 100%;

object-fit: cover;

}

</style>

</head>

<body>

<div class="slideshow">

<img src="image1.jpg" id="slide" alt="Slideshow Image">

</div>

<script>

var images = ["image1.jpg", "image2.jpg", "image3.jpg"];

var currentIndex = 0;

function changeSlide() {

var slide = document.getElementById("slide");

slide.src = images[currentIndex];

currentIndex++;

if (currentIndex >= images.length) {

currentIndex = 0;

}

setTimeout(changeSlide, 2000); // Change slide every 2 seconds

}

changeSlide(); // Start the slideshow

</script>

</body>

</html>

In this example, an array of image URLs is defined, and the «changeSlide()» function is used to update the «src» attribute of an image element with the next image in the array. The «setTimeout()» function is used to repeatedly call «changeSlide()» every 2 seconds, creating an automatic slideshow effect.

These are just a few examples of how JavaScript can be used to enhance websites and add interactivity. JavaScript's versatility allows it to be used for a wide range of purposes, including form validation, animation, DOM manipulation, and more.

# React.js Library. Creating reactive interfaces.

The development of any web project involves writing the client side of the application. Today, there are quite a lot of frameworks and libraries for solving this problem, for example: JQuery, React, Angular.js, Vue.js. Some of these tools are frameworks, some are libraries. Let us see what is the difference between them.

**Framework**

Any developer eventually comes across two new terms for himself: a library and a framework. The terms are different, but often their meaning is confused. Especially when it comes to the context of the frontend. Let us try to understand the similarities and differences.

The word "framework" literally translates as "basis". To put it even more simply: a framework is the foundation upon which any application can be built. It does not matter to the framework "what" you will develop with it, it is important "how". To do this, the framework has the special rules - restrictions.

The framework architecture may include a project file structure, a built-in set of tools for solving various tasks, a set of internal abstractions, CLI tools for automating tasks, and so on.

There are many frameworks for front-end development. Of the most popular, Angular, Vue and Ember stand out. All of them provide a set of built-in tools, abstractions, and so on. They can be used right out of the box, without wasting time searching for the right package and setting up relationships.

**Library**

If a framework is a ready-made instrument for developing an application, then a library is just a separate tool for solving some task or several tasks. The library does not impose any restrictions on the architecture or other tools used in the project. She is on her own.

You do not have to look far for examples. The once popular jQuery library is nothing more than a collection of functions that make interacting with the DOM easier. Another example - [Lodash](https://lodash.com/) — a library with an extensive set of utility functions for all cases.

The libraries listed above, as well as any others, do not impose any requirements on the architecture or type of project. It does not matter how your project is organized, nothing prevents you from including the library and using its functionality.

The library can be considered as an additional module with useful functions. Again, the library is a tool. It is quite logical that several different tools - libraries - can be used in one project. This is a normal situation, but there can be only one framework in a project.

**What is React.js?**

React.js, also known as React or ReactJS, is an open-source JavaScript library created by Facebook. It provides a declarative and efficient way to build user interfaces by breaking them down into reusable components. React follows a component-based architecture, allowing developers to build complex UIs by composing smaller, self-contained pieces.

**Key Concepts in React.js**

To understand React.js, it's important to understand some of its core concepts. Let's explore them one by one:

1. **Components:** At the heart of React.js are components. A component is a reusable, self-contained unit that renders a part of the user interface. Components can be thought of as building blocks that can be combined to create larger, more complex UIs. There are two types of components: functional components (also called stateless components) and class components.
2. **JSX (JavaScript XML):** JSX is a syntax extension used by React that allows you to write HTML-like code within JavaScript. It enables you to describe how your components should look. JSX makes React code more readable and intuitive, helping bridge the gap between HTML and JavaScript.
3. **Virtual DOM (Document Object Model):** React uses a virtual DOM to efficiently update and render components. The virtual DOM is a lightweight copy of the actual DOM, which represents the structure of your web page. When a component's state changes, React compares the virtual DOM with the actual DOM and updates only the necessary parts, resulting in optimal performance.
4. **State:** State represents the data that a component can hold and manage. It allows components to have dynamic behavior by updating and reflecting changes in the UI. State is typically used to store information that can change over time, such as user input, API responses, or the result of calculations.
5. **Props (Properties):** Props are inputs that components receive from their parent components. They are read-only and allow data to flow from a parent component to its children. Props are essential for passing information and behavior between components, making it easy to build reusable and modular code.

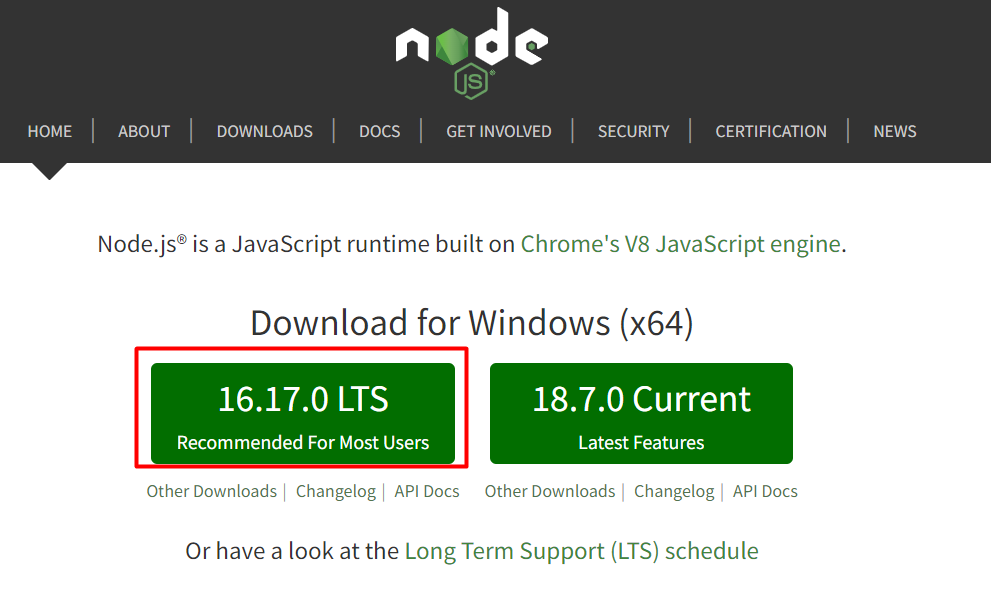
React is a JavaScript library used for building user interfaces. Unlike a framework, React provides a component-based approach to UI development. It can be employed for visualizations in conjunction with other libraries. For instance, React Native is utilized for developing mobile applications, while React 360 is used for creating virtual reality applications.

The primary objective of React is to minimize errors during UI development. This is accomplished through the utilization of self-contained and reusable components, which describe different parts of the user interface. These components can be combined to form a complete UI. By abstracting away much of the rendering process, React allows developers to focus more on the design and functionality of their applications.

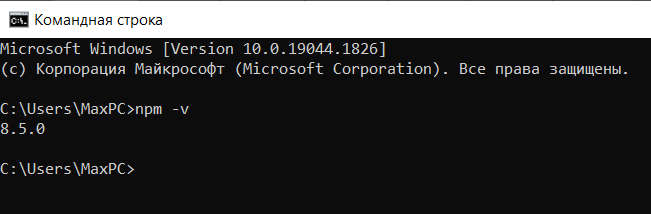
**Setting up a React.js Environment**

To start building with React.js, we need to set up our development environment. Follow these steps:

1. Install Node.js: React.js relies on Node.js for running JavaScript on the server. Visit the official Node.js website ([https://nodejs.org](https://nodejs.org/)) and download the latest LTS (Long-Term Support) version suitable for your operating system. Once installed, you'll have access to the Node Package Manager (npm).



To verify that the installation is correct, we can check the installed version of npm with the "npm -v" command:



1. Create a folder on your desktop.
2. Open a command prompt from the current folder.
3. Create a New React Project: run the following command to create a new React project:

npx create-react-app my-app

This command sets up a new React project with all the necessary dependencies and a basic project structure. Note that during project installation, all required dependencies are downloaded via «npm». This may take some time and nothing will happen.

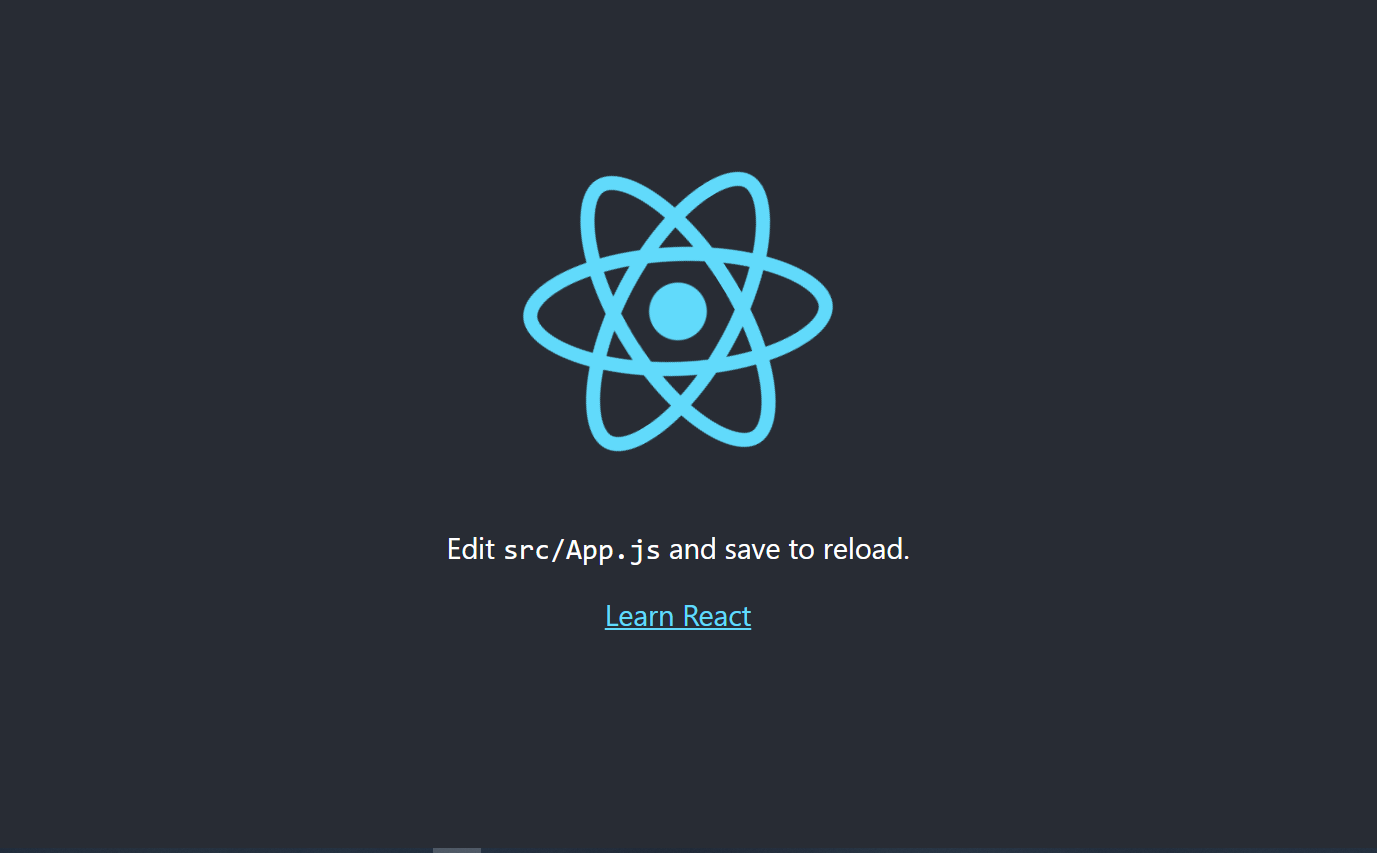
1. Start the Development Server: Navigate into the project folder by running:

cd my-app

Then, start the development server with the following command:

npm start

The basic page of a React app looks like this:

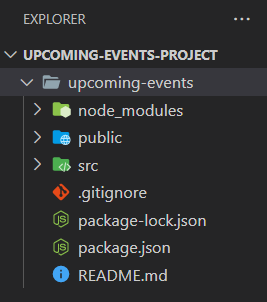


The project has an instant "hot update" feature enabled. This means that any changes made to the code will automatically trigger a page reload, allowing you to see the updates in real-time after saving.

Next, open the project in Visual Studio Code. (File -> Open Folder -> Select the project Folder).

**Project Structure**

CRA prepares a basic structure for each project (see figure below). Going to the project directory (upcoming-events) you will see several directories:



* + node\_modules. Directory with downloaded dependencies;
  + public. Directory for static file. A dev server is configured on this folder. It should contain all the static resources of the project (fonts, general styles, html, images, and so on);
  + src. Directory for the source code of the project. CRA automatically prepares several files with components for demonstration. Then they can be removed. At the root of the project are the files standard for any project: package.json, package-lock.json and readme.md.

**Scenarios out of the box**

In a CRA-created project, several scripts are available in package.json file. This set may differ depending on the template that was used when creating the project:

* + start. Launch of the project. By default, the development server rises on the port3000;
  + build. Preparing a project build. It is used when preparing a project for delivery to the product server;
  + test. Launch of automated testing;
  + eject. Extract configuration. For more fine tuning.

**Additional settings**

The CRA tool sets everything up on its own. However, the default settings are not always suitable. There may be, for example, such a problem: port 3000 (it is used by default) may be occupied by another service. Alternatively, you, as a developer, want the project to start in a specific browser, and not in the one that loads by default.

Some of the default settings can be overridden using [environment variables](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D1%8B). The documentation for CRA has a separate section "[advanced configuration](https://create-react-app.dev/docs/advanced-configuration)". It lists all the environment variables that allow you to do this.

You can set the desired value for a certain variable during the project launch right in the terminal. The command will look like this:

PORT=6000 npm run start

Here we are setting a new value for the PORT variable. When the application starts, it will be loaded on this port 6000 instead of the default 3000. Other available environment variables can be set like this:

PORT=6000 HOST=127.0.0.10 npm run start

This method has one serious drawback: environment variables must be set every time the application is launched. This is very time consuming and inefficient when you use different scenarios. This problem can be solved.

All environment variables can be set in a special configuration file. In this case, it is necessary to create a file in the root of the project with the name «.env.local» and set the necessary environment variables in it. When starting the project, the file will be parsed and the environment variables will automatically be set.

File «.env.local» cannot be committed to version control. In different cases, different environments can be used. This file is created for a specific task and is used only during development. Environment variables may contain some important information or confidential data, so the settings of the specified file should not end up in Git. Add an appropriate rule to the file gitignore.

For a complete list of supported environment variables, see [documentation](https://create-react-app.dev/docs/advanced-configuration). The following are most commonly used:

* + HOST. The address of the host on which the project is launched;
  + PORT. Port for launching the development server;
  + HTTPS. Use HTTPS by default;
  + BROWSER\_ARGS. Additional arguments for the browser in which the application opens after the start;
  + DISABLE\_ESLINT\_PLUGIN. Activating/deactivating the eslint-webpack-plugin plugin.

**Building a Simple React Component**

Let's now create a simple React component to understand how the different concepts we discussed come together. Open your code editor and locate the «src» folder in your React project. Inside it, create a new file called «Greeting.js». Add the following code to define our component:

import React from 'react';

const Greeting = () => {

return <h1>Hello, React!</h1>;

}

export default Greeting;

In the above code, we imported React and defined a functional component called «Greeting». The component returns «JSX» code, which renders an <h1> element with the text «Hello, React!».

To use this component, open the src/App.js file and replace its content with the following:

import React from 'react';

import Greeting from './Greeting';

const App = () => {

return <Greeting />;

}

export default App;

In the code above, we imported our «Greeting» component and used it within the «App» component. When we run our React application, we'll see the "Hello, React!" message rendered in the browser.

Let's create an example of a registration form using React.js. We'll create a form component that collects the user's name, email, and password. Here's how you can do it:

1. Create a new file called «RegistrationForm.js» inside the «src» folder of your React project.
2. Add the following code to define our registration form component:

import React, { useState } from 'react';

const RegistrationForm = () => {

const [name, setName] = useState('');

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const handleSubmit = (e) => {

e.preventDefault();

// Here, you can perform validation or submit the form data to an API

console.log('Form submitted:', { name, email, password });

// Reset the form

setName('');

setEmail('');

setPassword('');

}

return (

<form onSubmit={handleSubmit}>

<h2>Registration Form</h2>

<label>

Name:

<input

type="text"

value={name}

onChange={(e) => setName(e.target.value)}

required

/>

</label>

<label>

Email:

<input

type="email"

value={email}

onChange={(e) => setEmail(e.target.value)}

required

/>

</label>

<label>

Password:

<input

type="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

required

/>

</label>

<button type="submit">Register</button>

</form>

);

}

export default RegistrationForm;

In the code above, we imported React and useState from the 'react' package. We used the useState hook to manage the form's state. We declared three state variables: «name», «email», and «password», along with their respective setter functions (setName, setEmail, setPassword).

The «handleSubmit» function is triggered when the form is submitted. It prevents the default form submission behavior, logs the form data to the console (in this case), and resets the form fields by setting the state variables to empty strings.

Inside the «return» statement, we render a <form> element with the registration form's fields: name, email, and password. Each input element is associated with its respective state variable and onChange event handler, which updates the state as the user types.

1. To use the «RegistrationForm» component, open the «src/App.js» file and replace its content with the following code:

import React from 'react';

import RegistrationForm from './RegistrationForm';

const App = () => {

return <RegistrationForm />;

}

export default App;

In the code above, we imported the «RegistrationForm» component and rendered it inside the «App» component.

1. Start your React application by running «npm start» in the terminal. You should now see the registration form in the browser.

As you fill in the form fields and submit the form, you'll notice that the form data is logged to the console, and the form fields are cleared.

Let's add some CSS styles to our registration form example to make it visually appealing. We'll create a separate CSS file and import it into our components. Here's how you can do it:

1. Create a new file called «RegistrationForm.css» inside the «src» folder of your React project.
2. Add the following CSS code to style our registration form:

.registration-form {

max-width: 400px;

margin: 0 auto;

padding: 20px;

background-color: #f4f4f4;

border: 1px solid #ccc;

border-radius: 5px;

}

.registration-form h2 {

margin-top: 0;

text-align: center;

}

.registration-form label {

display: block;

margin-bottom: 10px;

}

.registration-form input[type="text"],

.registration-form input[type="email"],

.registration-form input[type="password"] {

width: 100%;

padding: 8px;

font-size: 16px;

border-radius: 3px;

border: 1px solid #ccc;

}

.registration-form button[type="submit"] {

width: 100%;

padding: 10px;

font-size: 16px;

background-color: #4caf50;

color: white;

border: none;

border-radius: 3px;

cursor: pointer;

}

.registration-form button[type="submit"]:hover {

background-color: #45a049;

}

In the CSS code above, we added styles for the «.registration-form» class, as well as specific styles for headings, labels, input fields, and the submit button.

1. Next, open the «RegistrationForm.js» file and import the CSS file by adding the following line at the top:

import './RegistrationForm.css';

Finally, apply the «.registration-form» class to the <form> element in the «return» statement of the «RegistrationForm» component, like this:

return (

<form **className="registration-form"** onSubmit={handleSubmit}>

{/\* Rest of the form code \*/}

</form>

);

Save the changes, and when you run your React application, you'll see that the registration form is styled with the CSS styles we defined.

With these CSS styles, the form will have a centered layout, a light gray background, and a border. The input fields and the submit button will have consistent styles.

Feel free to modify the CSS styles according to your preferences or add additional styles to further customize the registration form's appearance.

**Documentation**

You can find more information in the official documentation. Documentation can be found at the following resource: "https://create-react-app.dev". Interesting sections: «[Advanced usage](https://create-react-app.dev/docs/advanced-configuration)», «[Backend integration](https://create-react-app.dev/docs/proxying-api-requests-in-development)».

**Conclusion**

CRA is a tool for preparing the basis of a project using React. You get a ready-made project and do not think about setting up webpack, babel and other secondary modules. All this is configured out of the box and, if necessary, the configuration can be changed.

# Python basics

Python is a high-level, general-purpose language that is used to solve modern computer problems. People have a misconception about Python these days. They think that python is a data analysis and machine learning language. However, Python is actually a general purpose programming language. Guido van Rossum invented the Python programming language in the early 1990s.

Python is a versatile and popular programming language known for its simplicity and readability. Whether you are a beginner or an experienced programmer, Python can be a valuable tool in your arsenal. In this lecture, we will cover the fundamental concepts of Python and explore some examples to solidify our understanding.

Python programming language finds its applications in various domains. Here are some areas where Python can be used:

1. Web Development: Python offers robust web development frameworks such as Django and Flask. These frameworks provide tools and libraries to build scalable and feature-rich web applications. Python's simplicity and the availability of frameworks make it an excellent choice for rapid web development.
2. Data Analysis and Visualization: Python has become the de facto language for data analysis and visualization. Libraries such as NumPy, pandas, and matplotlib provide powerful tools for handling and analyzing data, creating visualizations, and performing statistical computations. Additionally, Python integrates well with other data-related tools such as Jupyter Notebook and Apache Spark.
3. Machine Learning and Artificial Intelligence: Python's simplicity and extensive libraries like scikit-learn, TensorFlow, and PyTorch make it a preferred language for machine learning and AI tasks. These libraries provide algorithms, models, and tools to develop and train machine learning models, natural language processing (NLP) applications, computer vision systems, and more.
4. Scientific Computing: Python is widely used in scientific computing and research. Libraries like SciPy and NumPy provide functions and routines for numerical computations, optimization, linear algebra, and signal processing. The integration with Jupyter Notebook further enhances its usability in scientific research and experimentation.
5. Automation and Scripting: Python's easy-to-read syntax and rich set of libraries make it an excellent choice for automating repetitive tasks and writing scripts. From system administration tasks to file processing, Python can simplify and streamline various automation workflows.
6. Internet of Things (IoT): Python's simplicity, availability of libraries, and support for microcontrollers make it suitable for IoT applications. Python can be used to develop firmware, control sensors and actuators, and communicate with IoT devices, making it a versatile language for building IoT solutions.
7. Game Development: While Python may not be the first choice for high-performance game development, it can be used to create simple games, prototypes, and game scripts. Libraries such as Pygame provide functionality for graphics and game development, enabling developers to build 2D games and interactive experiences.
8. Desktop Applications: Python can be used to develop desktop applications with the help of frameworks like PyQt and Tkinter. These frameworks allow developers to build cross-platform GUI applications quickly.
9. Scripting for Software and DevOps: Python is commonly used for scripting tasks in software development and DevOps. It can be used to automate software testing, deployment processes, and other repetitive tasks involved in software development and infrastructure management.
10. Education and Learning: Python's simplicity, readability, and extensive documentation make it an ideal language for teaching programming and computer science concepts. Many educational institutions and online courses use Python as the primary language for introductory programming courses.

Companies using Python in their projects:

* + Google
  + Facebook
  + Instagram
  + Spotify
  + Netflix
  + Dropbox

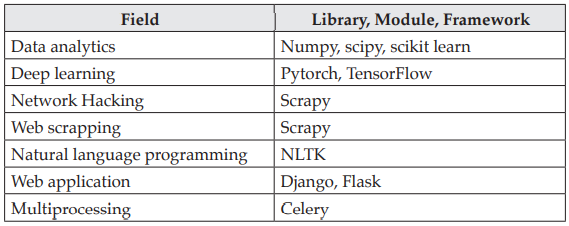
**There are many reasons to choose this particular programming language.**

**This language is multi-purpose**

Python is a general purpose language. Developers use python for data analysis, machine learning, artificial intelligence, web applications, network monitoring, ETL scripting, hacking and more.

**Extensive library and module support**

Python has a very large developer community. There are various libraries and modules available in Python:



**Ease of Code Reading**

Python code is easy to read and easy to understand. It does not contain a lot of syntax like Java does. Python uses tabulation to manage blocks of code, which is why margins are important part of Python programming. Tabulation makes the code easy to read. The Python syntax is quite brief. It helps beginners to quickly learn the basics of this language with less mental stress and quickly learn topics that are more complex.

**It is object oriented**

Python has the power of object-oriented programming (OOP). However, we can write a program without defining any classes.

**Platform Independence**

Python code is platform independent and can be run on any operating system. All you need to do is copy and paste the code.

**Python is dynamic and strongly typed**

With dynamic typing, the data type of a variable is interpreted at run time. In Python, there is no need to define a data type like «Int», «Float», etc. The following example should give more clarification regarding the variable:

>>> a = 10

>>> type(a)

<class 'int'>

>>> b = 10.9

>>>type(b)

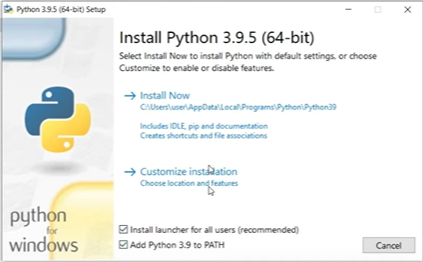
<class 'float'>

>>>

With strong typing, the type of a variable does not change during program execution. If a = 10, then the value remains 10 throughout the execution until we reassign the value of this variable.

**Installing Python**

Go to the official site <http://python.org>, download and install the latest version of Python on your computer (today it is 3.10.3). During installation, check the item "Add Python 3.10 to PATH".



Next, check if Python is installed on the computer. To do this, open the command line (Start Menu-Search-cmd) and type the command

python –version

Press "Enter". If you see the python version, it means that the installation was successful. We can also check if the Python interpreter is working correctly. In order to do this we can enter the command:

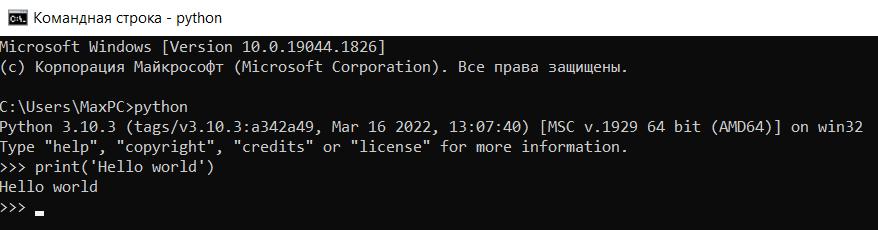
python

Press "Enter". After that, you can enter a Python instruction, for example:

print('Hello world!')

and press "Enter".

If everything works correctly, you will see the result of the command, i.e. line displayed on the screen:



You can write Python code in various code editors. All code editors and development environments known today have a built-in specification for this language.

Among the editors and development environments, the following are the most preferred:

* + Sublime Text
  + Visual Studio Code
  + Jupyter Notebook
  + PyCharm
  + Spyder
  + Atom
  + Notepad++ (this option is the least preferred)

You can read about the Python language and its use on the official website <https://www.python.org/doc/>.

The print() Function: Used to display output to the console. Example:

print("Hello, World!")

Variables and Data Types: a. Variables: Containers for storing data. Python uses dynamic typing, meaning you don't need to declare a variable's type explicitly. Example:

message = "Hello, Python!"

print(message)

Data Types: Python supports various data types, including:

* + Numbers: int, float, complex;
  + Strings: str;
  + Boolean: bool (True or False);
  + Lists, Tuples, and Sets;
  + Dictionaries c. Type Conversion: Changing the data type of a variable. Example:

x = 10

y = float(x)

Operators and Expressions: a. Arithmetic Operators: +, -, \*, /, // (floor division), % (modulo), \*\* (exponentiation) b. Comparison Operators: <, >, <=, >=, == (equality), != (inequality) c. Logical Operators: and, or, not d. Expressions: Combining variables, values, and operators. Example:

x = 5

y = 3

z = x + y \* 2

Control Flow:

a. Conditional Statements: if, elif, else. Example:

x = 10

if x > 0:

print("Positive")

elif x < 0:

print("Negative")

else:

print("Zero")

b. Loops:

For Loop: Iterates over a sequence or iterable. Example:

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

While Loop: Repeats a block of code as long as a condition is true. Example:

x = 0

while x < 5:

print(x)

x += 1

Functions:

a. Defining Functions: Creating reusable blocks of code. Example:

def greet(name):

print("Hello, " + name + "!")

greet("Alice")

b. Return Statement: Returning a value from a function. Example:

def square(x):

return x \*\* 2

result = square(5)

print(result)

**Advantages of Python:**

1. Readability and Simplicity: Python emphasizes code readability and uses a clean and easy-to-understand syntax. It employs indentation and whitespace to delimit code blocks, making it highly readable and reducing the likelihood of syntax errors. This feature makes Python an excellent choice for beginners and enhances collaboration among developers.
2. Versatility and Portability: Python is a versatile language that can be used for a wide range of applications, including web development, data analysis, machine learning, scientific computing, automation, and more. It runs on multiple platforms, including Windows, macOS, Linux, and has a large standard library and third-party packages, providing extensive functionality to developers.
3. Large and Active Community: Python has a vast and vibrant community of developers worldwide. This community contributes to the development of libraries, frameworks, and tools, making Python an ever-evolving language. The availability of resources, documentation, and community support ensures that developers can find assistance and solutions to their problems quickly.
4. Integration Capabilities: Python can easily integrate with other languages such as C, C++, Java, and .NET. This feature enables developers to leverage existing codebases and take advantage of Python's simplicity and productivity while interfacing with modules or components written in other languages.
5. Rapid Development and Productivity: Python's simplicity and expressiveness allow developers to write code quickly and efficiently. Its extensive libraries and frameworks, such as Django and Flask for web development, NumPy and pandas for data analysis, and TensorFlow and PyTorch for machine learning, provide high-level abstractions that accelerate development, reducing the time and effort required to build robust applications.

**Disadvantages of Python:**

1. Performance: Python is an interpreted language, which means it can be slower compared to compiled languages like C or Java. While Python's performance has significantly improved over the years, it may not be the best choice for highly computationally intensive tasks. However, performance-critical sections can be optimized using native extensions or by leveraging libraries written in lower-level languages.
2. Global Interpreter Lock (GIL): Python has a Global Interpreter Lock, which allows only one thread to execute Python bytecode at a time. This limitation can hinder the performance of multithreaded Python programs, as it prevents true parallelism on multiple cores. However, Python provides multiprocessing libraries that can overcome this limitation by utilizing multiple processes instead of threads.
3. Mobile and Game Development: While Python is widely used in various domains, it may not be the optimal choice for mobile app development or high-performance game development. Mobile platforms often rely on specific programming languages like Swift (iOS) and Java/Kotlin (Android). Similarly, game development often requires performance-oriented languages like C++.
4. Memory Consumption: Python's dynamic typing and high-level abstractions come at the cost of increased memory consumption compared to lower-level languages. This overhead may limit its suitability for memory-constrained environments or applications that need to handle a massive amount of data.
5. Version Compatibility: Python has undergone several major releases, with Python 2.x and Python 3.x being the most notable. Python 3 introduced significant improvements and enhancements, but it is not backward-compatible with Python 2. While Python 2 is deprecated, some legacy codebases or third-party libraries may still rely on it, creating compatibility issues for developers migrating from Python 2 to Python 3.

It's worth noting that the advantages and disadvantages mentioned above should be considered in the context of specific use cases and project requirements. Python's strengths often outweigh its limitations, making it a popular choice among developers worldwide.

These are just a few examples of where Python is commonly used, but its versatility allows it to be applied in many other areas as well. Python's vast ecosystem and active community ensure continuous growth and the development of new libraries and tools for various applications.

# Django Framework

Django is a powerful and popular framework that allows us to build web applications quickly and efficiently. It is written in Python and follows the Model-View-Controller (MVC) architectural pattern, making it easy to organize and maintain code.

Django is developed by the Django Software Foundation.

**Advantages of Django:**

* + Rapid Development: Django's built-in functionality and conventions allow developers to build web applications quickly.
  + Scalability: Django's scalability makes it suitable for handling large-scale applications with high traffic.
  + Security: Django includes built-in security features, such as protection against common web vulnerabilities.
  + Versatile: Django can be used to build various types of applications, from simple blogs to complex e-commerce platforms.

**Disadvantages of Django:**

* + Learning Curve: Django has a steeper learning curve compared to lightweight frameworks. Its extensive feature set may take some time to master.
  + Overhead: Django's comprehensive nature comes with additional overhead. It might not be the best choice for small, simple projects.
  + Opinionated: Django follows specific conventions, which might limit flexibility for developers who prefer more freedom in coding.

Django was developed between 2003 and 2005:

* + 2003: Development of the framework was started by Adrian Golovati and Simon Willison as an internal project in a newspaper;
  + 2005: Open source web framework released in 2005 and named after jazz guitarist Django Reinhardt;
  + 2008: First release (version 1.0) in September 2008.

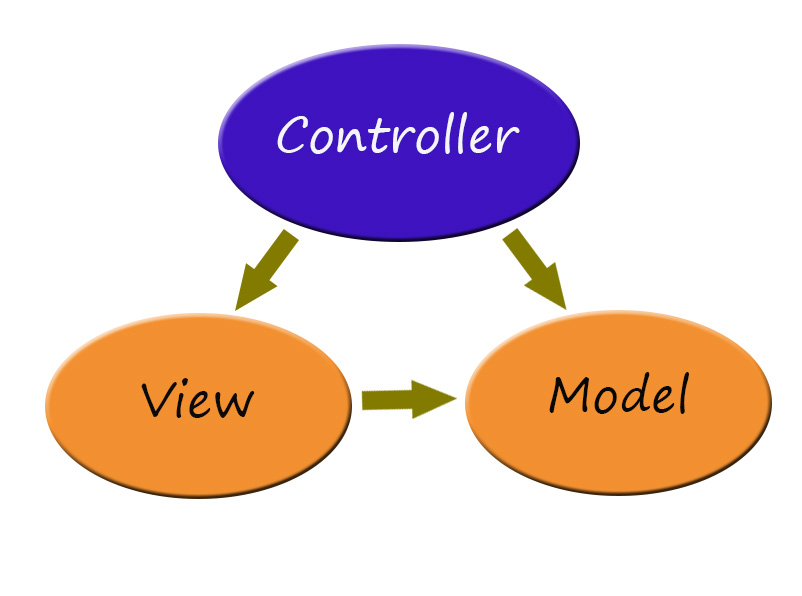
It is currently an evolving open source collaborative project involving developers around the world.

**MVC Pattern**

As mentioned above, Django is a Python web framework. Like most modern frameworks, Django supports the MVC pattern.

**What is the Model-View Controller (MVC) pattern?**

When we talk about applications that provide a user interface (web or desktop), we usually mean the MVC architecture. The MVC architectural pattern divides the application logic into three main parts: Model, View, Controller.



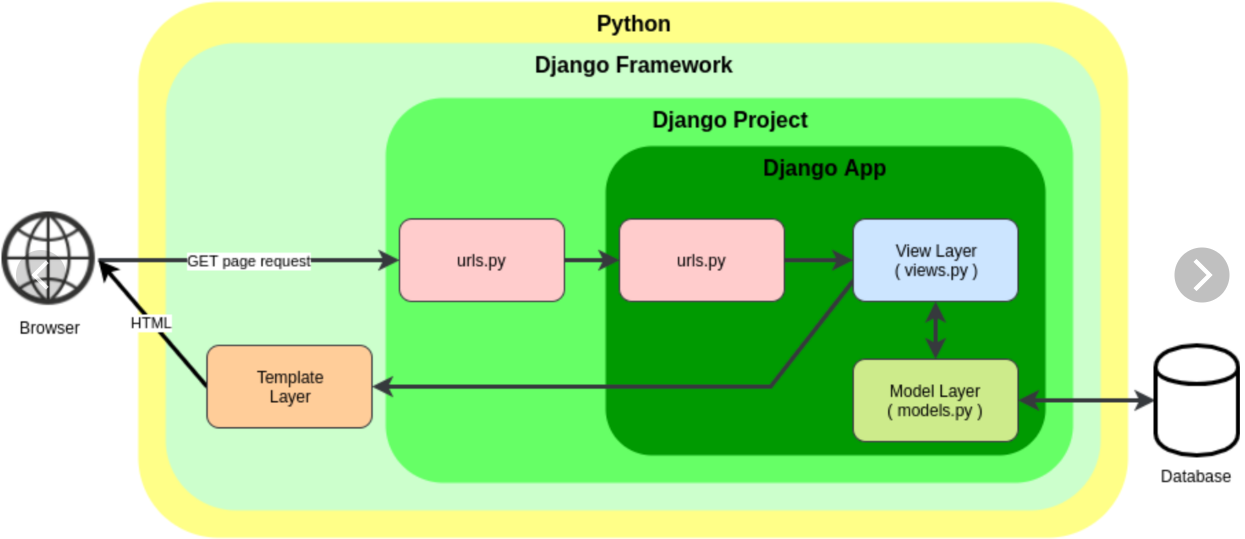
* + Models: Models define the structure and behavior of data in the application. They represent database tables and are defined in the «models.py» file within the app directory. For example, a simple model for a blog post could have fields like 'title', 'content', and 'publish\_date'.
  + Views: Views handle the logic behind each URL in your application. They process requests, interact with models, and render templates. Views are defined in the «views.py» file within the app directory.
  + Templates: Templates define the structure and layout of the user interface. They are typically HTML files with placeholders that Django fills with dynamic content. Templates are stored in a directory named 'templates' within each app directory.

**Django MVC - MVT Pattern**

Django uses a slightly modified template called the Model-View-Template (MVT).

The Model-View-Template (MVT) pattern is a little different from MVC. In fact, the main difference between the two patterns is that Django takes care of some of the Controller work itself. The controller here is the code that manages the interaction between the model and the view and produces a template as an output. The template is an HTML file mixed with Django Template Language (DTL).

The following diagram illustrates how each of the components of the MVT pattern interact with each other to serve a user request:



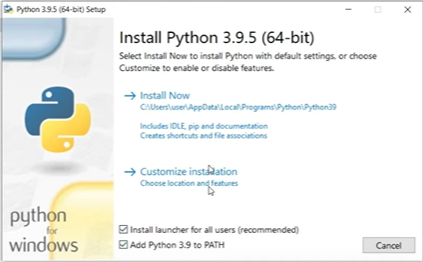
<https://github.com/fariba-Dk/django-urls-views-templates>

**Django Environment**

The Django development environment contains modules such as the Python interpreter, the Django framework, and the database.

Django is written entirely in Python, so you need to install Python to get started. It can be downloaded from the official website <http://python.org>.

Go to the official site <http://python.org>, download and install the latest version of Python on your computer (today it is 3.10.3). During installation, check the item "Add Python 3.10 to PATH".



Next, check if Python is installed on the computer. To do this, open the command line (Start Menu-Search-cmd) and type the command

python –version

Press "Enter". If you see the python version, it means that the installation was successful.

With Python installed, we can proceed to install Django.

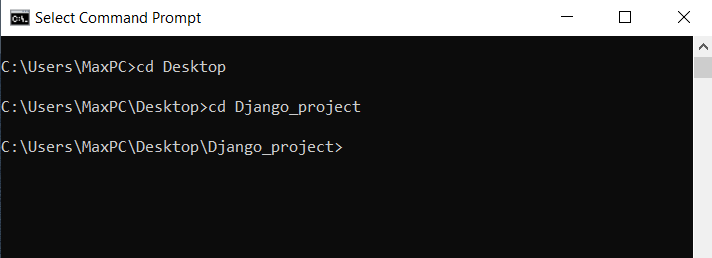
There are several different methods for installing Django, each with its own advantages in certain situations:

* + Installing Django globally with «pip». The «pip» tool is the Python package manager. It can be used to perform a system-wide installation of Django. It will usually provide the latest version of the package available. However, global (system-wide) installation is always less flexible.
  + Installation via pip in Virtualenv. The virtualenv package allows creating standalone environments for different projects. Using this technology, you can install Django in the project directory without affecting the system as a whole. This allows setting individual settings for each project. A virtual environment is a more flexible way to install a package.
  + Installing the development version with Git. To install the latest development version instead of the stable release, you need to get the code from the git repository. This will provide the latest features and fixes to the program; you can install this version either globally or locally. But keep in mind: development versions are not stable.

To install Django in a virtual environment, you need to do the following steps (using the Windows command line as an example):

To work with a virtual environment in Python, the built-in module "venv" is used.

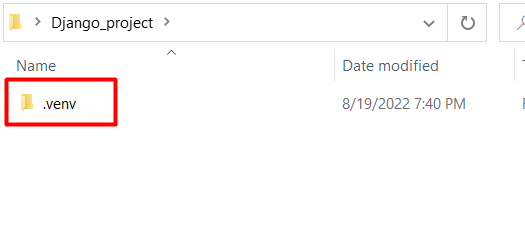
Let us create a folder "Django-project" on the desktop and go to it with the cd command:



Then, inside the current folder, run the following command:

python -m venv .venv

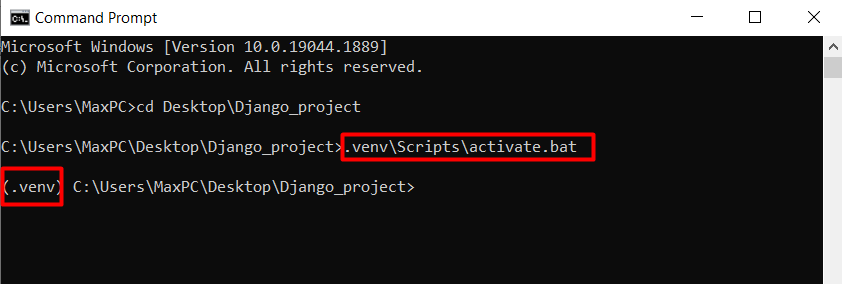
The venv module is passed the name of the environment, which in this case will be called ".venv". You can specify any name for the environment. After executing the command, the ".venv" directory will be created in the current folder.



To work with the installed virtual environment, it must be activated. To do this, run the following command (activation in the Windows command line):

.venv/Scripts/activate.bat

On the last line in the terminal, the characters "(.venv)" will appear which tells us that the virtual environment is activated.



***Note:*** *The virtual environment must be activated each time you work on a particular project. After finishing work, you should deactivate the virtual environment with the command: "deactivate".*

Install the Django framework in the created and activated virtual environment. To do this, enter the following command and press Enter:

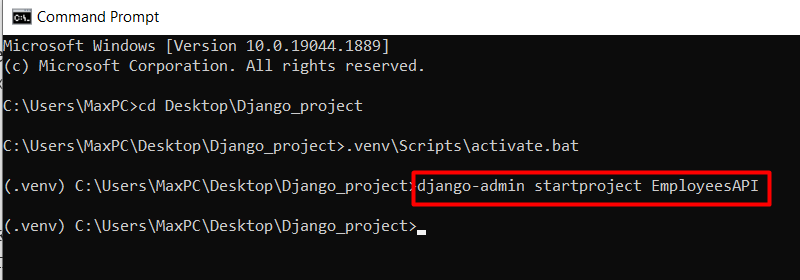
python -m pip install Django

The framework will be installed locally in our virtual environment.

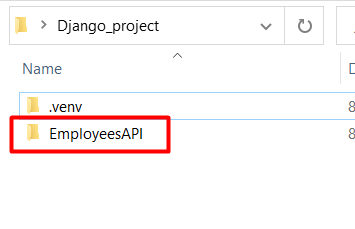
Django also installs the Django-admin special utility in the virtual environment directory, which provides a set of commands for managing a Django project.

Let us create the first project, which will be located inside the folder with the virtual environment. To do this, enter the following command:

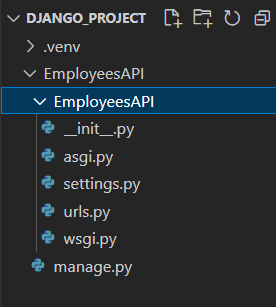
django-admin startproject EmployeesAPI



After that, you will see a new folder with the created project:



Let’s open the project in the VS Code editor. The project structure will look like this:



Let's look at the purpose of the files in the created project:

* + **\_init\_.py** is an empty file that indicates that the project is a Python project or a Python module;
  + **asgi.py** is the entry point for asgi (Asynchronous Server Gateway Interface) servers;
  + **wsgi.py** - entry point for wsgi (Web Server Gateway interface) servers;
  + **urls.py** is a file that declares all the routes needed for the project;
  + **settings.py** is a file that contains all the necessary project configuration settings;
  + **manage.py** is a service file that interacts with a Django project.

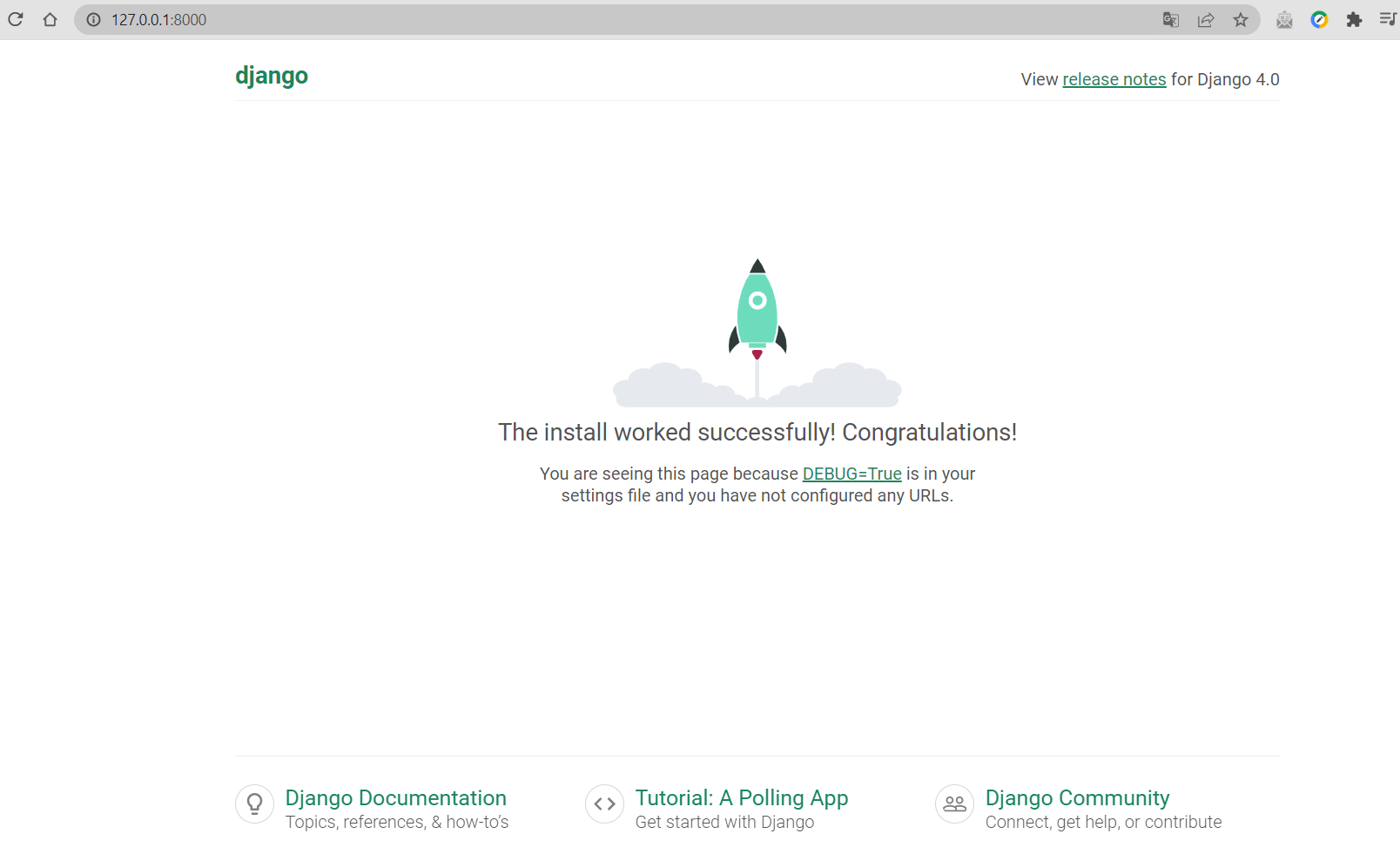
Now let's run the project for execution and look at its work. This can be done by running the following command:

python manage.py runserver

The application will run on port 8000.



Copy the Url and paste it into your browser's search bar:



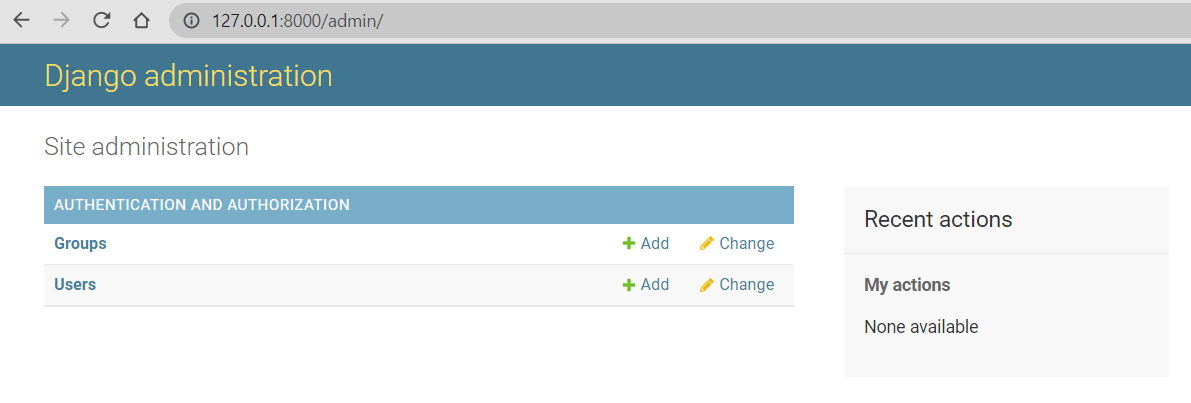
After running the project, we will see a basic Django page.

If you navigate to /admin, you can enter to the Django project's admin panel.

To log in, you need to have an account. To create an account with full access to the project and with all the necessary permissions (superuser), you need to run the following command in the terminal from the project folder where the manage.py file is located:

python manage.py createsuperuser

The system will ask you to enter your username and password. After creating an account, you will be able to log in the administration panel with the specified credentials.



Django provides a solid foundation for building web applications, and its versatility and scalability make it a popular choice among developers.

To find more information on the Django framework, you can refer to the following resources:

1. Django Official Documentation: The official Django documentation is an excellent starting point. It provides comprehensive and detailed information about Django's features, concepts, and how to use them. You can access the documentation at <https://docs.djangoproject.com/>.
2. Django Project Website: The Django project website (<https://www.djangoproject.com/>) offers various resources, including tutorials, documentation, community forums, and links to additional learning materials.
3. Django Community: The Django community is active and supportive. You can participate in discussions, ask questions, and seek help from other Django developers. The official Django forum (<https://forum.djangoproject.com/>) and the Django subreddit (<https://www.reddit.com/r/django/>) are popular platforms to engage with the community.
4. Django Packages: Django Packages (<https://djangopackages.org/>) is a website that showcases third-party packages and extensions available for Django. It's a valuable resource to discover additional functionality and tools that can enhance your Django projects.
5. Online Tutorials and Courses: There are numerous online tutorials and courses available that cover Django in depth. Platforms like Udemy (<https://www.udemy.com/>), Coursera (<https://www.coursera.org/>), and Real Python (<https://realpython.com/>) offer Django-specific courses that range from beginner to advanced levels.
6. Books: Several books are dedicated to Django development. Some popular titles include "Two Scoops of Django" by Daniel Roy Greenfeld and Audrey Roy Greenfeld and "Django for Beginners" by William S. Vincent. These books provide comprehensive guidance and best practices for Django development.
7. GitHub: Exploring open-source Django projects on platforms like GitHub (<https://github.com/>) can be a great way to learn from real-world examples. You can find code repositories, contribute to projects, and gain insights into how Django is used in different scenarios.

# API

APIs (Application Programming Interfaces) play a vital role in modern software development, enabling communication and interaction between different software systems, applications, and services. In this lecture, we will delve into the fundamental concepts of APIs, understand their significance, and explore various examples of API usage.

1. What is an API? An API, or Application Programming Interface, is a set of rules and protocols that allows different software applications to communicate and interact with each other. APIs define how software components should interact, what data can be exchanged, and the methods or functions that can be used to access and manipulate that data.
2. API Components:
   * Endpoints: An endpoint is a specific URL or URI that represents a unique resource or functionality provided by an API. It serves as the entry point for accessing and interacting with the API;
   * Request Methods: APIs use standard HTTP methods such as GET, POST, PUT, DELETE, etc., to define the type of operation to be performed on the resource;
   * Request Parameters: APIs often accept parameters as part of the request to specify additional information or modify the behavior of the API call;
   * Response: The response from an API call contains the data or status information requested by the client application. It is typically formatted in a specific structure, such as JSON or XML.
3. Types of APIs:
   * Web APIs: These APIs are designed to expose the functionality of web services over the internet, allowing developers to access and use services such as weather data, maps, social media platforms, and more;
   * Library or Framework APIs: These APIs are provided by programming languages, frameworks, or libraries and offer pre-built functions and classes to simplify software development. Examples include the Python Standard Library, Java API, and jQuery API;
   * Operating System APIs: These APIs allow developers to interact with the underlying operating system, accessing system resources, file management, network operations, and more;
   * Database APIs: These APIs facilitate interaction with databases, allowing developers to perform operations such as querying, updating, and managing data;
4. API Usage Examples:
   * Social Media APIs: Platforms like Twitter, Facebook, and Instagram provide APIs that allow developers to integrate social media functionalities into their own applications. For instance, retrieving user profiles, posting tweets or status updates, or fetching news feeds;
   * Payment Gateway APIs: Companies like PayPal, Stripe, or Braintree provide APIs to handle payment processing in e-commerce applications, allowing developers to accept payments securely;
   * Weather Data APIs: Services like OpenWeatherMap offer APIs that provide developers with real-time and forecast weather data for integration into their applications, enabling weather-based functionality;
   * Mapping APIs: Providers such as Google Maps, Mapbox, or OpenStreetMap offer APIs to access map data and geolocation services, empowering developers to embed interactive maps or build location-based applications.

APIs act as bridges between different software systems, enabling seamless integration and communication. Understanding the basic concepts of APIs is essential for developers to leverage the vast array of functionalities offered by various APIs.

# REST API

REST API, or Representational State Transfer API, is an architectural style and approach for designing networked applications. It is widely used in web development to create APIs that are scalable, stateless, and can be easily consumed by clients. REST API leverages the existing HTTP protocol and its methods to enable communication and data transfer between client applications and servers.

**Key Concepts of REST API:**

1. Resources: REST APIs expose resources, which are the key entities or objects that the API deals with. A resource can be anything that can be uniquely identified, such as an article, user, product, or any other data entity. Each resource is typically represented by a unique URL or endpoint.
2. HTTP Verbs/Methods: REST APIs utilize the HTTP methods (also known as verbs) to define the operations that can be performed on resources. The commonly used HTTP methods in REST API are:
   * GET: Retrieve a representation of a resource or a collection of resources.
   * POST: Create a new resource;
   * PUT: Update an existing resource or create a new resource if it doesn't exist;
   * DELETE: Delete a resource.
3. Uniform Resource Identifier (URI): REST APIs use Uniform Resource Identifiers (URIs) to uniquely identify resources. URIs are the URLs or endpoints that clients use to access and manipulate the resources. For example, https://api.example.com/users could be the URI for accessing a collection of user resources.
4. Stateless Communication: REST APIs are stateless, meaning that each request from a client to the server should contain all the necessary information for the server to understand and process the request. The server does not maintain any information or state about the client's previous requests. This design principle improves scalability and simplifies server-side implementation.
5. Representation of Resources: In REST APIs, resources are represented in a specific format, such as JSON (JavaScript Object Notation) or XML (eXtensible Markup Language). These formats define the structure and data representation of the resources exchanged between the client and the server.

**Numerical codes - HTTP statuses**

Each REST API request reports the results of the work with numerical codes - HTTP statuses.

HTTP status codes are three-digit numerical codes that are returned by a server in response to a client's request to indicate the status of the request or the server's ability to fulfill it. These codes are an essential part of the HTTP protocol and provide valuable information about the outcome of an HTTP request. Here are some commonly encountered HTTP status codes along with their general categories:

1. Informational (1xx):
   * 100 Continue: The server acknowledges the initial part of the request and expects the client to proceed with the rest.
   * 101 Switching Protocols: The server agrees to switch protocols as requested by the client.
2. Success (2xx):
   * 200 OK: The request was successful, and the server returns the requested content.
   * 201 Created: The request resulted in the successful creation of a new resource.
   * 204 No Content: The server successfully processed the request, but there is no content to return.
3. Redirection (3xx):
   * 301 Moved Permanently: The requested resource has been permanently moved to a new location.
   * 302 Found: The requested resource has been temporarily moved to a different location.
   * 304 Not Modified: The client's cached version of the requested resource is still valid, and there is no need to transfer it again.
4. Client Errors (4xx):
   * 400 Bad Request: The server cannot process the request due to malformed syntax or invalid parameters.
   * 401 Unauthorized: The client needs to provide valid authentication credentials to access the requested resource.
   * 404 Not Found: The requested resource could not be found on the server.
5. Server Errors (5xx):
   * 500 Internal Server Error: A generic server error occurred, and the server cannot fulfill the request.
   * 503 Service Unavailable: The server is temporarily unable to handle the request due to overloading or maintenance.
   * 504 Gateway Timeout: The server acting as a gateway or proxy did not receive a timely response from an upstream server.

**Advantages of REST API:**

1. Scalability: REST APIs are designed to be scalable, allowing multiple clients to interact with the server concurrently. The stateless nature of REST facilitates horizontal scaling by enabling load balancing across multiple servers.
2. Simplicity: REST API leverages the simplicity and universality of HTTP methods and status codes, making it easy to understand and use. It follows a standard and predictable design pattern, which aids in the development and integration of APIs.
3. Compatibility: REST APIs are compatible with a wide range of platforms, programming languages, and devices. As long as the client can make HTTP requests and understand the representation format, it can interact with a REST API.
4. Caching: REST APIs can take advantage of HTTP caching mechanisms, allowing clients to cache responses and reduce the number of round trips to the server, thus improving performance and reducing network bandwidth usage.

REST APIs are used for a variety of purposes in modern software development. Here are some common use cases and applications of REST APIs:

1. Web Application Development: REST APIs are extensively used to build web applications, both on the client-side and server-side. They allow client applications to interact with server resources, retrieve data, submit data, and perform various operations.
2. Mobile Application Development: REST APIs play a vital role in mobile app development. Mobile applications can consume REST APIs to access server-side resources, fetch data, send user input, and perform transactions. This enables seamless integration of mobile apps with backend systems.
3. Integration of Systems and Services: REST APIs facilitate the integration of different software systems, enabling them to communicate and share data. APIs allow applications to access functionality and services provided by third-party systems, such as social media platforms, payment gateways, email services, geolocation services, and more.
4. Microservices Architecture: REST APIs are often used in microservices architectures, where an application is built as a collection of small, independent services. Each microservice exposes a REST API that provides a well-defined interface for communication and interaction between services.
5. Internet of Things (IoT): REST APIs are utilized in IoT applications to enable communication between IoT devices and other systems. Devices can send data to the server or retrieve data from it through RESTful interactions, allowing for remote control, monitoring, and data exchange.
6. Webhooks and Event Notifications: REST APIs can be used to set up webhooks or event notifications, allowing systems to send real-time updates or trigger actions based on specific events. This is commonly used in applications like chatbots, real-time collaboration tools, or notification services.
7. Content Management Systems: REST APIs are employed in content management systems (CMS) to expose content and manage resources such as articles, images, user profiles, and more. They enable developers to build custom front-end interfaces or mobile apps that interact with the CMS.
8. Open Data and Public APIs: REST APIs are used to make data and services available to the public. Organizations and government bodies often provide public APIs to share data such as weather information, transportation data, financial data, and more, enabling developers to create innovative applications on top of that data.

These are just a few examples of how REST APIs are utilized in different domains. REST APIs provide a standardized, scalable, and flexible approach to building software systems and enabling seamless integration and communication between applications and services.