Chapter 1

Fundamentals of Web Programming

What is in This Chapter ?

This first chapter is just a very brief introduction to the idea of web programming. Its aim is to give you an idea of what the course is all about. By learning the fundamentals of web programming you will be able to build reliable, interactive websites and you will be able to learn new tools and technologies much easier.

1.1 Internet History

The Internet started in the late 1960s as a way for computers at universities and research centers to connect and share information, even if parts of the network failed. Early networks like the ARPANET experimented with connecting computers reliably, and later, standard rules called protocols (like TCP/IP) were developed so all these networks could communicate with each other. Over time, this grew into the global Internet, which supports services like email, file sharing, and eventually … websites.

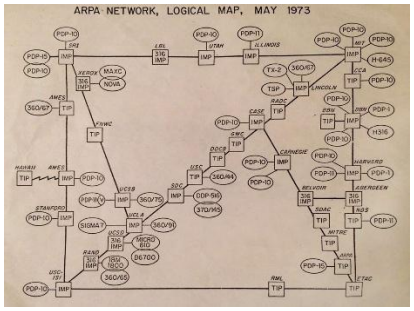


Image from:
https://www.thoughtco.com/arpanet-the-worlds-first-internet-4072558

The World Wide Web (WWW) is a system of linked web pages and content accessed through browsers using the internet.

The **WWW** was built on top of the Internet, was invented in the late 1980s and early 1990s by Tim Berners-Lee to make sharing documents easier. The first web pages were created using **HTML** … which is a simple way to structure content with headings, paragraphs, and links to other pages. At first, pages were mostly plain text and static, but as the web grew, new tools appeared: **CSS** (Cascading Style Sheets) to control appearance and layout separately from **HTML**, and **JavaScript** to add interactive features like buttons and forms.

Together, these technologies allowed the web to become the interactive, visually appealing platform we use today. Here are some terms we will use throughout the course:

Content is the text, images, videos, and interactive pieces displayed on a webpage for visitors to view or use.

A web resource is any single item on the web with its own address (URL) … could be an entire webpage, a single image file, a downloadable document, or even an API

A Uniform Resource Locator (URL) is a web address we type into a browser to find a specific web resource. We can think of it as the "home address" of a webpage … it tells the browser exactly where to go to get the content.

A website is a collection of pages we visit mainly to read or view information, like news, blogs, or company sites.

A web app is a program we can use in our browser to do tasks, like email, shopping, or editing documents.

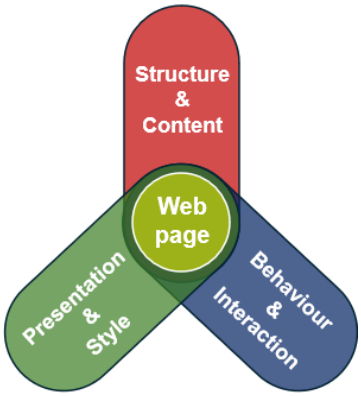
Over the past 25 years, the Internet and the web have grown far beyond their original purpose of sharing simple documents. Websites have become more interactive and visually rich, with images, videos, and dynamic content that respond to user actions. Internet connections have become faster and more reliable, making it possible to stream media, play online games, and use cloud-based applications. At the same time, tools and platforms have made it easier for people with little technical experience to create websites and apps.

1.2 Web Programming

In this course, we will learn all about creating webpages and how they work. We will explore what happens at the ***front-end***, which is everything the user sees and interacts with in their browser such as text, images, buttons, forms, and other interactive elements. At the same time, we will also look at the ***back-end***, which is everything happening behind the scenes on the server. The back-end stores data, processes requests, and sends the appropriate information to the user's browser so the webpage works correctly. By understanding both sides, you will gain a complete picture of how websites are built and how the different parts work together to create a smooth and interactive user experience.

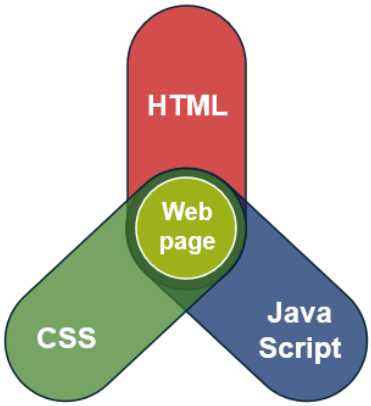
To better understand how front-end and back-end work together, it helps to break down a webpage into its three main aspects that are combine together:

- **Structure** – the content of the webpage
  - (e.g., headings, text, buttons, etc..)
- **Presentation** – the style and layout of the webpage
  - (e.g., colors, fonts, spacing, etc..)
- **Behaviour** – how the webpage interacts with the user
  - (e.g., validating forms, animations, dynamic content, etc..)



Notice how the Webpage is in the center, overlapping all three aspects. We will be discussing how to do all of this in the course, so that you have a full understanding of creating interactive webpages. More specifically, we will explain how to implement all these as follows:

- **Structure** = **HTML**
- **Presentation** = **CSS**
- **Behaviour** = **Javascript**



By separating the roles of **HTML**, **CSS**, and **JavaScript**, we can ensure each part of the code focuses on a distinct function, which is …

> ***Separation of concerns** is a design principle in software engineering that involves organizing a program so that different parts handle distinct, independent aspects of functionality.*

This makes code easier to manage, understand, and maintain. The result is that…

- ✓ web pages are more reliable, work across different devices, and are easier to maintain.
- ✓ accessibility and usability for all users is improved.
- ✓ data and content are easier to use for connected devices and intelligent systems.

In this course, we will learn web programming primarily through building and viewing pages in a web browser, since browsers provide a simple way to see how **HTML**, **CSS**, and **JavaScript** work together. However, the same technologies are also used in many apps we use every day. For example, messaging apps like **WhatsApp**, video streaming apps like **YouTube**, and shopping apps like **Amazon** all rely on the same core web technologies to display content, handle user input, and provide interactive features. Learning these basics in the browser gives us skills that apply to both websites and many types of applications.

When learning web programming, as computer science students … it is easy to focus on **HTML**, **CSS**, and **JavaScript** as the core skills. However, building a website that truly engages users is about more than just writing code … it is largely about art, style, and creativity. Choosing the right colors, fonts, layout, and visual elements can make a website intuitive, appealing, and memorable. Even the most technically correct site can feel dull or confusing if these creative aspects are ignored. In many ways, the coding provides the framework, but the design and creative choices give a site its personality and impact.

The good news is that we don't need to be a professional designer to create an attractive website. Web development provides many tools, frameworks, and templates that help guide layout, colors, and style choices. Learning the fundamentals of **HTML**, **CSS**, and **JavaScript** is the first step, and with practice, even small adjustments can make a site look polished. Creativity can grow over time, and collaboration with others or using pre-made resources can make it easy to build visually appealing pages without relying solely on natural design skills.

This course can feel overwhelming at first because there are many new "languages" and tools to get familiar with at the same time. **HTML**, **CSS**, and **JavaScript** each have their own rules, and then there are additional libraries, frameworks, and different versions of these tools that can make it seem even more complicated. It's a lot like trying to learn a bunch of programming languages at once because you will have to remember different grammar, syntax, and ways of expressing ideas. The best way to handle all of this is not to try and memorize everything. Just try to grasp the main ideas and focus on learning basic templates. You will always need to "look things up" because if you are a *normal* person … you won't be able to remember all the details that you learn here.

At its core, almost everything we do on the web is a conversation between a ***client*** (i.e., browser or app) and a ***server*** (i.e., the software that provides resources).

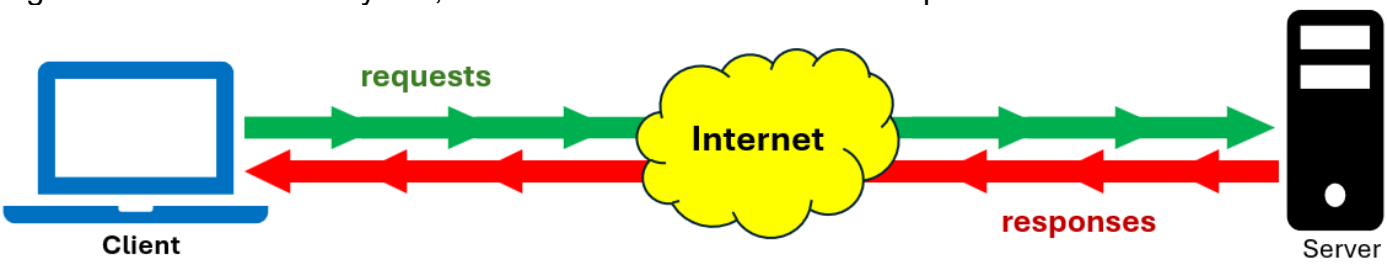
The request to the server usually using one of four methods:

- ask for a page, data, image or some other file
- send new data to the server
- update existing data
- remove data

The server responds with a **status code** (i.e., “everything was ok”, “I can't find that”, “the website is down”, etc..) and often some **content** (i.e., a webpage, some code, an image, etc.).

Everything else that we will get into in this course (e.g., **HTML**, **CSS**, **JavaScript**, **AJAX**, templating engines such as **Pug**/**EJS**, databases such as **MongoDB** and authentication systems) is ultimately just part of making, handling, and responding to these requests.

If you keep this … **request in → response out** … cycle in mind, the everything that you learn will fit together much more easily. So, the whole course is all about this picture:



All the different tools we will learn to use to make websites are there to make our work easier. They help us build pages faster, keep everything organized, and add features without having to do everything by hand. This lets us focus on creating websites that are useful and enjoyable for people. At the core, though, it all comes down to a simple idea: asking for something and getting back a response.

1.3 Course Overview

In this course, we will learn how to build websites from the ground up. We will start with the basics … putting content on a webpage with **HTML** and making it look good with **CSS**. Then we will add interactivity with **JavaScript** and see how the page’s structure (the **DOM**) lets us update content dynamically. From there, we will explore how browsers and servers talk to each other, how to fetch or send data without reloading pages using **AJAX**, and how to run **JavaScript** on the server with **Node.js** and use helpful tools from **NPM**. We will also learn how to create reusable web pages with templates, organize server code with **Express**, and communicate data in a predictable way with **REST**ful design. We will cover storing and managing information using **MongoDB**, protecting it with authentication, and making database work easier with **Mongoose**. Finally, we will see how all these skills apply to building mobile apps, so you can make web projects that work anywhere.

So, this course will give you the skills to build real, working web applications … both the parts users see and the behind-the-scenes server logic. You will get a solid foundation for an entry-level web developer role because employers are often looking for web developers, full-stack engineers, and mobile app developers.

Understanding these technologies also builds a strong foundation for other areas in computer science (e.g., software development, cloud computing, and data-driven applications). So, it will make you a more versatile programmer … ready for a wide range of tech careers.

Here is a list of the chapters that we will go through with a simple explanation of why we are learning it and why we are doing it in this order:

1. **Introduction** – this chapter that explains what the course is all about.
2. **HTML** – the "skeleton" of a webpage … it tells the browser what content to show and organizes it into a basic structure.

   Taught first so we can put content on a page before worrying about style or interactivity.
3. **CSS** – the "clothes" of a webpage … it styles the content with colors, fonts, and refined positioning to control the visual layout.

   This usually comes next, so we can make the content look good after it's in place.
4. **JavaScript** – the "muscles" of a webpage … it adds interactivity, responds to user actions, and updates content dynamically.

   *Added to make webpages interactive, enabling live updates, animations, and responses that **HTML** and **CSS** alone cannot handle.*
5. **Document Object Model** – a structure of all the content on a page that programs can use to update or interact with.

   *Shows how to dynamically update the page with **JavaScript**, allowing content and layout to change in response to user actions without reloading the page.*
6. **Client-Server Architecture** – explains how browsers (i.e., clients) ask servers for information and get back responses.

   Introduced after front-end basics so we understand where data comes from.
7. **AJAX** – a way to get or send data without refreshing the page.

   After learning client-server basics, it shows how to make pages faster and smoother.
8. **Node.js & NPM** – lets us to run our **JavaScript** on the server with **NPM** being a toolbox of extra features and libraries to make development easier.

   Taught now so we can start running code on the server and use helpful tools.
9. **Template Engines (EJS & PUG)** – tools to create webpages with reusable pieces.

   *After learning server-side **JavaScript**, templates make building pages faster and less repetitive.*
10. **Express** – a framework that organizes server routes and requests.

    Once we have templates, this helps us manage server logic more easily.
11. **RESTful Design** – a way to organize data requests and responses.

    Taught now so server communication is clear and predictable.
12. **MongoDB** – a flexible database to store information.

    After server basics, we need to know how to save and retrieve data for our pages.
13. **Authentication** – checking who a user is (e.g., logging in).

    Once data exists, we need a way to protect it and identify users.
14. **Mongoose** – a tool to work with **MongoDB** more easily.

    After learning databases and authentication, Mongoose helps us organize, validate, and interact with database data more efficiently in a structured way.
15. **Mobile Apps** – programs for phones and tablets using the same web technologies.

    Finally, we see how everything applies to mobile devices, tying all the skills together.