**Overview of Web and Cloud Development**

When you’re starting out as a Web Developer, it can be difficult to determine what you

need to learn and what order you should learn it in.

Understanding how the websites that you’re already familiar with are constructed and

delivered to you is a good starting point.

Let’s review the basics of how you interact with a website.

You launch an internet browser – there are lots available:

Google Chrome, Microsoft Edge, Mozilla Firefox, and Apple Safari are some of the most popular.

The browser has an address bar, into which you enter a URL, like www.IBM.com.

The browser then contacts the server with the name and requests the information that

makes up the website.

The server then sends a response, which contains the data that the client requires to display

the website.

For most websites, the server will return:

HTML, which defines the structure of the page, but doesn’t look very attractive

CSS, which adds styles and flair to the page and JavaScript, which adds interactivity and

dynamic content.

Content displayed by websites can contain elements that are either previously stored

on the server (called “static”) or generated each time they are requested by the client

(called “dynamic”).

Dynamic elements can involve information coming from other systems and applications, such

as databases.

Most websites contain static and dynamic elements to provide the best user experience.

Cloud Applications are similar to Websites in that they request content that a server

returns.

Cloud Apps are built to work seamlessly with a Cloud-based back-end infrastructure, Cloud-based

data storage and data processing, and other Cloud services, making them very scalable

and very resilient.

The environment for building websites and Cloud Applications is divided into two primary

areas:

front-end and back-end.

The front-end deals with everything that happens at the client-side, everything the user

can see and interact with.

You can choose to specialize in front-end coding, using HTML, CSS, JavaScript and related

frameworks, libraries, and tools.

The back-end deals with everything that happens on the server before the code and data are

sent to the client.

The back-end coding usually handles the logic and functionality that make the website or

app work, and the authentication processes that keep data secure.

Back-end developers may also work with relational or noSQL databases, even collaborating with

database administrators in bigger projects.

Full-stack developers have skills, knowledge, and experience in both front-end and back-end

environments.

Whichever way you choose to specialize, you will need the appropriate tools to help you

work.

The first tool most developers add to their resources is a code editor.

Developers also need tools to integrate, build, compile, and debug code.

Integrated Development Environments or IDEs incorporate some of these additional capabilities

beyond just code editing and make it easier to build and manage your code.

Good IDEs support multiple languages and integrate with management and storage tools like Git

and GitHub.

Other useful features are custom extensions and themes for supporting your working style

and environment.

Examples for code editors and IDEs include Sublime Text, Atom, Vim, VS Code, Visual Studio,

Eclipse and NetBeans.

In this brief overview, you learned about:

The basic communication between client and servers.

How websites are built and displayed,

Front-end development is about what happens on the client side,

Back-end development is about what happens on the server,

Full-stack development incudes both front-end and back-end development,

IDEs will help you create and manage your code.

**Learning Front-End Development**

Internet websites offer lots of different services, one of the most popular being online

shopping.

When you explore an online shopping website, navigating through pages, choosing different

product categories, or comparing products, you are interacting with the front-end of

a website.

Let us see how the front-end of a website is developed by front-end developers.

For this, we need to understand how a website is made.

To create a website, web developers usually use Hypertext Markup Language (HTML), Cascading

Style Sheets (CSS) and JavaScript. These languages are designed to work in conjunction with each

other.

HTML is used to create the physical structure of a website.

The physical structure contains elements such as text, links, images/videos, page dividers

and buttons.

The HTML code ensures a proper formatting of all text and image elements so that browsers

display the page consistently.

The backend developer codes the structure of the website.

A website is like a house which has only been constructed.

Just like we need interior designers to add style to a space, we need front-end developers

to add the necessary glamor and appeal to a website.

When you order products from any website, you realize that the pages have a pleasing

font, attractive colors, and are easy to navigate.

Developers use CSS to create stylish websites.

CSS provides front-end developers with a standard method to define, apply, and manage different

sets of style characteristics for a website and each of its components.

CSS ensures uniformity in look and feel, style, colors, fonts, designs and layouts.

So, HTML is used to create the structure and CSS is used to design it and make it appealing.

CSS is also used to create websites that have cross browser compatibility which means that

they are compatible with multiple browsers and multiple devices such as PC, mobile devices,

iPads etc.

Online shopping websites are intuitive, interactive and quick to load. This is where

JavaScript comes into the picture.

JavaScript is an object-oriented programming language that is used in conjunction with

HTML and CSS to add interactivity to a website.

For example, you use HTML to add a login button to a page, and CSS to style that button.

You then use JavaScript to add log-in functionality to that button.

A new front-end development language is Syntactically Awesome Style Sheets (SASS) called SAAS.

It is an extension of CSS that is compatible with all versions of CSS.

SASS enables you to use things like variables, nested rules, and inline imports to keep things

organized.

SAAS allows you to create style sheets faster and more easily.

Another language that is being used now is Learner Style Sheets (LESS) or LESS.

LESS enhances CSS, adding more styles and functions. It is backwards compatible with

CSS.

Less.js is a JavaScript tool that converts the LESS styles to CSS styles.

Using all these languages, websites are designed as reactive and responsive.

Reactive or adaptive websites display the version of the website designed for a specific

screen size.

For example, a website can provide more information if opened on a PC than when opened on a mobile

device.

Responsive design of a website means that it will automatically resize to the device

it is being accessed from.

For example, if you open up a products website on your mobile device, it will adapt itself

to the small size of the screen and still show you all the features.

A JavaScript framework is an application framework that is written in JavaScript.

Programmers can manipulate the different functions, use them wherever required and can create

device responsive applications.

A few examples of several frameworks being used are:

Angular framework: an open-source framework and is being maintained by Google.

Angular frameworks allow websites to render the HTML pages quickly and efficiently.

It has built-in tools for routing and form validation.

React.js has been developed and maintained by Facebook.

It is a JavaScript library that builds and renders components for a web page.

It is not a complete suite of tools.

For example, routing is not a part of this framework and will need to be added using

a third-party tool.

React.js only helps build and drop components into a page.

Vue.js is maintained by the community and its main focus is the view layer which includes

user interface, buttons, and visual components.

It is flexible, scalable and integrates well with other frameworks.

It is very adaptable. It can be a library, or it can be the framework.

The task of a front-end developer evolves continuously.

The technologies are upgraded constantly and so front-end developers needs to keep

upgrading the websites that they create.

The websites that they create should work in multiple browsers, multiple operating systems

and multiple devices.

**Introducing Application Development Tools**

Welcome to Introducing Application Development Tools.

Getting your Cloud App from the ideas stage to fully formed, written, and deployed is

a long process, but there are many tools which will help you along the way.

A cloud application developer’s workbench includes:

Version Control, Libraries, and Frameworks.

When many developers are working on the same project, knowing what order changes were made,

thereby creating a new version of the source code, becomes overridingly important.

Version control systems keep track of what changes were made when and by whom and resolve

any conflicts between changes.

For developing your code, version control can be useful even when you are the sole contributor

on a project.

Properly used, it can give you a way to revert to an older version of your code if something

goes wrong and gives you some basic information about how the code developed over time.

Version control functionality is generally tied to the storage system you are using,

which is why a code repository is recommended, even for beginners.

Git and GitHub are extremely popular for source code storage and management.

Git stores files in repositories where you can track changes, split code into different

branches for more focused development, and then merge them back into the main body of

code.

Libraries are collections of code, like standard programs and subroutines, that you can use

within your code.

For example, you might want to include a navigational feature, like a carousel. A code library

can supply you with the code for that so that you don’t have to spend the time and energy

creating one from scratch.

Being able to reuse code in this way makes developing your app much quicker and easier.

Multiple code libraries can be integrated into your existing project.

As you discover a need for a specific function or feature, you can research an appropriate

library.

You determine when to call the required method as needed, and control returns to the program

flow once the subroutine is finished.

When you use a code library, you are in control.

Code libraries are generally used to solve a specific problem or add a specific feature

set.

Either way, there are lots for you to choose from, so do your research.

Here are some examples of code libraries:

jQuery is a JavaScript library that simplifies Dom manipulation.

Email-validator is a small library that checks an email address is correctly constructed

and valid.

Apache Commons Proper is a repository of reusable Java components.

Frameworks provide a standard way to build and deploy applications.

You can think of a framework as being a skeleton that you can extend by adding your own code,

providing a scaffold on which to build your apps.

The framework you intend to use must be determined early in your development planning and used

right from the beginning.

New frameworks can’t be incorporated into an existing project.

Your chosen framework dictates the architecture of your program and controls the program flow.

The framework determines which subroutines and methods will be called when. When working

with a framework, there is a specific structure that you must follow.

The framework is calls on your code, rather than you calling on the framework.

Frameworks are less flexible than libraries, allowing you less control, but they do provide

good standardization and can help you create efficient code.

To use an analogy, if you are a carpenter building a house, the framework is the frame

that you add to – bricks on the outside, plasterboard on the inside, and so on.

The frame acts as a guide for how the house is constructed.

Here are some examples of frameworks.

AngularJS is a JavaScript-based framework for dynamic web applications.

Vue.js is a JavaScript framework focused on the user interface.

Django is a framework that uses Python for web development.

Frameworks define the workflow that you must follow, unlike libraries, which allow you

to call functions as and when required.

When using a framework, it can sometimes feel like you, as a developer, are not in full

control of the development process.

This sense of the framework and its predefined workflow controlling the development process

is referred to as inversion of control.

Frameworks that have a lot of control are known as opinionated – they have opinions

on how their workflow should be used and remove a lot of the decisions you would otherwise

have to make about how code is written, the location of files, and even file names.

Frameworks often include their own libraries, which they call when needed.

Inversion of control allows you to create standardized apps, and takes away a lot of

the tedious configuration work, so you can focus on the code for your app.

In this video, you’ve learned about some of the tools that you will utilize in your

career as a developer including:

Version control, libraries, frameworks

**More Application Development Tools**

Welcome to More Application Development Tools.

Let’s look at some tools which can help you get your app built and deployed:

CI/CD, Build Tools, Packages, and Package Managers.

CI/CD refers to the practices of continuous integration and either continuous delivery

or continuous deployment.

CI/CD is a best practice for devops teams enabling developers to deliver frequent changes reliably.

Implemented through a build-automation server, Continuous Integration (CI) ensures that all

the code components work together smoothly.

A CI build environment enables you to integrate newly developed code frequently, at least

every day, if not every hour, depending on how quickly the project changes.

Continuous delivery (CD) begins where CI ends. The CI process automatically builds and tests

your code, then CD deploys all code changes in a build to a testing or staging environment.

A build tool transforms your source code into the binaries needed for installation.

Build tools organize your source code, set compile flags, and manage dependencies.

They are most important in environments where there are many inter-connected projects, with

multiple developers contributing to each project.

In these environments it can be very difficult to keep track of what changes were made, in

what order, what dependencies exist, and what needs to be incorporated in the next build.

So automation is key to keeping everything running smoothly.

Build automation can automate a wide variety of tasks that developers do in their day-to-day

activities like:

Downloading dependencies.

Compiling source code into binary code.

Packaging that binary code.

Running tests and

deployment to production systems.

You can initiate a build from the command line or from an IDE.

There are two categories of Build Tools widely in use:

Build-automation utilities, which generate build artifacts like executables, by compiling

and linking source code.

Build-automation servers, which execute build-automation utilities on a scheduled or triggered basis.

Some examples of build tools are:

Webpack – a module bundler for JavaScript and

Babel – a JavaScript compiler.

Now that you have your app developed and tested, you’re ready to deploy.

But how does that happen?

The app needs to be simple and trouble free for the user to install, so a commonly used

technique is to collect all the necessary files and bundle them together into a package.

Packages are archive files that contain the app files, instructions for installation,

and any metadata that you choose.

They have their own metadata too, including the package description, package version,

and any dependencies, like other packages that need to be installed beforehand.

Once you have bundled your app into a package, you can use a package manager to distribute it.

Package managers take care of the tasks of finding, installing, maintaining or uninstalling

software packages at the user's request.

Package management systems:

Coordinate with file archivers to extract package archives

Verify checksums and digital certificates to ensure the integrity and authenticity of

the package

Locate, download, install, or update existing software from a software repository and

manage dependencies to ensure a package is installed with all packages it requires.

Some commonly used package managers for each of the major platforms are listed here:

On Linux - Debian Package Management System (DPKG) and Red Hat Package Manager (RPM)

On Windows - Chocolatey

On Android - Package Manager

On MacOS - Homebrew and MacPorts

Any libraries or utility code that is developed as part of the application

is managed with the cloud application package managers.

Here are some examples of package managers for popular languages:

for Node.js/Javascript - npm

for Java - Gradle and Maven

for Ruby - RubyGems

for Python - Pip and Conda

In this video you’ve learned about some of the tools that you will utilize will support

you in your career as a developer , including:

CI/CD, Build Tools, Packages and Package Managers

**The Importance of Back-End Development**

Welcome to the importance of back-end development.

A front-end developer creates websites and Cloud applications, using HTML, CSS and JavaScript

to create what the user sees and interacts with in the client software.

A back-end developer creates and manages all the resources that are needed to respond to

the requests that the user makes through the client.

The back-end developer’s tasks focus on enabling the server infrastructure, or back-end,

to process requests, supply data, and provide other services securely.

Front-end and back-end developers must work together very closely.

Each needs to understand the requirements of the solution and how their respective parts

will interact before the development process can begin.

Throughout the lifecycle of the website or Cloud app, front-end and back-end developers

collaborate to resolve issues and add functionality.

How does the work of a back-end developer affect you as you are browsing the internet

or using a Cloud app?

Think about it like this: when you’re shopping online, what happens to the data you enter?

Your login information, your product searches, your payment info?

The back-end processes all these things, and the back-end developers write and maintain

the parts of the application that process the inputs.

Let’s think about your experience as you explore an online shopping site and make a

purchase.

As you search for products, your search request is submitted to a web application which then

retrieves the data from a separate database and serves it back to the client for display.

To facilitate this, a back-end developer must understand the language that the web application

uses, how to query the database for the correct data, and how to bring the two together.

Even a simple task like navigating around the site can require the skills of a back-end

developer.

Many sites have restricted areas that are only available to users who have an account

and have logged in.

User account management, authentication and authorization can be the responsibility of

the back-end developer too.

Once you’ve decided what you want to purchase, you must add it to your cart and make a payment.

The purchase process requires you to submit sensitive information, such as your address

and credit card number, and the back-end developer must ensure that this data is securely handled

and stored.

Front-end client interactions, whether a request for data like an image, accepting input from

a user filling out a form, or securing sensitive information like a credit card number all

require different services from the back-end server.

Each request needs to interact with the back-end in a different way.

Back-end developers use APIs, routes, and endpoints to process incoming requests:

An API is code that works with data, usually using JSON or XML. APIs have set rules and

structure.

A route is a path to a website or page that the user interacts with.

Routes generally take user input and show results based on the input.

An endpoint may be an API or may simply be a path.

When a request from the front-end arrives at the back-end, it is routed to the correct

service.

If the backend has an end point defined for the request by using routing, the request

will be addressed and replied to.

If the end point is missing, the server returns a 404 error.

Back-end developers must create and maintain this server-side routing.

Along with backend APIs, routes effectively allow the front-end client to plug into the

correct socket at the back-end.

APIs provide a mechanism for Cloud apps, mobile apps, and other types of software to access

resources from the back-end.

To perform all this back-end development you will need to be familiar with at least one

back-end language, and its associated frameworks.

Among the most popular languages today is JavaScript, which was originally designed

to run in web browsers, adding extra interactivity and dynamic content to web pages.

JavaScript is also being used on the back-end, with new releases adding server-side functionality.

JavaScript has many frameworks, but two of the most well-known are Node.js and Express.

Python is another popular language; it’s very flexible and easy to learn. Python has

wide functionality – it can be used for everything from creating web pages to connecting

to a database, to performing data analysis.

Two well-known Python frameworks are Django and Flask.

Back-end developers often work with data and databases, so you will also benefit from learning

some SQL.

To help handle requests from databases, back-end developers can use object-relational mapping

tools (ORM) to connect to the database and retrieve the correct data.

Although an ORM can hide some of the complexity of querying databases, it’s useful to understand

the fundamentals of databases so that you can troubleshoot any issues that arise.

The day-to-day tasks of a back-end developer focus on the behind the scenes functionality

that keeps websites, cloud apps and mobile apps up and running.

Back-end development covers a wide range of technologies, from managing user accounts,

authentication and authorization to ensuring that sensitive data is stored and transferred

securely.

Back-end developers also work with databases, retrieving , processing and storing data as

required.

Life for a back-end developer is varied, challenging, and ever changing.

**What is Geolocation API?**

Most smartphones today have a GPS(Global Positioning System) receiver built-in. Using the GPS, they can tell your exact location. Needless to say GPS feature is an useful feature. Imagine the advantages if a web page can identify the current location without any sophisticated hardware.

HTML5 has many cool APIs. One of it is Geolocation. Using this API(Application Programming Interface) a web developer can find the location by writing just a few lines of code. Before the days of this API, finding the location was a difficult task.

If a web browser can find out it's current location, it would be of great utility in many scenarios. For example, when you order food online, the food delivery company expects you to type in your address. If your browser can identify your current location, you can automatically find it out and key in the current location.

**How Geolocation API identifies the current location.**

The Geolocation API uses variety of sources to find out your current location. Usually Geolocation API will try to determine the position using one of these methods.

**GPS (Global Positioning System)** If your device has any GPS capabilities, the current position is identified using the satellite signal.

**Mobile Network Location** If you are browsing the internet using a mobile phone or a wireless modem, your location is tracked using the location of the nearest cell phone towers.

**WiFi Positioning System** If you are using a WiFi, then the Wi-Fi positioning system is used.

**IP Address Location** If none of the above is a possibility, location is detected based on the nearest Public IP Address. The accuracy of IP based location may not be very accurate. It can be used to identify the city, region or country.

**What is the Drag and Drop API?**

Drag and drop is a popular feature of many Graphical User Interface (GUI) systems. Implementing the same functionality in a browser was an exceedingly difficult task. HTML5 Drag and Drop API enables web applications to use drag-and-drop features in browsers. This helps web developers to create rich user interfaces which can compete with the desktop user interfaces. The Drag and Drop API allows you to make your divs, spans, and paragraphs draggable. You can drop them onto any other container tags like divs.

**Use cases of the Drag and Drop API**

How do you add an item into a shopping cart on an e-commerce web site? You may have to select an item, key in the quantity and click on the 'add to cart' button. Imagine how cool would it be if you could drag an item on to your shopping cart.

**Introduction to HTML**

Hello and welcome to Introduction to HTML.

After watching this video, you will be able to explain

what Hyper Text Markup Language (HTML) and HTML5 is,

and list the objectives of HTML5.

The core HTML specification includes this background information:

HTML stands for HyperText Markup Language, which can be thought of as “the language

of the Internet”.

It is a markup language that was originally designed for sharing scientific documents.

Adaptations to HTML over the years have made it suitable to describe a number of other types

of documents that can be displayed as web pages on the Internet.

HTML Elements are the building blocks of an HTML page. The pieces of content that form

the page are assigned labels such as “paragraph,” “list,” and “table”.

They are represented by tags.

Browsers do not display the tags but use them to render the content.

HTML has been in continuous evolution since it was introduced to the Internet in the early

1990s by CERN:

The European Organization for Nuclear Research and the IETF: Internet Engineering Task Force.

The World Wide Web Consortium (W3C) made a number of recommendations to the HTML standards

over the years.

New features were continually added, and specifications changed over time.

Initially, the Web Hypertext Application Technology Working Group (WHATWG) worked on recommendations

independently.

Then, in 2007, W3C formed a working group chartered to work with the WHATWG on the development

of the HTML5 specification.

The W3C published the specification under the W3C copyright, while a version with a

less restrictive license was kept on the WHATWG site.

Since 2007, both the W3C and WHATWG groups worked together on the development of the

HTML5 standard.

The HTML5 specification, as drafted by W3C, includes the following objectives:

The HTML5 specification defines a single language called HTML5 that can be written in HTML or

XML syntax.

The HTML5 specification defines a processing model that can interoperate with earlier HTML

implementations.

HTML5 improves the markup for documents.

HTML5 includes markup and APIs for idioms, such as web storage, video, and audio content.

When speaking about creating web pages, developers may use the terms HTML and HTML5 interchangeably.

In general, the term HTML implies HTML5.

Use the full term HTML5 if you need to differentiate between versions, such as when discussing

features new to HTML5.

In this course, HTML refers to HTML5, unless specified otherwise.

In this video, you learned:

HTML stands for HyperText Markup Language, a markup language that enables documents to

be displayed as web pages on the Internet.

Tags represent the elements of an HMTL page like paragraphs, lists, and tables.

HTML5, the latest version supports:

Pages written in HTML or XML syntax,

Interoperation with earlier HTML versions,

Markup and APIs for web storage, video, and audio content

**HTML Features**

Hello and welcome to HTML Features.

After watching this video, you will be able to:

List some of the features of HTML5,

Describe the Document Object Model (DOM) tree,

Explain when you would use XML and when you would use HTML.

So, what are some HTML5 features?

It provides the means to categorize web pages into different sections, and includes tools

for effective data management, drawing, video, and audio.

Facilitates the development of cross-browser applications for the web and portable devices.

Allows greater flexibility, permitting the development of exciting and interactive websites.

Helps to create a more engaging user experience. Pages that are designed by using HTML5 can

provide an experience similar to desktop applications.

Allows for enhanced, multiple-platform development by combining the capability of an application

programming interface (API).

By using HTML5, developers can create a modern application experience that is uniform

across platforms and devices.

Here is an example of a document that conforms to the syntax of HTML5.

Documents that have the text/html media type are parsed using HTML rules.

If a document is transmitted with an HTML content type, such as text/html, it is processed

as an HTML document by web browsers.

This example uses the following elements:

The &lt; !DOCTYPE &gt; is a declaration tag that represents the document type.

The &lt; !DOCTYPE &gt; declaration is not an HTML tag; it is an instruction to the web browser

about what version of HTML the page is written in.

Although this declaration is not required, it should be the first line of the HTML code

if the developer decides to include it.

The &lt; html &lt; tag is the root element of this tree.

It contains all of the other HTML elements, except the &lt;!doctype&gt; tag.

This example, contains two elements: &lt;head &lt; and <b>.</b>

The &lt;head &lt; tag contains a &lt;title &lt; tag, which contains the text “Sample page”

in the example.

The &lt;head &lt; element can contain the following tags:

title ( &lt;title &lt; ),

scripts (<s>),</s>

style (<s>),</s>

style sheet links ( &lt;link &lt; ),

meta information (&lt;meta &lt; ),

browser support information and other initialization functions (<b>).</b>

Finally, the <b> tag contains all content that is displayed on the webpage.</b>

HTML user agents, commonly known as browsers, parse the markup, turning it into a DOM (Document

Object Model) tree.

A DOM tree is an in-memory representation of a document.

DOM trees contain several kinds of nodes, which include a DOCTYPE node, elements such

as headers and paragraphs, text nodes, and comment nodes.

Here is an example of a document that conforms to the XML syntax of HTML5.

Extensible Markup Language (XML) documents look similar to HTML documents, except they

have an XML tag on the first line.

In addition, with XML documents, the Content-type must be specified as an XML media type such

as application/xml.

When a document is transmitted with an XML content type, it is treated as an XML document

by a web browser and an XML processor parses the document.

How do you decide whether to use HTML or XHTML for developing web pages?

Which of the two options you choose is related to how you intend to use the application

For instance, are you using XSLT (Extensible Style Sheet Language Transformations) to create

other documents and resources from your web pages?

People are equally divided about whether to use XHTML or HTML.

Both HTML and XHTML use the same semantic (tags). However, XHTML tags all need to be

in lowercase, while the case used does not matter in HTML.

In addition, XHTML must be well-formed.

Every element must have an end tag. All attributes must have a value and double or single quotation

marks must surround all attribute values.

If an XML parser encounters a situation where the syntax is not well-formed, it stops processing.

In HTML, different case, unmatched quotation marks, and non-terminated and uncontained

elements are allowed and commonplace.

In this regard, HTML syntax is less rigorous than XHTML syntax.

In this video, you learned that HTML5

Includes features for categorizing sections of web pages, and managing data, video and

audio tools.

Enables you to develop a single, cross-browser application for the web and portable devices,

Helps you to create a more engaging user experience.

The DOM tree is a representation of a document which is held in memory.

It contains nodes which define the type and structure of the document.

**HTML Management and Support**

Hello, and welcome to HTML Management and Support.

After watching this video, you will be able to:

List some HTML document API properties and methods,

Describe how scripting is enabled in browsers,

Describe browser support for HTML5 features.

Let’s examine some of the key themes of the HTML5 specification.

The syntax it defines is compatible with HTML4 and XHTML1 documents.

HTML5 is defined in a way that it is compatible with earlier versions in the way browsers

handle deployed content.

It separates conformance requirements for user agents and authors. In other words, how

browsers treat HTML5 code and how page authors treat HTML5 code is different.

For user agents, HTML5 is defined in a way that supports elements and attributes of the

earlier specifications.

For authors, several elements and attributes have been removed in order to simplify

the language.

HTML5 includes some elements and application programming interfaces that help in the creation

of web applications.

Examples include video and audio elements, and an API that supports the creation of offline

applications.

What do you think makes HTML suitable for creating web applications? There are a number

of possible reasons:

The range of devices that all have built-in browsers that support HTML5 features.

The number of APIs that enhance the user experience, such as the advanced animation, drawings,

audio, and video elements.

The efficient use of HTML and CSS, while reducing the number of images, can lead to faster load

times for rendering web pages.

Search engine optimizations can use keywords that are taken from HTML attributes to improve

the visibility of a website when search results are displayed.

The meta tags shown are used to provide information to search engines.

Here are some examples of elements in HTML5.

New structural elements are part of the HTML5 specification. These markup elements help

you to logically define the page structure.

New APIs such as canvas, audio, and video help with graphics and embedded content.

New attributes were added to the input element attributes.

Examples include the email, datetime, and number types.

The browser automatically validates the input field according to the input attribute, without

the need for custom validation scripts.

By using web storage APIs, you can store data in the browser.

The web workers feature can be used for non-interactive processing. Web workers provide a way to run

processing-intensive tasks without blocking the user interactions to the current page.

In this video, you learned about

HTML5 Themes, which provide:

Syntax compatible with HTML4 and XHTML1,

New and refined APIs for video and audio elements, offline web apps, and drag and drop.

About HTML5 Web Applications, which have:

Improved search indexing with meta tags,

Better page load times, and

enhanced user experience.

And about HTML5 Elements, which enable you to define and manage web page structure

and content.

**HTML Scripting**

Hello and welcome to HTML Scripting.

After watching this video, you will be able to describe how scripting is enabled in browsers.

Scripting is enabled for a browser context, when the conditions shown on the slide are

true.

Scripting is used to provide a more interactive user experience when browsing websites.

Since scripting can be turned off, the recommendation is to use scripting but not to rely on it.

HTML5 also defines a text or html-sandboxed media type for documents.

This attribute can be used when hosting untrusted content.

The sandboxed browsing context can be set at the page level or specified as an attribute

on any tag that contains an embedded object.

If you are running a page with an embedded object without the sandbox attribute, you

are implicitly giving a third-party vendor permission to run scripts with the same permissions

that you have for the page.

Granting the embedded object the same permissions as the rest of your page is one way that you

can inadvertently allow advertisements to occur in your application.

To prevent granting implicit permission to embedded objects, use the sandbox attribute

on any tag that contains an embedded object.

Each HTML document that is loaded into a browser page becomes a Document object.

The Document object provides access to all HTML elements in a page and can be accessed

from within a script.

The DOM tree accessors are HTML document APIs that provide access to all the HTML elements

on a page.

The property to access is prefixed by the word document.

For example, document.head returns the first head element that is a child of the html element,

if there is one, otherwise null.

The function document.images returns an HTMLCollection of the image elements in the document.

The function document.scripts returns an HTMLCollection of the script elements in the document.

Here are some common HTML DOM tree methods:

The document.getElementBy Id (‘id’) function has one required parameter which is the id

of the element you want to access.

The id is specified as an attribute on one of the elements that is coded elsewhere on

the page.

The document.getElementsByTagName(‘tag’) function has one required parameter which

is the tag name of the element you want to access.

This function returns a node list of all the elements with the tag that matches the required

parameter.

This is a fragment of HTML with JavaScript code that uses the document API.

When submit is clicked inside the page that is running in a web browser, the textChecker

function begins.

The document.getElementById(‘textField1’) retrieves the contents of what was typed into

the input field that has the id of textField1 and places this content into a variable.

The JavaScript logic then tests whether any data was typed into the field, and an alert

dialog is displayed with the result.

A similar function will be used later in Exercise 1.

Here is the output from running the code on the previous page that uses the document API.

The user types “Test” in the input field and clicks Submit. The result is seen in an

alert dialog box on the same page.

In this video, you learned:

Scripting is used to provide a more interactive user experience when browsing websites.

Since scripting can be turned off, the recommendation is to use scripting but not to rely on it.

Scripting is enabled when certain browser conditions are met.

HTML5 sandboxes enable you to manage iframe mash-ups; web pages that pull content from

more than one site.

HTML documents loaded into a browser page become Document objects, which you can access

through scripts.

**HTML5 Browser Support**

Hello and welcome to HTML5 Browser Support.

After watching this video, you will be able to:

Describe browser support for HTML5 features.

Not all browsers fully support all the features that are described in the HTML5 and CSS3 specifications.

For example, older versions of browsers do not support certain features.

The developers of different browsers are continually adding support for HTML5 features to their

new browser versions.

For the most comprehensive support of HTML5 features, consider using Google Chrome.

Support tables show which features are supported by which browsers and can be a useful tool

when developing web pages.

For example, caniuse.com has great support tables for HTML5, CSS3, and more.

Here are the results of displaying the &lt; input type= ‘date’ &gt;element in different browsers.

The Google Chrome browser displays a calendar to enter the data.

The Microsoft Edge browser displays spinning boxes to enter the data.

Notice the difference in date formats (yyyy-mm-dd vs mm/dd/yyyy).

The Firefox browser displays the date field as a text field.

This could raise several issues regarding input format and validation.

This example shows a support table for the &lt; input type= ‘date’ &gt; element.

The numbers represent the browser version. The color determines the level of support.

Green: Full support

Olive: Partial support

Red: No support

Edge, Chrome, and Opera browsers are green, meaning they support the &lt; input type= ‘date’ &gt;

element.

Internet Explorer and Safari browsers are red, meaning they do not support the &lt;input

type =‘date’&gt; element.

Support levels change as developers of browsers add new features, so check back frequently

to see if the features you require are supported.

You can also use JavaScript to check whether a certain HTML5 element is supported by a

browser.

First, create a DOM element by typing document.createElement() and include the element type as a parameter

argument. As a result, the DOM object gets created.

If the browser does not support that element, the DOM object that gets created has a set

of common properties, but nothing specific for that element.

You then test for a known property or method on the DOM object that you created.

If that property or method does not exist, then the browser does not yet fully support

the DOM element (or the HTML5 tag) that has been created.

Browsers that do not support the element can revert to a fallback or default behavior such

as displaying the field as a regular text field.

Here is an example of using JavaScript to check whether a certain HTML5 element is supported

by a browser.

This code is used to test whether the browser supports the element &lt; input type= ‘date’ &gt;

In the example, you dynamically create the DOM object for the input element,

and then you set the type attribute to the value of "date“.

If the browser does not support the input type =date, the field is displayed in the browser

as a regular text field.

In this video, you learned:

Not all browsers fully support all the features that are described in the HTML5 and CSS3 specifications.

CanIUse.com has good support tables that show what features are supported by which

browsers.

You can use JavaScript to check whether a certain HTML5 element is supported by a browser.

**Hands-On Lab: HTML - Creating a Simple Web Page**

**Add a heading to the page**

To add a heading, we use the <h1> tag.

**Add a Paragraph**

The tag <p> is used to define a paragraph.

**Add an Image**

<img> tag is used to add an image.

To add an image you need to know the image file name and mention it in the 'src' attribute.

The 'src' attribute specifies an external resource that you want to link, such as the URL of an image.

You can optionally specify how many pixels the width and height of the image should be.

**Create a List**

To create a list of items, we can use the <ol> tag for numbered lists, and <ul> for bulleted lists.

**Add a Table**

A table is created using the <table> tag.

Within the table, each row of data is represented using the <tr> tag.

The column or row headings can be specified by <th> element.

Finally, each data element within the table cells is specified using the <td> tag.

**Add Links to other pages**

Web pages connect to other web pages and files using hyperlinks.

A hyperlink is created using the anchor tag <a>.

Once you click on the link the 'href' attribute tells the browser to which web page you should be taken to.

<!DOCTYPE html>

<html>

    <head>

        <title>Thomas J. Watson Sr - Fan Page </title>

    </head>

    <body>

        <h1>Thomas J. Watson </h1>

        <img src="https://upload.wikimedia.org/wikipedia/commons/7/7e/Thomas\_J\_Watson\_Sr.jpg" width="300" height="300">

        <p> Thomas J. Watson, Sr. is the American industrialist, who

            built the International Business Machines Corporation (<a href="https://www.ibm.com">IBM</a>)

            into the largest manufacturer of electric typewriters and

            data-processing equipment in the world.

        </p>

        <ul>

            <li>In 1939, he received an honorary degree in Doctor of Commercial Science from Oglethorpe University. </li>

            <li>In 1940s, Watson was on the national executive board of the Boy Scouts of America. </l>

            <li>Watson served as a trustee of Columbia University from June 6, 1933, until his death. </li>

            <li>He was posthumously inducted into the Junior Achievement U.S. Business Hall of Fame in 1990.</li>

        </ul>

        <p><strong>IBM Growth Over Years: </strong></p>

    <table>

        <tr>

            <th>Year </th>

            <th>No. of Employees </th>

            <th>Gross Income (In m$) </th>

        </tr>

        <tr>

            <td>1925</td>

            <td>3698</td>

            <td>13</td>

        </tr>

        <tr>

            <td>1930</td>

            <td>6346</td>

            <td>19 </td>

        </tr>

        <tr>

            <td>1935</td>

            <td>8651</td>

            <td>21</td>

        </tr>

    </table>

    </body>

</html>

**CSS: Styling HTML**

Hello and welcome to CSS: Styling HTML.

After watching this video, you will be able to:

Explain what a Cascade Style Sheet (CSS) is,

Describe how to use CSS to format HTML elements,

Describe how to use CSS in HTML documents to format web pages.

By using a combination of HTML5, JavaScript, and cascading style sheets, developers can

create web pages that can be displayed on desktop computers and mobile devices.

These technologies have powerful features that can be used to show rich interactive

applications without requiring plug-in technologies, like the Flash media player.

HTML documents might be rendered to a browser window, or through a speech synthesizer to

provide assistive technologies for vision impaired people.

Being able to meet accessibility requirements is one advantage of using HTML markup over

browser plug-ins for delivering interactive content.

CSS is the design that is layered over the top of an HTML web page.

CSS is a style sheet language that describes how HTML elements are displayed.

What makes the style sheet “cascading” is that developers can apply CSS to create

a uniform look throughout each element of each page of the website.

Child and descendant elements often inherit styles that are defined for parent elements,

however, exceptions to this rule do exist.

For websites, it is an important concept to separate the data from the design.

The data is sent to the browser by using HTML, and the design is applied to that data by

using a CSS.

This separation allows people to render a web page without your design if they have

special accessibility needs.

It allows machines such as search engines to index a website without the design interfering.

You can code CSS as a style attribute in an HTML tag, a head section of a document, or

an external document.

The preference is to code CSS in external documents referred to as style sheets.

You can use CSS to control a document’s appearance and specify style rules for the

following web page elements:

Fonts, Text, Colors, Backgrounds, Sizes, Borders, Spacing, Positioning, Visual effects, Tables,

and Lists.

CSS style element are structured like this.

In this example, the html-tag-name can be one of the following elements:

Any of the tags you find in HTML code (for example, &lt;a&gt;, &lt;div&gt;, &lt;li&gt;, or &lt;label&gt;, etc.)

An id reference that is displayed with a preceding hash symbol (#) in CSS code.

A class reference that is displayed with a preceding dot/period (.) symbol in CSS

code.

When making a site design, begin by establishing the base style. Establish a base style by

styling the <b> tag, as shown here.</b>

This example makes the following styles:

Sets the background color: off-white (background-color),

Ensures that the font color is black (color),

Ensures that all content edges match the edge of the browser window frame (margin and padding),

Horizontally aligns the textual content to the left using (text-align),

Sets the font size to the browser’s default (font-size) and uses a sans-serif (a font

without the little flicks around the edges) as font family using (font-family).

These settings are simple. Generally, follow these guidelines:

When a color is specified, use Red-Green-Blue (RGB) hexadecimal light values.

When a size is specified, use pixels (indicated by a px after the number) and em, which

is indicated by em after the number (that is, the size of the font multiplied by the

specified number) or a percentage, which is indicated by a % after the number.

Text can be aligned left, right, or center.

Floats can also be left or right.

Vertical alignments must be top, middle, or bottom.

Fonts can be any specific font or font family (serif, sans-serif, or monospace) or even

a downloadable font.

One of the most important decisions you must make when you are determining the design of

your website is whether to use a fluid or a fixed layout.

A fluid layout is a layout in which the height and width of elements is flexible and

can expand or contract based on the browser window, the operating system, and other user

preferences. You specify these elements mostly by using percentages and ems.

A fixed layout is a layout where you specify the height and width of elements, and those

values remain the same regardless of which operating system or browser you use to access

the website. You specify these elements mostly by using pixels.

When determining the layout, consider also the pros and cons for fluid and fixed layouts.

The type of layout you choose depends on the type and amount of content and the target

audience of the website.

To apply a CSS, you must tell the browser where to look for it. This step is the only

true point where HTML references a CSS. You can make the reference in one of two ways, the <s> tag</s>

or the &lt;link&gt; tag

Style tag

It “dirties” the page with a non-HTML code.

You apply this style to only one page.

If you copy and paste this style on each page, this approach increases the load time of each

page, which causes the user to wait longer.

Link tag

You can link to it from other pages, which ensures a clean HTML and a smaller page size.

To use this method, the &lt;link&gt; tag must be added to the &lt;head&gt; tag section.

In this video, you learned:

CSS creates a uniform look throughout each element of each page of the website.

CSS design is separate from the data.

Design can be removed for special accessibility renders.

Search engines can index the website without the design interfering.

CSS is usually coded in external style sheets.

Create base styles for your website first.

Choose a layout type.

Fluid: the height and width of elements is flexible and can expand or contract.

Fixed: the height and width of elements remains the same.

**2. Specify a font family and font size for text**

CSS helps define style, layout, colors, and fonts for your pages. In this exercise, you will set the font family. We'll also use CSS to change the font characteristics for some elements.

We'll use the **font-family** and **font-size** properties to change the appearance of the heading.

**Note**: In this lab, you will apply styles using **inline CSS format**. You will add the **style** attribute to the element to change the style.

**Hands-On Lab: CSS Basics - Styling Your Web Page**

**Change the heading font**

Modify the <h1> tag so that it looks like this:

<h1 style="font-family: Cursive">Solar System</h1>

Here, Cursive is a generic font family. You could also specify a font family such as Arial or a group of font family names separated by a comma, such as font-family: Arial, serif.

**Change the font size**

Add the **font-size** property to the <h1> style so that your code looks like this:

<h1 style="font-family: Cursive; font-size: 70px"> Solar System </h1>

**Change the color of an element**

You will now set the background color of the page and update the color of an HTML element. To change the background color, you use the CSS **background-color** property. To change an element's color, you use the **color** property.

**Change the background color**

To change the background color of the body of the web page, update the <body> tag as in the following code:

<body style="background-color:wheat">

**Change the font color**

You specify a color for an element using a predefined color name, or by using RGB, HEX, HSL, RGBA, or HSLA values.

The RGB system enables you to specify a color as a combination of the primary colors red, green, and blue. RGB colors range from *rgb(0,0,0)*, which is black, to *rgb(255,255,255)* which is white.

Change the color of your heading using the RGB system, as follows:

<h1 style="font-family: Cursive; font-size:70px; color:rgb(139,0,0)">Solar System</h1>

**4. Create borders**

Tables created in HTML do not display a border by default. Using CSS, we can apply borders to tables, <div> elements, and other tags.

**Add a table border**

We'll add a border to the table in your web page. The border will be solid, black, and two pixels wide.

Update the <table> tag with the following CSS code:

<table style="border: 2px solid black">

**Add a border to table headings and table cells**

CSS gives you the flexibility to add a style to multiple instances of the same HTML tag. For example, instead of specifying the style for each individual <th> and <td> tag, we can add a style to all of those tags using **internal CSS**. We'll add a <style> tag within the <head> section of our page to specify how we want all <th> and <td> tags to display.

To apply an internal CSS style to your page, replace the <head> section with the code below:

<head>

<title>Solar System</title>

<style>

table,th,td {

border: 2px solid black;

}

</style>

</head>

**Highlight the table header**

Finally, let's make the table header stand out by specifying a background color for the entire table row. By adding the style to the row, you do not need to update each individual <th> tag within the row.

Add a style to the first <tr> tag in the table, as follows:

<tr style="background-color:yellow">

<th>No. </th>

<th>Planet Name </th>

<th>Distance from Sun </th>

</tr>

<!DOCTYPE html>

        <html>

            <head>

                <title>Solar System</title>

                <style>

                    table,th,td {

                        border: 2px solid black;

                    }

                </style>

            </head>

           <body style="background-color:wheat">

                <h1 style="font-family: Cursive; font-size:70px; color:rgb(139,0,0)">Solar System</h1>

                <img src="https://upload.wikimedia.org/wikipedia/commons/c/cb/Planets2013.svg" alt="Solar System image" height= "250px" width="700px">

                <p>The solar system consists of the sun and everything that orbits the sun. This includes the eight planets and their moons, the dwarf planets, asteroids, comets and other small objects.

                    <br><br>All the planets and dwarf planets, the rocky asteroids, and the icy bodies in the Kuiper belt move around the Sun in elliptical orbits in the same direction that the Sun rotates. This motion is termed prograde, or direct, motion.

                </p>

                <p><strong><u> Planets in the solar system: </u></strong> </p>

                <table style="border: 2px solid black">

                    <tr style="background-color:yellow">

                        <th>No. </th>

                        <th>Planet Name </th>

                        <th>Distance from Sun </th>

                    </tr>

                    <tr>

                        <td>1 </td> <td>Mercury</td> <td>57.91 m km</td>

                    </tr>

                    <tr>

                        <td>2 </td> <td>Venus</td> <td>108.2 m km</td>

                    </tr>

                    <tr>

                        <td>3 </td> <td>Earth</td> <td>149.6 m km</td>

                    </tr>

                    <tr>

                        <td>4 </td> <td>Mars</td> <td>227.9 m km</td>

                    </tr>

                    <tr>

                        <td>5 </td> <td>Jupiter</td> <td>778.5 m km</td>

                    </tr>

                    <tr>

                        <td>6 </td> <td>Saturn</td> <td>1.434 b km</td>

                    </tr>

                    <tr>

                        <td>7 </td> <td>Uranus</td> <td>2.871 b km</td>

                    </tr>

                    <tr>

                        <td>8</td> <td>Neptune</td> <td>4.495 billion km</td>

                    </tr>

               </table>

            </body>

        </html>

**HTML5 Tags and Structural Elements**

Hello. Welcome to HTML5 Tags and Structural Elements.

After watching this video, you will be able to:

List specific elements to HTML5 and

Describe HTML5 structural elements:

section, article, header, footer, figure, figcaption.

There are many HTML5 specific elements. The table shows some of them.

Notice that the HTML5 element tags have intuitive names, such as audio that is used to embed

sound content, and canvas that is used for graphical content.

Not all of these element tags are covered in this video, so take a moment to read through

the table.

This table shows more HTML5 elements.

The last tag in the table is the comments tag.

Comments in HTML can assist the developer to explain what the code does.

The browsers do not display these comments, unless the viewer views the source.

In a division-based layout, each part of a page that you want to format differently is

separated into its own division.

Think of the natural divisions into which you can divide the page that is shown here.

These divisions might include the page title, a banner with selectable tabs, an area for

the image, and further blocks for text and thumbnail images.

The &lt;div&gt; tag was not introduced in HTML5, but is a good starting point for laying out

a page or document.

Then, you can use other HTML5-specific structural elements inside the &lt;div&gt; tag.

Most HTML elements have a semantic meaning. In other words, the element describes either

the function or the type of data that is contained within.

For example, a &lt;p&gt; element is a paragraph that usually contains some text.

In contrast, the &lt;div&gt; tag has no innate semantic meaning besides the logical grouping of the

content.

You can use these dedicated elements to mark up your website, rather than using the generic

&lt;div&gt; or <s> elements.</s>

An &lt;article&gt; defines a block of code that can be distinguished from the rest of the

page.

A <s> defines a logical separation in the document such as the numbered chapters</s>

of an online manual.

A &lt;header&gt; element is used to group the headers for a page.

The &lt;footer&gt; defines the area at the bottom of the page.

This code example shows some of the HTML5 structural element.

Here, the &lt;article&gt; element is used to create an online news report.

The &lt;article&gt; includes a header.

Within the &lt;article&gt;, the page is marked up into sections that have their own headers.

The page also includes a &lt;footer&gt; element.

The &lt;aside&gt; tag is used to provide additional information that is related to the main discussion.

The aside tag lets you extract and display further content or go to additional resources

without detracting from the main discussion.

The &lt;figure&gt; tag defines a self-contained element that is referred to from the main

content.

The &lt;figure&gt; element can be used to embed graphics, images, or code sections.

The &lt;figurecaption&gt; tag defines the caption for the contents of the &lt;figure&gt; element.

Websites typically have sections that are dedicated to navigational links that go to

these sites or for moving between pages of a single application.

In HTML5 syntax, these navigational links are placed inside a &lt;nav&gt; tag.

The &lt;nav&gt; tag is a convenience tag for grouping navigational links in a web page.

Websites typically have sections that are dedicated to navigational links that go to

these sites or for moving between pages of a single application.

In HTML5 syntax, these navigational links are placed inside a &lt;nav&gt; tag.

The &lt;nav&gt; tag is a convenience tag for grouping navigational links in a web page.

In this video, you learned:

Tags provide control within an HTML5 document.

Some tags provide structural elements:

The &lt;div&gt; tag separates areas in a document into divisions, enabling you to apply different

styles to different parts of a document.

Dedicated elements like &lt;article&gt;, <s>, &lt;header&gt;, and &lt;footer&gt; are more specific than</s>

the generic &lt;div&gt; element.

The &lt;aside&gt;, &lt;figure&gt; and &lt;figcaption&gt; tags enable you to group content.

And the &lt;nav&gt; tags enable you to group navigational links.

Table

Description automatically generated

Table

Description automatically generated

**HTML5 Input Element: Attributes for the Input Tag**

Hello. Welcome to HTML5 Input Element: Attributes for the Input Tag.

After watching this video, you will be able to:

Describe the attributes of the HTML5 input element input type attributes tel, email,

date time, number, range, and color.

The &lt; input type= "color" / &gt; attribute allows the user to select a color.

The dialog varies depending on the browser.

Some browsers don’t support this input type.

In non-supporting browsers, this input type is displayed as a regular text input field

into which the user might type a valid color name or color code.

The &lt; input type="date" &gt; attribute is a date control (year, month, day) with no time zone.

The example shows the input type="date" field as it is seen in Google Chrome.

The input dialog varies from browser to browser.

The datetime-local attribute provides input for a date and time (year, month, day, hour,

minute, AM/PM) with no time zone.

The datetime-local input field is displayed as a drop-down calendar in the Google Chrome

browser. The time can be typed or entered using the spinner control.

The &lt; input type="email" &gt; attribute is displayed as a regular text input field.

It provides feedback when the input does not follow the email format.

The &lt; input type="number" &gt; takes a numeric value as input.

You can optionally specify the minimum, maximum values, step size, etc.

The example shows the number selector in the Chrome browser. Only the numbers between 5

and the 10 are available for selection.

The &lt; input type="range" &gt; takes a numeric range as input.

The example shows the number selector in the Google Chrome browser.

Only the numbers in the range of the minimum and the maximum are available for selection.

The range attribute displays a slider with the range of values between the minimum and

maximum.

Only the slider itself is shown. Additional JavaScript code is needed in order to display

the value of the slider.

The differences between &lt; input type="search"\ &gt; and &lt; input type="text" / &gt; are mostly in

style.

WebKit-based browsers return a history of recently searched text strings.

The search input field on the Safari browser has rounded corners.

The input type="tel" pattern="[parameters]" attribute expects a telephone number as input.

On its own, the &lt; input type="tel" &gt; provides nothing more than a text entry field in the

browsers.

It does not enforce numeric only input since many telephone numbers include other characters,

such as the plus sign and hyphens.

You need to supply your own pattern matcher if you want the browser to validate the telephone

number.

The URL attribute is used to validate that the user typed in a properly formatted URL

or web address.

The &lt; input list="some\_list" &gt; uses the &lt;datalist&gt; feature.

Not to be confused with the &lt; select &gt; element. The &lt;datalist&gt; options are only suggestions.

Useful for auto-complete functionality.

You can fill the list by nesting &lt;option&gt; elements inside the &lt;datalist&gt; tag.

These options are the types of fruits listed in the drop-down list.

Placeholder text is used to provide hints of what the input text format looks like.

The placeholder fills the input text field with the example values in a lighter shade

of text.

The form does not submit the placeholder text value if the input text is not overwritten.

The required attribute implies that some text must be typed.

The requirement to type some input applies even if the field contains placeholder text.

What happens if browser-based validation is not supported for these input attributes?

There are several options to performing validation in browsers that do not support all HTML5

input attributes.

You can use JavaScript and JQuery libraries.

You can assume that more browsers will support these features over time, and leave all final

validation to server-side processing.

You can code client-side validation that is attached to the form submit event handler

to validate all the fields on the form when the form is submitted.

In this video, you learned:

Common attributes of the &lt; input &gt; tag and

how to implement validation fallback.

**Special HTML Elements**

In this section, we will explore a tag found within the <form> tag called **<fieldset>** tag and **<legend>** tag.

**HTML fieldset tag**

* The HTML **<fieldset>** tag is found within the **<form>** tag and is used to group related elements in an HTML form inside a box.
* There is no restriction to the kind of elements that can be inside a fieldset. But they are mostly used to group related input type of elements, as shown in the example below.
* The elements can be grouped in a fieldset. This element can be specially useful in large forms, where readability and ease of access can be improved with segmentation. Browsers will most likely render a frame around the grouped controls.

**Syntax**

<fieldset> Contents... </fieldset>

**Attribute**

1. **disabled**: It specifies that the elements belonging to the fieldset should be disabled.
2. **form**: It specifies id of the form the fieldset is to be considered a part of.
3. **name**: It specifies the name for the fieldset.

**Example**

<!DOCTYPE html>

<html>

<body>

<h1>The fieldset element</h1>

<form>

<fieldset form="user\_regn" name="user\_details">

<label for="fname">First name:</label>

<input type="text" id="fname" name="fname"><br><br>

<label for="lname">Last name:</label>

<input type="text" id="lname" name="lname"><br><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email"><br><br>

<label for="birthday">Birthday:</label>

<input type="date" id="birthday" name="birthday"><br><br>

</fieldset>

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

</form>

</body>

</html>

**Output**

Graphical user interface, application

Description automatically generated

**Note: You can edit the form as per your convenient**

**HTML legend tag**

A **fieldset** can additionally have a title or name, that can be provided by **legend**. The **<legend>** tag is used with the **<fieldset>** element as a first child to define the caption for the grouped related fields This tag is also commonly referred to as the **<fieldset>** element. By using **<legend>** tag with **<fieldset>** elements, it is easy to understand the purpose of grouped form elements.

**Example**

To understand the **<legend>** tag, let's add this tag to the above example and see what will be the output:

<!DOCTYPE html>

<html>

<body>

<h1>The fieldset element</h1>

<form>

<fieldset form="user\_regn" name="user\_details"> <legend>Personal Details:</legend>

<label for="fname">First name:</label>

<input type="text" id="fname" name="fname"><br><br>

<label for="lname">Last name:</label>

<input type="text" id="lname" name="lname"><br><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email"><br><br>

<label for="birthday">Birthday:</label>

<input type="date" id="birthday" name="birthday"><br><br>

</fieldset>

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

</form>

</body>

</html>

**Output**

Graphical user interface, application

Description automatically generated

**1. Create a form**

You will use the <form> tag to create an HTML form. The action attribute of the form specifies the URL that processes the data when the form is submitted.

**Note**: The code at the destination URL is written by the back-end system developer. As a front-end developer, your focus is on creating the HTML form.

The following code creates a basic form within an HTML document, to which you will add controls. Copy the code into your HTML file and save it.

<!DOCTYPE html>

<html>

<head>

<title> Registration Form </title>

</head>

<body>

<h2>Registration Form</h2>

<form action="/register">

</form>

</body>

</html>

**2. Add a label and a text input field**

The <input> tag is used to create different types of controls that enable the user to input values. The default input type is text.

The <label> tag defines the label text for other HTML form elements. The for attribute of the <label> tag should be the same as the ID of the element it binds to.

Let's add a label and a text input field for the first name of the user.

<label for="firstname">First name :</label>

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

The id attribute enables the front-end developer to access this input control. The name attribute enables the back-end system developer to access the value that's entered in this input field.

We need a similar field for the user's last name:

<label for="lastname">Last name :</label>

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

After you add the two input text fields, your code should look like this:

<!DOCTYPE html>

<html>

<head>

<title> Registration Form </title>

</head>

<body>

<h2>Registration Form</h2>

<form action="/register">

<label for="firstname">First name :</label>

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

<br><br>

<label for="lastname">Last name :</label>

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

</form>

</body>

</html>

Note that the <br> tag creates a line break. Without the <br> tag, all our form controls would display in a single line in our HTML page.

**3. Add a control to input an email ID**

To enable users to enter their e-mail address, you will use the <input> tag with the email type. Add the following code to your form after the **Last name** text field:

<label for="email">Email :</label>

<input type="email" id="email" name="email">

Your page should now like this:

<!DOCTYPE html>

<html>

<head>

<title> Registration Form </title>

</head>

<body>

<h2>Registration Form</h2>

<form action="/register">

<label for="firstname">First name :</label>

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

<br><br>

<label for="lastname">Last name :</label>

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

<br><br>

<label for="email">Email :</label>

<input type="email" id="email" name="email">

</form>

</body>

</html>

**Add a fieldset**

Now that the form is complete. We can use the **fieldset** tag to group the related elements of the form. The following code will group the elements in the form in a single box.

<!DOCTYPE html>

<html>

<head>

<title> Registration Form </title>

</head>

<body>

<h2>Registration Form</h2>

<form action="/register">

<fieldset>

<label for="firstname">First name :</label>

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

<br><br>

<label for="lastname">Last name :</label>

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

<br><br>

<label for="email">Email :</label>

<input type="email" id="email" name="email">

<br><br>

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

</fieldset>

</form>

</body>

</html>

**Note: the fieldset is basically used in large form where you have to group the elements differently.**

**Add a legend tag**

What if there are multiple elements inside a form that needed to be grouped separately. How are we going to identify them?

By using the **legend** tag, we can add a title to the grouped element to recognize them.

The following code will group the elements and segregate them with the help of title.

<!DOCTYPE html>

<html>

<head>

<title> Registration Form </title>

</head>

<body>

<h2>Registration Form</h2>

<form action="/register">

<fieldset>

<legend>Personal Information</legend>

<label for="firstname">First name :</label>

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

<br/><br/>

<label for="lastname">Last name :</label>

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

<br/><br/>

<label for="email">Email :</label>

<input type="email" id="email" name="email">

</fieldset>

<br/><br/>

<fieldset>

<legend>Education Qualification</legend>

<label for="hdegree">Highest Degree :</label>

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

<br/><br/>

<label for="Collegename">College name :</label>

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

<br/><br/>

<label for="University">University :</label>

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

<br/><br/>

</fieldset>

<br/><br/>

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

</form>

</body>

</html>

**Output**

Graphical user interface, text, application

Description automatically generated

**JavaScript Language: Overview and Syntax**

Hello. Welcome to The JavaScript language, Overview and Syntax.

After watching this video, you will be able to

describe JavaScript primitives and objects.

JavaScript is a scripting language that is derived from the ECMAScript standard and originally

designed to run on the Netscape Navigator browser.

Virtually all browsers now support JavaScript.

Although the name “Java” appears in the word “JavaScript”, the two languages are

not related.

When a JavaScript interpreter is embedded in a browser, the result is the ability to

create dynamic web pages:

JavaScript adds behavior to otherwise static web content.

The content of a web page can be dynamically changed with the scripting capability of the

JavaScript interpreter.

JavaScript code acts on the document object model that the web browser generates.

One of the ways that server programming and browser scripting work together is in an architecture

that is called Ajax, or Asynchronous JavaScript and XML.

The term "Ajax" encompasses more than asynchronous server calls through JavaScript and XML.

Ajax represents a series of techniques that provide richer, interactive web applications

through HTML, JavaScript, Cascading style sheets, and modifying the web page through

the Document Object Model.

Nowadays, JSON is commonly used instead of XML.

In JavaScript, there are five primitive types that are associated with various primitive

values:

Number: All numbers, such as 0 or 3.1412,

String: All strings, such as “Hello World”,

Boolean: The values true or false,

Null: The value null.

Undefined: The value undefined, since a data type has not been assigned or the variable

does not exist.

All other non-primitive data types are objects.

The number primitive represents both integer and floating point values, the value NaN (not

a number), and Infinity.

Integers can be coded as base 10 (decimal), base 8 (octal), or base 16 (hexadecimal) values.

The integer literal 16 in decimal, 020 in octal, and 0x10 in hexadecimal all have the

same value.

All numbers in JavaScript are represented internally as double precision or 64 bit floating

point numbers.

JavaScript strings are delimited by either double or single quotation marks.

There is no behavior or methods that are associated with primitive data types.

The primitive types number, string, and boolean can be wrapped by their object counterparts.

Wrapper objects have the same name as the primitive type, except they start with an

uppercase letter.

Like most object-oriented programming languages, JavaScript provides built-in ways to convert

between these wrapper objects and primitive values.

The wrapper objects use special methods such as the valueOf and toString methods to convert

between objects and primitive literals.

The typeof keyword in JavaScript is used to find out the data type of the supplied operand.

Notice how strings created without the new keyword have a type of string.

The keyword new is used to create the String wrapper object.

This object can be converted to a primitive string type by calling the valueOf function

on the object wrapper class.

Arrays are specialized collection objects that aid the programmer in the storage and

retrieval of data by indexed keys.

Arrays use a zero-based indexing scheme, meaning that the first element of an array has an

index of zero.

Arrays grow or shrink dynamically by adding or removing elements.

The length property holds the number of elements that occur in the array.

Arrays can be declared by using either an array constructor or an array literal.

When declaring an array with a constructor, you use the new array keywords and specify

the array elements as parameters of the new array.

Array literals are created by declaring the array elements within square brackets.

You then assign the array to a variable, as seen in the last example on the slide.

The Date object is a specialized object that is used to hold the date and time.

The constructor for a date object is in the format: new Date ([with optional parameters]).

If you create a Date object without any parameters, JavaScript returns an object that contains

the current local date and time.

If you send this date object to the console or try to display the date object on a web

page, JavaScript automatically applies a toString method to the object.

The result that is displayed is a string version of the date as shown on this slide.

You can also create new dates by passing parameters to the new Date function.

Examples on the slide show how either string or numeric values can be used for the date

parameters when creating new date objects.

As with other object-oriented languages, JavaScript creates error object instances when an exception

occurs.

The error object instance includes two properties that contain information about the error:

The message property contains a description about the error.

The name property identifies the type of error such as a RangeError.

A RangeError is an error instance that is created when a numeric value or parameter

is outside of its valid range.

Besides a generic error, there are six other core errors types in JavaScript, three of

which are shown on the slide.

The other three are EvalError, ReferenceError, and SyntaxError.

The Error object can be extended to create custom error types.

The last line of the slide shows the creation of a generic error object with a custom error

message in the parameter field.

In this video, you learned:

JavaScript is a scripting language that adds behavior to otherwise static web content.

Primitives are values, and have no properties or methods. Examples of primitives include:

number, string, boolean, null, and undefined.

Wrapper objects allow objects of corresponding primitive data types to be created.

Wrapper objects can store a primitive value and provide methods with which to process

it.

Wrapper objects have the same name as the primitive type, but they begin with a capital

letter to differentiate them from the primitive data type itself.

Some examples of wrapper objects are array, date, and error objects.

**JavaScript: Variables and Control Statements**

Hello. Welcome to JavaScript Variables and Control Statements.

After watching this video, you will be able to:

Explain how variables are declared and used in JavaScript, and

describe JavaScript control structures,

Variables are declared with the var keyword followed by the variable name, as in the example,

var age,

Variables can be declared and initialized in one step, as in the example, var age = 54,

You can assign a value to the variable at a later time, or reassign the value of a variable,

Since JavaScript is a loosely typed language, you do not need to declare the data type of

a variable. The variable assumes the data type from the value of the field during assignment

and the type of a variable can change during program execution.

Beware of not initializing variables though – the value of ‘var age’ – that has

not been assigned a value, is not zero, or an empty String, or any other useful default

value; it is ‘undefined’.

Remember that the only keyword is ‘var’, and JavaScript has no way to know whether

this variable is a numeric or a string, or anything else, until you provide a value.

It therefore cannot decide by default whether to assign a zero or an empty string.

Variable names, or identifiers, have these rules:

The name must start with a letter, underscore (\_), or a dollar sign ($),

Subsequent characters can also be digits [0-9] Identifiers are case-sensitive.

Variables also have a scope:

Variables declared within a function have a scope local to that function,

Variables declared outside of a function have a global scope.

Variables with a global scope can be used elsewhere in the JavaScript program,

Variables declared without the var keyword have a global scope.

Variables that are not initialized have a value of undefined.

You learn about functions shortly.

Conditional statements are the set of commands that are used to perform different actions

for different conditions:

In JavaScript, the IF statement is the way the program logic decides which path to take

based on the current values of variables or object properties,

The JavaScript syntax for a decision begins with the keyword IF, followed by the condition

to test; then the statements that run if the condition yields a true result.

The true processing follows immediately after the test condition and is delimited by braces,

unless it is a single statement.

If the condition resolves to false, the statements that follow the else keyword are executed.

Indentation of the statements of a compound control statement is not required in JavaScript

However, a programmer can find that indentation helps in deciphering the control statement.

Many JavaScript aware text editors automatically indent the control structure to make it more

readable.

Unlike Java, there is no block statement scope in JavaScript.

Having no block statement scope means that variables declared inside one IF condition

can be used outside the scope of that condition.

JavaScript supports the switch statement as an alternative to the IF then ELSE control

statements.

The condition that is being tested is placed in parentheses that follow the switch keyword.

The expression parameter of the switch statement can evaluate to any number or string value.

The labels in the case statement are enclosed in quotation marks when the labels represent

string values of the expression.

The program looks for a case clause with a label that matches the value of expression

and then transfers control to that clause, running the associated statements.

If no matching label is found, the program looks for the optional default clause, and

if found, transfers control to that clause, issuing the associated statements.

The break keyword is used to prevent the code from automatically falling into the next case

clause.

The FOR loop repeats a series of statements for any number of times.

The FOR loop takes three parameter arguments, namely the initial value, the condition that

is being tested, and the increment expression.

When the FOR loop runs, the following occurs:

First, the initial expression is set,

second, the conditional expression is evaluated.

If this condition evaluates to true, the loop statements runs and the increment expression

is updated.

If the condition evaluates to false, the FOR loop terminates.

If the loop does not terminate, control returns to the second step and the conditional expression

is evaluated again.

The WHILE loop is another common loop in JavaScript,

The loop repeats while the condition remains true,

The WHILE loop assumes that the condition reaches a conclusion and then exits the loop.

Make sure that the condition evaluates to false at some point; otherwise the loop never

terminates.

In this video, you learned:

Variables are declared using the keyword var followed by the variable name.

You can initialize variables at the time of declaration or assign a value later.

You do not need to declare the data type of a variable. Variables take their data type

from the value assigned and can change type during program execution.

Variables also have a scope .

Variables declared within a function have a scope local to that function.

Variables declared outside of a function have a global scope.

Variables declared without the var keyword have a global scope.

When executing, the flow of the program is directed by control statements, including:

Conditional statements like if...then...else,

Switch statements,

repeat statements like for loops and while loops.

**JavaScript: Functions and Prototypes**

Hello. Welcome to JavaScript Functions and Prototypes.

After watching this video, you will be able to:

Describe functions in JavaScript, and

describe prototypes in JavaScript.

A function is a block of code which can be called from any point in a script after being

declared.

A function is made up of the following parts:

The keyword function,

The name of the function,

Parentheses, with optional parameter arguments,

Curly braces, with the logic,

The last statement in a function block is the optional return statement that returns

control back to whatever called the function.

This example is a function that is named add.

This function takes two parameter arguments and returns the sum of the arguments or concatenates

the two arguments if they are strings.

Notice that you do not specify the data type for the arguments of the function.

The data types are determined by the values of the arguments that are being passed to

the function.

There is no specific return type declared , the function returns whatever type is

required.

In this case, the return is a simple addition or concatenation of the input parameters.

The action chosen depends on what data is provided to the function.

If the values can be interpreted as numbers, they are added.

If they can be interpreted as strings, they are concatenated.

This is an example of declaring a function named Car that accepts three arguments as

parameters

A little further down, you specifically ask the function to run, by declaring the statement:

var c = new Car with the arguments “meridian”, “Sabre GT” and 2012.

In the Car function, the keyword "this" refers to the current instance of the Car object

that is being created. In other words, an instance of Car that is associated with the

variable named c.

The getName function of Car returns the make, model, and year of the newly created Car object.

By using prototypes, you can easily define properties and methods for all instances of

a particular object.

Prototypes exist for all JavaScript objects that can be created with the new keyword.

All object constructors create objects that inherit the properties and methods that are

defined by the prototype for that object.

Any object that gets instantiated inherits the current state of the prototype.

Scripts can override prototype properties and functions.

If they do so, these changes affect the current working instances of objects that match the

prototype.

In the Car example that is shown on the slide, you can add another property to the Car prototype,

with the statement:

Car.prototype.property\_name as in:

Car.prototype.floor\_model = true;

Any Car object that you create, or any car object that is already created, automatically

inherits the new floor\_model property and its value.

Here is another example that uses a prototype to change Car instances.

This time, you add a method function called getName to the Car prototype.

Now when a Car object is instantiated, it also includes the getName function that returns

the make, model, and year.

All existing instances of the Car object also inherit the getName method.

Functions are usually declared first, and not run until you specifically ask them to,

as you saw in some of the previous examples.

Auto-invocation or self-executing functions start running immediately after being declared.

The functions and variables inside self-executing functions are only available to the code inside

the self-executing function.

Auto-invocation functions can also be unnamed or anonymous functions, and have the format

that is shown in the code block on the slide.

Self-executing functions are often used to initialize data or to declare DOM elements

on the page.

In this video, you learned:

A function is a block of code which can be called from any point in a script after being

declared.

Functions can take arguments passed as parameters and can return results.

By using prototypes, you can easily define properties and methods for all instances of

a specific object.

Prototypes exist for all JavaScript objects that can be created with the new keyword.

To add a new function to the template for the object, modify the prototype for the object.

Self-executing or auto-invocation functions start running immediately after they have

been declared. The functions and variables are isolated from the rest of the script.

**Client-Side JavaScript: with HTML**

Hello. Welcome to Client-side JavaScript.

After watching this video, you will be able to:

Define a client-side script.

Give examples of when client-side script can be used.

Explain how to use the &lt;noscript&gt; tag.

Describe Event Binding in scripts.

A client-side script is a program that accompanies an HTML document or might be embedded directly

in the HTML document itself.

The script program runs on the client device when the document loads, or at some other

time such as when a link is activated or when a button is clicked.

Although JavaScript is widely used as a scripting language in HTML, other scripting languages

can be used instead.

Scripts offer authors a means to modify and extend HTML documents in highly interactive

ways.

Scripts can run after an HTML document is loaded.

Scripts can be used to validate forms or to process input as it is typed.

Scripts can be triggered by events that occur on a web page, such as the clicking of a button.

Scripts can be used to dynamically create document elements on an HTML page.

This slide shows two ways in which the <s> tag is used to include scripts in an HTML</s>

document.

Example 1 demonstrates how you can include a script directly inside the HTML document.

This method is good for short scripts, but when a script is long, the method used in

example 2 is preferred.

Example 2 uses the src attribute to point to an external script file.

This method has several use cases, such as importing JavaScript libraries for complex

interactions or using the same script across several HTML documents.

Some users who visit your website might disable JavaScript from running, or they might be

using a browser that does not support scripting.

To allow for these situations, place the content for the alternative path within the &lt;noscript&gt;

tag.

If the browser does not support scripting, the browser runs the section of code that

is within the noscript tag.

Scripts can be run on the detection of certain events that happen when the page is running

in a browser session.

For example, the onload event can run a script when the browser finishes loading a page

or a function can be performed when the onclick event occurs.

This event occurs when the pointing device is clicked over an element, such as a button,

that declares a handler for the event.

The event handler is a function that declares what to do when the button is clicked.

Here the inline event handler named showAnswers runs when the button is clicked.

In this video, you learned:

A client-side script is a program that accompanies an HTML document.

It may even be embedded into the HTML itself.

Scripts offer developers ways to extend HTML documents to enhance the user experience,

especially by incorporating more interactive elements.

You can use the script tag to include a script within the HTML document, or to call a script

from an external file.

Use the noscript tag to provide an alternative when scripting is disabled.

Scripts can be bound to events so that they run automatically.

For example, the onload event can run a script when the browser finishes loading a page.

Graphical user interface, text, application, email

Description automatically generated

**Client-Side JavaScript: with DOM**

Hello. Welcome to Client-side JavaScript.

After watching this video, you will be able to:

Describe the document object model (DOM) hierarchy.

Describe the window and document objects.

Identify the DOM objects that are commonly used in JavaScript applications for working

with HTML documents.

The document object model is the programming interface between HTML or XHTML and JavaScript.

The Document Object Model (DOM) is a browser-based interface for applications and scripts to

dynamically access and update the content, structure, and style of documents.

JavaScript uses the DOM to access and modify web page elements in the web browser.

The World Wide Web Consortium released four levels of the Document Object Model specifications.

Each successive level provides a more detailed set of features for describing structured

documents.

The different browsers have various levels of compatibility with the DOM standard.

The discussion of the DOM in this unit focuses on the DOM Level 1 Core and DOM Level 1 HTML

APIs to access HTML elements with the DOM.

The JavaScript engine in most web browsers fully supports DOM level 1.

Here is a representation of the basic DOM model for browsers.

The window object is at the top of the DOM hierarchy and controls the environment that

contains the document.

The history object keeps internal details about the recent history of pages in the browser.

The history object has methods for letting you simulate clicking the back or forward

buttons in a browser.

The location object contains information about the URL of a page.

The navigator is an object representation of the client Internet browser or user agent.

There is no standard that applies to the navigator object, so the property values returned when

running queries on the navigator object are not consistent across browsers.

The screen object is used to derive information about a user's screen, such as the dimensions

of the display screen.

The screen object is useful for determining the screen size of browser windows that run

on mobile devices.

The document object provides access to all HTML elements within a page.

Each HTML document that gets loaded into a window becomes a document object.

The window object is the outermost global container of all the objects in the DOM hierarchy.

When the browser loads a page, a window object is automatically created for you.

You can then access the window object properties and functions from your JavaScript code.

In client-side JavaScript, the Window object serves as the global object and everything

in the DOM takes place in a window.

A number of predefined methods exist for the window object.

The window.alert, window.confirm, and window.prompt dialogs that are used in web pages, come from

the global window object.

You can leave out the window prefix for methods in the DOM API. So the window.alert method

can be coded more simply as alert with a message argument.

This figure shows the object model for a simple HTML document.

Notice how the object hierarchy matches the HTML containment hierarchy on the left.

The object diagram can also be represented as a tree structure that corresponds to the

structure of the HTML document.

The branches of the tree structure are termed nodes.

There are two types of nodes in the W3C DOM, element nodes and text nodes.

All HTML tags (html, head, meta, title, and body) are element nodes.

The nodes that contain actual text that go between an element start tag and end tag,

are text nodes.

The figure shows the DOM level 2 tree for the FORM portion of the document.

The line feeds between elements are text nodes and are part of the DOM level 2 tree.

The DOM level 2 tree includes a line feed text node before the paragraph and input elements.

The input element includes a text node that contains all the text that follows the input

tag.

An additional line feed text node follows the input element.

The DOM level 0 for the form portion of the document would have only the form, p, and

input boxes.

The DOM level 2 adds the carriage returns, tabs, and spaces, sometimes referred to as

"white space".

In this video, you learned:

The document object model (DOM) is the programming interface between HTML or XHTML and JavaScript.

Each successive level of the DOM provides a more detailed set of features for describing

structured documents.

Different browsers have different levels of compatibility with the DOM standard.

The basic DOM for browsers is a hierarchy that includes objects that perform different

functions.

For example, the window object controls the environment of the document, the location

object contains information about the URL of the page, screen object derives information

about the user’s screen, and the document object provides access to all HTML elements

within a page.

DOM levels define object types, with which developers can build a variety of documents,

from plain HTML documents for web pages, to more complex forms.

**JavaScript DOM Objects**

Hello. Welcome to JavaScript Dom Objects.

After watching this video, you will be able to

identify the DOM objects that are commonly used in JavaScript applications for working

with HTML documents.

The W3C DOM level 2 defines 12 different types of nodes, seven of which have direct applicability

in HTML documents.

Node types that are not applicable to HTML are omitted from the table.

This table is of interest so that when you view a DOM tree, you understand the meaning

of the numeric node types.

Each node type is a named constant that is also represented by an integer value.

So, for example:

An ELEMENT\_NODE type is represented by the integer 1.

An ATTRIBUTE\_NODE type is represented by the integer 2.

A TEXT\_NODE type is represented by the integer 3.

A COMMENT\_NODE is represented by the integer 8.

Table

Description automatically generated

In the DOM tree, the node name for ELEMENT\_NODE type is the name of the element or tag.

For example, if you are looking at a DIV element, the node name is DIV.

If the DIV element has an attribute like id=div123, then the attribute name is “id” and the

attribute value is “div123”, which is the name-value pair.

Another example:

If a paragraph element is followed by some text, the text string has a node name of hash-

sign-text, and the node value is the text string itself.

The table lists some of the DOM Level 2 node object properties and corresponding data types.

Table

Description automatically generated

You see these properties when you view a DOM tree for an HTML page in a browser’s developer

tools, like Chrome’s DevTools.

How do you access the elements on an HTML page with the DOM API?

When the document is loaded, the browser creates arrays for forms, images, anchors, links,

applets, and embeds.

It then places all the objects of each type into these arrays.

The arrays are indexed as they occur in the source document. The first index of each array

starts at zero.

Each of the array types, such as forms[], contains an array of elements[] with each

index element being the fields or buttons that occur in that form.

You can reference the element named field1 in the figure by its relative position as

document.forms[0].elements[0].

You can reference the same field with named elements; for example, document.forms["form1"]

.elements["field1"], or even the shortened document.form1.field1.

Since only one document can be contained in a window, shown by the dashed line in the

figure, you can leave out the window prefix. However, you cannot omit the document prefix

from the referenced object.

The id attribute identifies an element in a document.

The id attribute of an element is used by scripts to refer to the element with a name

that matches the value of the id attribute.

In order to assign a scriptable reference name to an HTML element with the id attribute,

use the following conventions:

The id must be a unique name in the document.

The name must be in quotation marks when assigned to the id attribute.

The name must not start with a numeric digit.

The function that is used to return a node object that matches the id value is document.getElementById

with the name of the id as a parameter argument.

It is suggested that the same value is used for both the id and the name attribute (seen

earlier) when both are used.

In this video, you learned:

How to work with DOM objects in JavaScript, including:

You can access nested objects using a dot notation and

objects can be named to make accessing them from the script easier.

**JavaScript APIs**

Hello. Welcome to JavaScript APIs.

After watching this video, you will be able to:

Explain how to work with nodes.

Describe how to modify the content of an element.

Explain how to modify the inline style of an object.

Describe how to modify the attributes of an object and

describe how to use Window object methods and events.

The DOM Level 1 core and Level 1 HTML specifications have too much detail for one unit.

Instead, the remainder of the unit focuses on script functions and properties that are

commonly used when working with HTML pages.

Some of these common APIs used in when accessing HTML DOM elements in web pages are seen on

the slide.

Text, table

Description automatically generated

DOM elements are reviewed next.

To retrieve a node reference for an element of a document given an id, use the document.getElementById

function and pass the id value as an argument.

This returns one specific HTML or XML element if a matching unique id is found on the web

page.

The getElementsByTagName function retrieves a NodeList of elements with a specified tag

name.

The NodeList contains an array of elements in your document.

You provide the name of the element you are interested in and an array of all elements

with that name found in the document is returned.

For HTML elements, the tagName parameter is the literal name of the HTML tag.

If you run the function getElementsByTagName with a “p” as a parameter argument, a

NodeList of all the paragraphs in the document is returned.

This example shows how you can retrieve all the image elements from a web page by using

the getElementsByTagName function and passing “img” as the parameter argument.

The result is a NodeList that is assigned to the variable imgSet.

In the next part of the code, you loop through the nodeList and concatenate the results into

a DOM output stream.

The src attribute is a common property for the img tag. The src attribute indicates the

location of the image source.

The document.write() function adds the script-generated HTML to the document.

You can use the DOM API function document.createElement(TagName) to create an element in the current document.

After creating the element, you can use any of a number of functions to place the element

in the appropriate location within the document.

Examples of these functions include the insertBefore, appendChild, or replaceChild function that

can be used to add the newly created element into the document.

This example shows the source code for adding a node to a document.

A new paragraph element is being created that includes a text node with the string “Hello

world!”

The text node is then appended as a child of the paragraph element.

Finally, the entire paragraph with text is appended as a child node at the end of the

body node of the HTML page.

The function element.innerHTML retrieves or sets the contents of an HTML element.

The innerHTML property returns all child elements as a text string.

With the element.innerHTML function, you can change the contents of an HTML element, by

setting it to a text string that can include HTML tags.

Setting the innerHTML value of an element to a string removes all of the current child

elements. The browser then parses the string and sets the contents of the HTML element.

You can use the element.style method to retrieve or set the inline CSS style for a particular

element.

If you use element.style to set the style of an element, it overrides any setting from

a CSS style sheet with one specific style.

The way to set the style in JavaScript is with the format element.style.propertyName

= value.

For example, if you had an element &lt;div style="color:blue"&gt;.

Here the &lt;div&gt; tag is used to group block-elements to format them with a color style. You can

change the style of this div tag with the JavaScript statement:

div.style.color = 'red';

In contrast, the element.setAttribute('style', …) wipes out all previously set inline CSS

styles.

The function, element.setAttribute with parameters(attrName, attrValue) dynamically modifies the attribute

of an element.

In the example, the src attribute of an element with an id of theImage is set to a different

target image.

The function element.removeAttribute(attrName) removes an attribute from an element.

The function element.getAttribute(attrName) retrieves the value of the specified attribute

in the element, if it exists.

Here are some window object functions and events.

To open a new browser window, use the window.open() function.

This method returns a reference to the new window object.

You can use this reference later to close the window, with the reference\_name followed

by the close() function.

The parameters of the window.open function are:

URL - A string that indicates the location of the web page to be displayed in the new

window.

You can pass an empty string if you are going to write some script- generated content to

the new window in the current URL context.

Name - A string that specifies the name of the window.

Features - An optional string that specifies the features of the window, such as its placement

and dimensions. The features string is a comma-separated list of name- value pairs.

Replace – An optional boolean value. If true, the new location replaces the current

location in the browser history.

The window.onload function can be used to start a function after the page is loaded.

The function, window.dump("message") writes a string into the console for the web browser.

The dump() function is a less intrusive way to display diagnostic information than the

alert() method.

Finally, the window.scrollTo(x-value, y-value) scrolls the web browser to a particular set

of coordinates on a page.

The onload event handler runs in the current window after the document loads the web page.

In the example, the onload event causes an anonymous function to run. This function,

in turn, runs the function addPara().

In this video, you learned:

You can retrieve a reference to a node using:

document.getElementById(id) - Returns one specific element that is based on the id attribute.

document.getElementsByTagName(tagName) - retrieves a of elements with the specified tag.

You can create an element using:

document.createElement(TagName).

And place it using:

insertBefore, appendChild, or replaceChild.

You can modify elements using:

element.innerHTML to retrieve or set the contents of an HTML element.

element.style to retrieve or set the inline CSS style.

element.setAttribute to modify the attributes of an element.

You can manage a window object using functions including:

window.open to return reference to a new window object for the web browser.

window.dump("message") to write a string into the console for the web browser.

This is not an exhaustive list – many more functions exist that enable you to work with

HTML elements and nodes.

**Lab: Validating a JavaScript form**

**2. Add the <script> tag**

We use the <script> tag to embed executable code, usually JavaScript, into an HTML page. The tag can contain scripting statements, or it can refer to an external script file. We use a type attribute to specify the scripting language.

Although you can put the <script> tag anywhere in your HTML document, for this lab you'll put it in the <head> section.

Replace the <head> section of your file with the following code. It tells the browser that the code we are about to put inside the <script> tag must be executed as JavaScript.

<head>

<script type="application/javascript"></script>

<title>Contact Details</title>

</head>

**3. Add a function**

Now you'll specify what happens when a user clicks the **Submit** button. We specify this behavior with a user-defined JavaScript *function*, which is a block of code that is executed when it's called. A function can be called any number of times.

A function in JavaScript looks like this:

function functon\_name()

{

// code goes here

}

Let's add an empty function that has the name checkdata. Replace the <script> tags in your file with the following code:

<script type="application/javascript">

function checkdata()

{

}

</script>

**4. Access HTML controls within JavaScript**

The function you've created is intended to validate the contents of each of the input elements in the form. To access the data for an element, the script needs to refer to the correct element.

One way to identify an element is to use a method called getElementByID(elementID). The following line of code returns the element with the ID name:

document.getElementByID("name");

The following lines of code enable you to access the name and email input elements of the form. The references to the elements are stored in two JavaScript variables named username and emailid.

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

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

**5. Access and check data**

When the references to the elements are stored in the variables, the values of the elements can be retrieved using the value attribute. If username is the variable that contains the input element's reference, then its value can be accessed using

username.value

To check if this value is blank, we can use the following statement:

username.value == ""

"" indicates an empty string.

**6. Execute a set of statements based on a condition**

If the value is blank, we will print an error message and return the focus back to the empty element.

To perform this action, we use a JavaScript *conditional statement* called the if statement. The if conditional statement allows us to specify a block of code to be executed *if* a condition is true.

The syntax of the statement is as follows:

if(condition){

//block of code to be executed, if the condition is true.

}

Let's check if the username value is empty by using an if statement:

if(username.value==""){

return false;

}

If the value is blank, the return false; statement returns a boolean value false from the checkdata function that we added in step 3.

We check all input elements of the form in this way to determine whether they are empty.

**7. Display error messages**

You can display a message to a user with the help of a pop-up alert message box. To do this, you will use the alert method.

Let's use this method within the function to alert the user.

if(username.value==""){

alert("Please enter the name");

fname.focus();

return false;

}

The fname.focus() statement is used to bring the input focus back to the element where we found a problem, in this case, *name*.

We indicate that none of the elements are blank by returning true. So, we need to add a return true statement at the end of the function.

It's a good practice to include comments in your code. Comments will help you and other programmers easily debug any errors that we might encounter while running the code. In JavasScript, we add comments using two forward slashes: //

Our final checkdata function with comments added looks like:

function checkdata(){

//Create references to the input elements we wish to validate

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

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

//Check if username field is empty

if(username.value == ""){

alert("Please enter the name");

username.focus();

return false;

}

//Check if email field is empty

if(emailid.value == ""){

alert("Please enter the email");

emailid.focus();

return false;

}

//If all is well return true.

return true;

}

**8. Execute a function when the form is submitted**

Our final step is to ensure that the checkdata function is executed when the form is submitted. We do this using the onsubmit event. This event occurs when users click the **Submit** button.

T

he following code links the onsubmit event to the checkdata function:

<form id="form1" onsubmit="return checkdata()">

{:codeblock}

This code ensures that the checkdata function is invoked when the form is submitted.

Following is the complete code along with the HTML form and JavaScript validation function. Copy and paste the code into your file and check it to determine if it is properly validating:

<!DOCTYPE html>

<html>

<head>

<title>Contact Details</title>

<script type="application/javascript">

function checkdata(){

//create references to the input elements we wish to validate

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

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

//Check if username field is empty

if(username.value == ""){

alert("Please enter the name");

username.focus();

return false;

}

//Check if email field is empty

if(emailid.value == ""){

alert("Please enter the email");

emailid.focus();

return false;

}

//If all is well return true.

alert("Form validation is successful.")

return true;

}

</script>

</head>

<body>

<h2>Enter your contact Details:</h2> <br>

<form id="form1" onsubmit="return checkdata()">

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

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

<br>

<br>

<label for="email">E-mail ID :</label>

<input type="email" id="email" name="email">

<br>

<br>

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

<input type="reset" value="Reset">

</form>

</html>

**Hands-on lab on Javascript**

The purpose of this lab is to javascript before you set off doing server side coding with Node JS. This lab presumes that you have completed all the other labs in the course IBM HTML CSS and JS for Web development.

**Objective**

After completing this lab you will be able to:

1. Write and run Javascript on the browser console
2. Create variables, work with conditional statements, create loops and define methods in Javascript.

**Task 1 – Open the browser console.**

In this task, we are going to run the Javascript code in the browser console. The Chrome browser has a v8, which is Google’s open source high-performance JavaScript engine.

1. Open a new blank browser page clicking on Ctrl+T(Windows) or Command+T(Mac) to open a new tab.
2. Right-click anywhere on the new blank browser tab and choose **Inspect** or **Inspect Element** depending on the browser you are using. The image below is for Chrome Broswer.

Graphical user interface, application

Description automatically generated

1. Go to the **Console** tab, as shown below. You will see a command prompt. You can run the rest of the tasks there.

Graphical user interface, text

Description automatically generated

1. If your console has any logs printed, clear it by running the following command. This is not mandatory. It will just help in a fresh start.

clear()

**Task 2 - Running JS commands**

**NOTE:** At any point of time, when you want to clear the console run the clear() command.

To run the commands we will use the command prompt on the browser control. Type or paste the command and press **enter** to run the command.

1. Let's start with a simple code to print **Hello World!** to the console. Run the following command.

console.log("Hello World!")

The output would be as below.

Graphical user interface, application

Description automatically generated

The undefined means console.log doesn't return anything.

1. Let's create some variable and print them. Run the following command.

let num = 5

var mystr = "John"

console.log(num)

console.log(mystr)

Both let and var can be used to create variables. var is used when you want the variable to have global scope and let is used when you want the variable to have scope within the block where it is created.

1. Let's create a constant and print it. Run the following command.

const pi\_val = 3.147

console.log(pi\_val)

Const is used to declare variable whose values can never change

1. Let's create function which prints any value that is input to it.

function printMyInput(user\_input) {

console.log("The parameter passed is "+user\_input)

}

1. Call the function you created in the previous step once with a number and once with a string.

printMyInput(9)

printMyInput("John")

1. Let's rewrite the function printMyInput according to the ES6 standard. This syntax is also called arrow functions and provide a shorthand to write functions.

let printMyInputES6 = (user\_input)=>{

console.log(user\_input)

}

1. Call the function you created in the previous step once with a number and once with a string.

printMyInputES6(9)

printMyInputES6("John")

Since the function is passed a single value and the body of the function is a single line, the brackets can be omitted. The code can also be written as below.

let printMyInputES6Short = user\_input => console.log(user\_input)

Now when we call it, the output should remain the same.

printMyInputES6(9)

printMyInputES6("John")

**Task3 - Operators, Conditions, Loops**

In this task you will be running some javascripts from which you can learn how to use operators, controls and loops.

Ensure that you understand the code in each file. These are primitive and foundational for your understanding of JS

1. **Arithmetic operators** are operators that we use to perform arithmetic operations.
   * + (plus) operator is used to add
   * - (minus) operator is used to subtract)
   * \* (star or asterisk) operator is used to multiply
   * / (slash) operator is used to divide
   * \*\* (double star) operator is used for exponentiation/power
   * % (percentage) operator is used for modulus operation

console.log("5 + 3 = ",5+3)

console.log("7 - 3 = ",7-3)

console.log("8 \* 2 = ",8\*2)

console.log("27 / 3 = ",27/3)

console.log("4 power 3 = ",4 \*\* 3)

console.log("19 mod 4 = ",19%4)

1. **Assignment operators** are operators that are used to assign values to variables
   * = operator is used to assign value on the right to the variable on left
   * += operator is used to increment the value stored in the left operand by the value of the right operand and store it back to the left operand
   * -= operator is used to decrement the value stored in the left operand by the value of the right operand and store it back to the left operand
   * \*= operator is used to multiply the value stored in the left operand by the value of the right operand and store it back to the left operand
   * /= operator is used to divide the value stored in the left operand by the value of the right operand and store it back to the left operand
   * \*\*= operator is used to raise the value stored in the left operand to the power value of the right operand and store it back to the left operand
   * %= operator is used to get modulus of the value stored in the left operand by value of the right operand and store it back to the left operand

x = 5

console.log("Old value x ",x)

x += 3

console.log("New value x ",x)

y = 5

console.log("Old value y ",y)

y -= 3

console.log("New value y ",y)

a = 6

console.log("Old value a ",a)

a \*= 3

console.log("New value a ",a)

b = 6

console.log("Old value b ",b)

b /= 3

console.log("New value b ",b)

c = 6

console.log("Old value c ",c)

c %= 3

console.log("New value c ",c)

d = 6

console.log("Old value d ",d)

d \*\*= 3

console.log("New value d ",d)

1. **Comparison Operators** are used to compare values or variables against values or other variables

* == operator checks if the operand on the left is of equal value to the operand on right
* === operator checks if the operand on the left is of equal value and equal type to the operand on right
* != operator checks if the operand on the left is not of equal value to the operand on right
* > operator checks if the operand on the left is greater than that on the right
* < operator checks if the operand on the left is lesser than that on the right
* >= operator checks if the operand on the left is greater than or equal to that on the right
* <= operator checks if the operand on the left is lesser than or equal to that on the right

//Checking equality of 5 number type and 5 string type

console.log("5=='5' ", 5=='5')

console.log("5==='5' ", 5==='5')

console.log("5===5 ", 5===5)

console.log("5 != 5 ", 5 !== 5)

console.log("5 != 6 ", 5 != 6)

console.log("5 != '5' ", 5 !== '5')

console.log("5 > 2 ", 5 > 2)

console.log("5 > 7 ", 5 > 7)

console.log("5 > 5 ", 5 > 5)

console.log("5 < 7 ", 5 < 7)

console.log("5 < 2 ", 5 < 2)

console.log("5 < 5 ", 5 < 5)

console.log("5 >= 5 ", 5 >= 5 )

console.log("5 <= 5 ", 5 <= 5 )

1. **Logical Operators** are used to combine more than one conditions.

* && operator checks if the condition on left and right are true. Returns true only of both conditions are true. Else returns false.
* || operators checked if either the condition on the left is true or right is true. Returns true even if one of the two conditions is true.
* ! operator checks if the condition is not met.

Practice exercises for logical operators will be covered along with if-else conditions.

1. **if-else-else if** Conditional statements are very useful to control the flow of your code.

//Accept a input from the user. If it is a number print the multiplication table for the number. Else print the input and the length of the input.

let user\_input = prompt('Enter a value');

//check if the user input is not a number

if(isNaN(user\_input)) {

console.log("Your input is ",user\_input)

console.log("The length of your input is ",user\_input.length)

} else {

console.log(user\_input, " X 1 = ",user\_input\*1)

console.log(user\_input, " X 2 = ",user\_input\*2)

console.log(user\_input, " X 3 = ",user\_input\*3)

console.log(user\_input, " X 4 = ",user\_input\*4)

console.log(user\_input, " X 5 = ",user\_input\*5)

console.log(user\_input, " X 6 = ",user\_input\*6)

console.log(user\_input, " X 7 = ",user\_input\*7)

console.log(user\_input, " X 8 = ",user\_input\*8)

console.log(user\_input, " X 9 = ",user\_input\*9)

console.log(user\_input, " X 10 = ",user\_input\*10)

}

1. **Loops** can be used when the same block of code is to be executed many times.

**for loops** have an initial value, condition based on which the loop is executed and an incremental value.

//Accept a input from the user. If it is a number print the multiplication table for the number.

let user\_input = prompt('Enter a number');

//check if the user input is not a number

if(!isNaN(user\_input)) {

//Using for loop for the repitive statement

for (let i=0;i<10;i++) {

console.log(user\_input, " X ",i," = ",user\_input\*i)

}

}

**while loops** have just a condition based on which a block of code is executed many times.

//The code below is to find the length of the words the user is entering. The loop will go on and on until the user chooses not to continue by pressing 'n'

let do\_more = true

while(do\_more) {

//Accept a input from the user.

let user\_input = prompt('Enter a word');

//check if the user input is not a number and then print the length of the input

if(isNaN(user\_input)) {

console.log("Length of the word you entered is "+user\_input.length)

} else {

console.log("You entered a number. Only words are allowed")

}

let should\_continue = prompt("Do you want to continue. Press n to stop")

if(should\_continue === "n") {

do\_more = false

}

}

1. **switch-case** is used to replace multiple if-else if conditions checking the same variable. After one of the conditions is satisfied and the block of code is executed, the control should explicitly **break** out of the switch block.

let user\_input = prompt('Enter a number between 1 to 7');

//Using logical OR operator to check if the input is a number and it is between 1 to 7

if(isNaN(user\_input) || user\_input< 1 || user\_input>7) {

console.log("Invalid input")

} else {

user\_input = parseInt(user\_input)

switch(user\_input){

case 1: console.log("Sunday");break;

case 2: console.log("Monday");break;

case 3: console.log("Tuesday");break;

case 4: console.log("Wednesday");break;

case 5: console.log("Thursday");break;

case 6: console.log("Friday");break;

case 7: console.log("Saturday");break;

default: console.log("Invalid entry");

}

}

**Task 4 – Collections**

1. Array is an indexed collection. The index positions are from 0. To access the element in first position, we use 0, second position will be 1 and so on. The index of the last position will always be one less than the length of the array.

let myArray = ["Jack","Jill",4,5,true,"John"]

console.log(myArray[0]);

console.log(myArray[5]);

1. To iterare through arrays there is a special for loop, **forEach**, which executed for each value in

let myArray = ["Jack","Jill",4,5,true,"John"]

myArray.forEach(x => {

console.log(x)

})

1. To find the index position and the value, we can use the generic Object.entries method, which can be used with all collection objects. This maps each index position to the value.

let myArray = ["Jack","Jill",4,5,true,"John"]

for (const [idx, value] of Object.entries(myArray)) {

console.log(idx," - ",value);

}

1. **Map** object maps a key to a value. The keys have to be unique. The values can be string, int, float or any other valid javascript datatype. An empty Map object can be create with the new keyword.

let myMap = new Map();

//Here name is key and John is the value.

myMap.set("name","John")

//Here age is the key and 22 is the value.

myMap.set("Age",22)

myMap.forEach((val,key) => {

console.log(key," - ",val)

})

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Syntax** | **Description** | **Example** |
| Declaring Variables var, let, const | let < var\_name > = < value > | **var** - global access, value can chage **let** - access within block where it is declared, value can change **const** - access within block where it is declared, value cannot change | let i = 5; var myStr = "John"; const pi = 3.14 |
| **Strings** | | | |
| length | *string\_obj*.length | **length** Returns the length of the string | let myStr = "Hello"; console.log(myStr.length); Output is 5 |
| split | *string\_obj*.split(*separator*) | **split** Splits the string based on the separator and returns an array. | let myStr = "Hello! How are you?"; console.log(myStr.split(" ")) Output is [ 'Hello!', 'How', 'are', 'you?' ] |
| charAt | *string\_obj*.charAt(*index*) | **charAt** returns the character at a specified index in a string. Index starts at 0 ends at length-1 | let myStr = "Hello";< console.log(myStr.charAt(0)) Output is H |
| replace | *string\_obj*.replace(*"SearchValue","NewValue"*) | **replace** searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced. | let myStr = "Hello User"; console.log(myStr.replace("User","World")); Output is Hello World |
| substring | *string\_obj*.substring(start, end) | **substring** is used to extract characters, between to indices from the given string, and returns the substring. It excludes the last index | let myStr="Hello"; console.log(myStr.substring(1,4)); Output is ell |
| startswith | *string\_obj*.startsWith(searchvalue) | **startsWith** returns true if a string begins with a specified string, otherwise false | let myStr="Hello from the other side"; console.log(myStr.startsWith("Hello")); Output is *true* |
| endsWith | *string\_obj*.endsWith(searchvalue)) | **endsWith** returns true if a string ends with a specified string, otherwise false | let myStr="Hello from the other side"; console.log(myStr.startsWith("side")); Output is *true* |
| toUpperCase | *string\_obj.*toUpperCase() | **toUpperCase** converts a string to uppercase letters | let myStr="hello"; console.log(myStr.toUpperCase()); Output is HELLO |
| toLowerCase | *string\_obj.*toLowerCase() | **toLowerCase** converts a string to lowercase letters | let myStr="HELLO"; console.log(myStr.toUpperCase()); Output is hello |
| concat | *string\_obj.*concat(*string1, string2,..,stringN*) | **concat** joins two or more strings. | let myStr="Hello"; let str="World"; console.log(myStr.concat(str)); Output is HelloWorld |
| **Arrays** | | | |
| push | *arr\_name.*push(*value*) | **push** adds new items to the end of an array. | let myArr=["Hello"]; myArr.push("World"); console.log(myArr); Output is ["Hello","World"] |
| pop | *arr\_name.*pop() | **pop** removes the last element of an array. | let myArr=["Hello","World"]; myArr.pop(); console.log(myArr); Output is ["Hello"] |
| length | *arr\_name.*length | **length** sets or returns the number of elements in an array. | let myArr=["Hello","World"]; console.log(myArr.length); Output is 2 |
| indexOf | *arr\_name.*indexOf(*item*) | **indexOf** searches for a specified item and returns its position. | let myArr=["Hello","World"]; console.log(myArr.indexOf("World") Output is 1 |
| lastIndexOf | *arr\_name.*lastIndexOf(*item*) | **lastIndexOf** returns the last index (position) of a specified value. | let myArr=["Hello","World","Hello"]; console.log(myArr.lastIndexOf("Hello"); Output is 2 |
| entries | *arr\_name.*entries() | **entries** Returns and Array Iterator that helps you to iterate through the array and recieve each entry as an array of two elements containing the key and the value, where in the key is the index position of the element and value is the element itself. | const hello = ["h", "e", "l", "l","o"]; console.log(hello.entries()); Output is Object [Array Iterator] {} |  |
| find | Array.find(<arrElemet>=>{ //return boolean based on a condition } | **find** Finds the first occurance of an element in the array which returns true on checking the condition | //Find the first string with s let myarr = ["Mercury","Venus","Earth","Mars"]; let found = myarr.find(val=>{ return val.includes("s"); }) console.log(found); Output Venus |  |
| filter | Array.filter(<arrElemet>=>{ //return boolean based on a condition } | **filter** Finds the all occurances of elements in the array which returns true on checking the condition | //Find the all strings with s let myarr = ["Mercury","Venus","Earth","Mars"]; let found = myarr.filter(val=>{ return val.includes("s"); }) console.log(found); Output [Venus,Mars] |  |
| map | Array.map(<arrElemet>=>{ //return processed value } | **map** Processes the all elements of the array which returns a new processed array of same size | let myarr = ["name","place","thing","animal"]; let found = myarr.map(val=>{ return val+"s"; }) console.log(found); Output [ 'names', 'places', 'things', 'animals' ] |  |
| concat | *arr\_name.*.concat(arr1.name); | **concat** concatenates (joins) two or more arrays. | let hello = ["hello", "world" ]; let lorem = ["along","lorem"] let h = hello.concat(lorem); console.log(h);  Output is ["hello", "world", "along", "lorem"] |  |
| **Map** | | | |  |
| set | mapName.set(key,value); | **set** helps you define a new element with akey and its value | var newMap = new Map(); newMap.set("h", 1); console.log(newMap); Output is {"h" => 1} |  |
| get | mapName.get(key); | **get** helps you return a value of key you are searching for | var newMap = new Map(); newMap.get("h"); console.log(newMap); Output is Map(0) {size: 0} |  |
| keys | mapName.keys(); | **get** is used to get all of the keys associated with the mapName | var newMap = new Map(); newMap.set("h",1); newMap.set("i",2); console.log(newMap.keys()); Output is {"h", "i"} |  |
| values | mapName.values(); | **values** is used to get all of the values to the keys associated with the mapName | var newMap = new Map(); newMap.set("h",1); newMap.set("i",2); console.log(newMap.values()); Output is {1,2} |  |
| has | mapName.has(key\_name); | **has** is used to check if the key passed resides in the map or not, and returns true or false | var newMap = new Map(); newMap.set("h",1); newMap.set("i",2); console.log(newMap.has(i)); Output is true |  |
| delete | mapName.delete(key\_name); | **delete** is used to delete the key and the value from the map | var newMap = new Map(); newMap.set("h",1); newMap.set("i",2); newMap.delete("h"); console.log(newMap); Output is {"i" => 2} |  |
| **JSON** | | | |  |
| Create JSON | let varname={name1:value1,name2:values2,.....} | JSON is a dictionary Object with Key-Value pairs. | let myjson1={}; let myjson2 = {"name":"Jennifer","age":"32"} |  |
| Add entry to JSON | let jsonObj[<key>]=<value> | Adds an entry to JSON Object mapping the key to value | let myjson1 = {}; myjson1["name"]="Jason"; console.log(myjson1); |  |
| **Operators** | | | |  |
| Arithmetic | <Operand1> <Operator> <Operand2> | **+** addition **-** subtration **/** division **\*** multiplication **%** modulus(gives remainder) **++** increment by 1 **--** decrement by 1 | let num1 = 2; let num2 = 2; console.log(num1+num2); console.log(num1-num2); console.log(num1/num2); console.log(num1\*num2); console.log(num1%num2); num1++; console.log(num1); num2--; console.log(num1); Output is 4 0 1 4 0 3 3 |  |
| Logical | condition1 && condition2 condition1 || condition2 ! condition1 | **&&** (AND)is used to check if all the operand conditions are true **||** (OR)is used to check if either of the operand condition are true **!** (NOT) is used to check if the operand condition is not met | let num1 = 12, num2 = 2; console.log(num1>10 && num2>10); console.log(num1>10 || num2>10); console.log(!(num1==num2)); Output is false true true |  |
| Assignment | variable = value variable += incremental value variable -= decremental value %= modulus value /= divide value \*= multiply value | **a=b** assigns the value of b to a **a+=b** adds the value of b to a and stores it in a **a-=b** subtracts the value of b from a and stores it in a **a%=b** divides the value of a by b and stores the remainder in a **a/=b** divides the value of a to b and stores the quotient in a **a\*=b** multiplies the value of a and b and stores the value in a | let num1 = 12, num2 = 2; console.log(num1=num2); console.log(num1+=num2); console.log(num1-=num2); console.log(num1/=num2); console.log(num1\*=num2); console.log(num1%num2); console.log(num1=num2); Output is 2 14 10 6 24 0 2 |  |
| **Loops** | | | |  |
| For Loop | for(initialization;condition;increment/decrement){ //code block } | **for** loops throughout the block of code a number of times making sure the condition is satisfied | for(let num = 0 ; num <=5 ; num++){ console.log(num) } Output is 0 1 2 3 4 5 |  |
| while | while(condition){ //code block } | **while** itrates through the block of code while a specified condition is true | let num1 = 0; let num2 = 5; while(num1 < num2){ console.log(num1) num1++; }  Output is 0 1 2 3 4 |  |
| do while | do{ //code block } while(condition) | **do while** loops throughout the block once before checking condition. | let num = 5; do { console.log(num); num--; }  while(num > 0)  Output is 5 4 3 2 1 |  |
| for in | for (var in object) { //code block } | **for in** is used to itrate through the specific property/type of the object | let arr = ["a","b","c"]; for(let i in arr) { console.log(arr[i]); }  Output is a b c |  |
| **Conditional statements** | | | |  |
| if | if(*condition*){ //code Block... } | **if** a specified condition is true, a block of code will be executed | let num = 5; if(num = 5){ console.log(true); } Output is true |  |
| if-else | if(*condition*){ //Code Block... } else { //Code Block... } | **if** a specified condition is true, a block of code will be executed. in case of false, else block is executed | let num = 5; if(num = 4){ console.log(true) } else { console.log(false) } Output is false |  |
| if-else if-else | if(*condition*){ //Code Block... } else if (*condition*) { //Code Block... } else { //Code Block... } | **else if** to specify a new condition to test, if the first/previous condition is false | let num = 10; if(num < 10){ console.log("number is smaller"); } else if(num = 10) { console.log("number is equal"); } else { console.log("number is greater"); } Output is number is equal |  |
| switch | switch(expression) { case <value1>: //code break; case <value2>: //code break; . . . default: //default code block } | **switch** to select one of many blocks of code to be executed. And **break** is used to end the preocessing within the switch statement. | let num = 2; switch(num) { case 1: console.log("Hello world!"); break; case 2: console.log("Hi"); break; default: console.log("this is default"); } Output is Hi |  |
| **Other useful operations** | | | |  |
| typeof | typeof(operand) | **typeof** operator returns a string indicating the type of the unevaluated operand | console.log(typeOf("Hello")) Output is "string" |  |
| isNaN | isNaN(operand) | **isNaN** determines whether a value is anythying but a number or not. It returns false for a number | console.log(isNaN("Hello")) Output is true |  |
| parseInt | parseInt(string, radix) | **parseInt** is a function that parses a string argument and returns an integer of the specified radix.(radix is a base) | //0011 is 3 for binary, since binary only has 2 numbers 0, 1 the radix is 2 console.log(parseInt("0011", 2)); //Default parseInt takes decimal system console.log(parseInt("54")); Output is 3 54 |  |
| parseFloat | parseFloat(string) | **parseFloat** is a function that parses a string argument and returns an float | parseFloat("3.14") Output is 3.14 |  |

**FINAL PROJECT**

**HTML code**

<!DOCTYPE html>

<html>

    <title>Simple Interest Calculator</title>

    <head>

        <!--Link CSS and Javascript files-->

        <script src="script.js"></script>

        <link rel="stylesheet" href="style.css">

    </head>

    <body>

        <div class="maindiv">

            <h1>Simple Interest Calculator</h1>

            <p>

                <!--Input principal and use a Javascript function to ensure inputed value is positive-->

                <label>Amount</label> <input type="number"  id="principal" oninput="checkprincipal\_is\_positive()">

                <br/>

                <!--Select interest rate using a slider and use a Javascript function to show slider value that dynamically changes-->

                <label>Interest Rate</label> <input type="range"  id="rate" min=1 max=20 step=0.25 value=10.25 onchange="updateRate()"> <span id="rate\_val">10.25%</span>

                <br/>

                <!--Select number of years using a dropdown list-->

                <label>No. of Years</label>

                <select id="years">

                    <option value="1">1</option>

                    <option value="2">2</option>

                    <option value="3">3</option>

                    <option value="4">4</option>

                    <option value="5">5</option>

                    <option value="6">6</option>

                    <option value="7">7</option>

                    <option value="8">8</option>

                    <option value="9">9</option>

                    <option value="10">10</option>

                </select>

                <br/>

                <!--Use a Javascript function to compute the interest rate and to output the result-->

                <button onclick="compute()">Compute Interest</button>

                <br/><br/>

                <!--Show the result-->

                <span id="result"> </span>

            </p>

            <footer>

                &#169; Everyone Can Get Rich <br/>

                This Calculator belongs to Petar Antovski

            </footer>

        </div>

    </body>

</html>

**CSS code**

body {

    background-color:black;

    font-family:arial;

    color:white

}

h1 {

    color:gray;

    font-family:verdana;

    height: 55px;

}

/\* Styles the maindiv \*/

.maindiv {

background-color:white;

color:black;

width: 480px;

padding: 20px;

border-radius: 25px;

align-content: center;

}

/\* Indents the paragraph further to better match the sample image provided \*/

div p {

padding-left: 50px;

}

/\* Ensures that the input fiels are aligned and start at the same pixel \*/

label {

height: 30px;

width: 120px;

display: inline-block;

}

/\* Highlights the selected elements (<span class="highlight"></span> marked in the javascript) in the result paragraph \*/

.highlight {

    background-color: yellow;

}

**Javascript code**

//Function that checks that proper value is entered in the principal field, computes the interest and returns the result

function compute()

{

    //Obtain the values inputted for the principal, interest rate and number of years

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

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

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

    //Check if principal field is empty or negative or zero

    var principal\_is\_not\_empty = document.getElementById("principal");

    if(principal\_is\_not\_empty.value == "" || principal\_is\_not\_empty.value <= 0)

    {

        alert("Enter a positive number");

        principal\_is\_not\_empty.focus();

        return false;

    }

    else

    {

        //Calculate interest

        var interest = principal \* years \* rate /100;

        //Convert the 'No of Years' into the actual year in the future

        var current\_year = new Date()

        var year = current\_year.getFullYear() + parseInt(years);

        //Return the result with highlighted numbers

        document.getElementById("result").innerHTML= 'If you deposit <span class="highlight">' +principal+ ',</span> <br>at an interest rate of <span class="highlight">' +rate+ '%,</span><br>You will receive an amount of <span class="highlight">' +interest+ ',</span> <br>in the year <span class="highlight">' +year+ '</span><br>';

    }

}

function updateRate()

{

    //Show the interest rest value chosen using the slider and update it on change

    var rateval = document.getElementById("rate").value + "%";

    document.getElementById("rate\_val").innerText=rateval;

}

function checkprincipal\_is\_positive()

{

    //Obtain the value inputted for the principal

    var principal\_is\_positive = document.getElementById("principal");

    //Check if principal is positive and greater than zero

    if(principal\_is\_positive.value <= 0 && principal\_is\_positive.value!="")

    {

        alert("Enter a positive number");

        principal\_is\_positive.focus();

        return false;

    }

}

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated