



European IT Certification Curriculum Self-Learning Preparatory Materials

EITC/WD/JSF
JavaScript Fundamentals



This document constitutes European IT Certification curriculum self-learning preparatory material for the EITC/WD/JSF JavaScript Fundamentals programme.

This self-learning preparatory material covers requirements of the corresponding EITC certification programme examination. It is intended to facilitate certification programme's participant learning and preparation towards the EITC/WD/JSF JavaScript Fundamentals programme examination. The knowledge contained within the material is sufficient to pass the corresponding EITC certification examination in regard to relevant curriculum parts. The document specifies the knowledge and skills that participants of the EITC/WD/JSF JavaScript Fundamentals certification programme should have in order to attain the corresponding EITC certificate.

Disclaimer

This document has been automatically generated and published based on the most recent updates of the EITC/WD/JSF JavaScript Fundamentals certification programme curriculum as published on its relevant webpage, accessible at:

<https://eitca.org/certification/eitc-wd-jsf-javascript-fundamentals/>

As such, despite every effort to make it complete and corresponding with the current EITC curriculum it may contain inaccuracies and incomplete sections, subject to ongoing updates and corrections directly on the EITC webpage. No warranty is given by EITCI as a publisher in regard to completeness of the information contained within the document and neither shall EITCI be responsible or liable for any errors, omissions, inaccuracies, losses or damages whatsoever arising by virtue of such information or any instructions or advice contained within this publication. Changes in the document may be made by EITCI at its own discretion and at any time without notice, to maintain relevance of the self-learning material with the most current EITC curriculum. The self-learning preparatory material is provided by EITCI free of charge and does not constitute the paid certification service, the costs of which cover examination, certification and verification procedures, as well as related infrastructures.

TABLE OF CONTENTS

Introduction	5
Introduction to JavaScript	5
What is JavaScript	6
How dynamic webpages work	7
How JavaScript is executed	9
Dynamic vs weakly typed	10
JavaScript runs on a host environment	11
Programme outline	12
Java vs JavaScript	14
History of JavaScript	15
Getting started	17
Setting up development environment	17
Syntax and features	19
Project setup	20
Adding JavaScript to a website	21
Basic programming in JavaScript	22
Introduction to variables and constants	22
Declaring and defining variables	23
Working with variables and operators	25
Number and string data types	26
Using constants	28
More on strings	29
Functions in JavaScript	32
Introducing functions	32
Adding a custom function	34
Returning values in a function	36
Exploring the importance of code order	37
Introduction to global and local scope	38
Return statement	40
Executing functions indirectly	41
Type conversion	42
Splitting code and functions	43
Clickable buttons with event listeners	44
Advancing in JavaScript	45
Adding comments	45
More operators	46
More core data types	47
Arrays	48
Objects in JavaScript	49
Objects	49
Accessing object data	50
Adding a reusable function that uses objects	51
Undefined, Null, NaN	52
TypeOf	53
Script tags, defer, async	54
Summary	55
Debugging and efficient development	56
Introduction to debugging and efficient development	56
Overview of efficient development workflow	57
The IDE look and feel	58
Using shortcuts	59
Auto completion and hints	60
Extensions	61
Working with editor settings	62
Using different editor views	63
Finding help	64
Googling correctly for JavaScript hints	65

EUROPEAN IT CERTIFICATION CURRICULUM SELF-LEARNING PREPARATORY MATERIALS

Debugging overview	66
Working with error messages	67
Debugging logical errors with console.log	68
Chrome DevTools and breakpoints	69
Testing code changes in DevTools	70
Debugging vs Visual Studio Code	71
Summary	72

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: INTRODUCTION TO JAVASCRIPT**

JavaScript is a dynamic programming language that is compiled at runtime. It can be executed as part of a web page in a browser or directly on any machine in a host environment. JavaScript was created to make web pages more dynamic by allowing the content on a page to be changed directly from inside the browser. Originally called LiveScript, it was later renamed to JavaScript to resemble Java, although it is completely independent from Java and has nothing in common with it.

To understand JavaScript better, let's take a step back and look at how web pages work. As a user, when you visit a web page, you use a browser on your machine. The browser acts as a client and communicates with the server hosting the web page. The server sends the necessary HTML, CSS, and JavaScript files to the browser, which then renders the web page and displays it to you.

JavaScript plays a crucial role in enhancing the functionality and interactivity of web pages. It allows developers to add dynamic elements, such as animations, form validation, and interactive features, to web pages. With JavaScript, you can respond to user actions, manipulate the content of a page, and interact with external APIs to fetch data and update the page dynamically.

One of the key features of JavaScript is its weak typing system, which means that variables do not have a fixed data type and can be dynamically assigned different types of values. This flexibility allows for more fluid programming and makes JavaScript suitable for a wide range of tasks.

JavaScript is also known for its extensive ecosystem of libraries and frameworks, which provide developers with ready-to-use solutions for common tasks and help streamline the development process.

JavaScript is a dynamic programming language that is used to make web pages more interactive and dynamic. It can be executed in a browser or a host environment and allows developers to add functionality and interactivity to web pages. Despite its name, JavaScript is independent from Java and has its own unique features and capabilities.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: WHAT IS JAVASCRIPT**

JavaScript is a programming language that is commonly used in web development. It allows developers to make web pages more interactive and responsive. In order to understand how JavaScript works, it is important to first understand the basic flow of how web pages are loaded and interacted with.

When you enter a URL or click on a search result on a search engine like Google, a request is sent from your computer to a server on the internet where the web page is hosted. The server then loads the web page, which is typically in the form of an HTML file, and sends it back to your browser as a response. This is the basic process of loading a web page.

Once the web page is loaded, you can interact with it by clicking buttons, submitting forms, or performing other actions. For example, let's say you are on an online shop web page and you click a button to submit a form to order some products. This action triggers a new request, which is sent by your browser to the server to send the form submission data. The server handles this incoming request, possibly storing the order data in a database, and then sends back a new response in the form of a new web page. This new web page, such as an order confirmation page, is then sent back to your browser.

This is the traditional flow of how web pages work. However, JavaScript allows us to make this process more reactive and dynamic. Instead of always sending a new request and receiving a new response for every action, JavaScript allows us to change the already loaded web page and perform actions directly on it. This can make web pages feel more interactive and responsive.

To illustrate this concept, let's consider an example. You can find a simple project attached, which includes two folders (assets and info) and an index.html file. These additional folders and files are not relevant to the web page itself and can be ignored for now. The index.html file is the main file that you can open in a browser window by double-clicking it. This will display a dummy web page that contains some information about a course and the instructor.

The web page is primarily built using HTML and CSS at this point. If you are curious, you can explore the files in the assets folder, which contains images and CSS files. By right-clicking on the index.html file and opening it with a text editor, such as TextEdit on Mac or Notepad on Windows, you can view the raw HTML code of the web page.

JavaScript can be added to this web page to make it more interactive and dynamic. It allows you to manipulate the elements on the page, handle user interactions, and perform various actions without the need for a new request and response cycle.

JavaScript is a programming language that enhances the functionality of web pages by making them more reactive and dynamic. It allows developers to change the already loaded web page and perform actions directly on it, without the need for a new request and response. By understanding the basic flow of web pages and how JavaScript fits into it, you can begin to explore the possibilities of creating interactive and engaging web applications.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: HOW DYNAMIC WEBPAGES WORK**

In web development, JavaScript plays a crucial role in creating dynamic webpages. Unlike HTML and CSS, which are static, JavaScript allows for interactive and user-friendly experiences on websites. In this didactic material, we will explore how dynamic webpages work and the importance of JavaScript in achieving this.

A traditional webpage consists of HTML and CSS code. However, to create a more modern and user-friendly experience, we can use JavaScript to modify the existing webpage without loading a new page. This can be achieved by adding overlays or modals, which are small pop-up windows that appear on top of the existing page.

To demonstrate this concept, we have provided an HTML file and a JavaScript file called "app.js". The JavaScript code is stored in the scripts folder. While you may not fully understand the code at this point, we will gradually cover the basics throughout the course.

To implement JavaScript functionality, we need to add a script tag to the HTML file. Below the CSS import, add a new script tag with a src attribute pointing to the "app.js" file. This allows the browser to load and execute the JavaScript code.

Next, let's modify the links on the webpage. In the anchor tags for the "dynamic interpreted" and "weekly type" links, remove the existing link and replace it with a hash symbol (#). Add a new attribute called "data-text" and set it to the desired text for each link. The text can be copied from the corresponding HTML files in the info folder, excluding any line breaks. Additionally, add the class "info-modal" to both anchor tags.

After making these changes, save the HTML file and reload the webpage. Now, when you click on one of the links, a modal overlay will appear with the dynamically added content. You can close the overlay by clicking on the backdrop.

By using JavaScript, we can enhance webpages by dynamically adding content and providing a more interactive user experience. Throughout this course, you will gain a deeper understanding of JavaScript and its role in creating dynamic webpages.

JavaScript is a dynamic, weakly typed, and interpreted programming language that allows us to make web pages more dynamic. Unlike traditional web pages that require the user to navigate to a new page or wait for new HTML code to be downloaded, JavaScript enables us to change the existing page without any delays. This makes web pages more responsive, similar to mobile apps where users don't have to wait for actions to complete.

JavaScript is a hosted language, meaning it can run in different environments. In the example we just saw, it ran in the browser, but it can also run in other environments, which we will explore later in this course. The primary use case for JavaScript is running code in a browser on a web page to enhance its interactivity.

When we write JavaScript code and want it to have an effect on a web page, we rely on a JavaScript engine built into the environment where the code is executed. For example, in the Chrome browser, the engine is called V8, while in Firefox, it is called SpiderMonkey. Other browsers either use these engines or have their own. The JavaScript engine's role is to parse, or read and understand, the code. It then compiles the code on the fly into machine code, which executes faster. Finally, the engine executes the machine code, resulting in the desired effect on the web page.

Modern JavaScript engines have various optimizations to improve performance. For example, they may start executing the code before it is fully compiled, allowing for faster execution. They can also dynamically switch between compiled and uncompiled code to optimize performance further. These optimizations are implemented to ensure that JavaScript code runs as efficiently as possible.

JavaScript is a dynamic and versatile programming language that allows us to create dynamic web pages. It is interpreted on the fly, meaning it is compiled just before execution. JavaScript engines, such as V8 in Chrome,

play a crucial role in executing JavaScript code by parsing, compiling, and executing it. These engines also incorporate various optimizations to enhance performance.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: HOW JAVASCRIPT IS EXECUTED**

JavaScript is a dynamic interpreted programming language that is also weakly typed. Being dynamic and interpreted means that JavaScript is not pre-compiled like other languages such as C++. Instead, the code is evaluated and executed at runtime. This allows for flexibility in the code as it can be changed during execution. For example, you can switch the type of data stored in a variable dynamically.

In JavaScript, the code is compiled on the fly, meaning it is parsed, interpreted, and compiled at runtime. This allows for certain functionalities that are not possible in other programming languages. One such functionality is the ability to switch the type of data stored in a variable. For instance, you can start by storing text in a variable and later change it to a number.

JavaScript is also a weakly typed language, which means that data types are not strictly enforced. Unlike strongly typed languages, such as Java, JavaScript allows for more flexibility in data manipulation. However, this flexibility comes with potential risks and should be used judiciously.

It is important to note that JavaScript code execution occurs on a single thread in the browser. While modern machines have the capability for multi-threading, JavaScript code execution remains on a single thread. This concept of single threading will be further explored in the course, as it has implications for JavaScript development.

JavaScript is a dynamic interpreted programming language that is also weakly typed. It allows for on-the-fly compilation and interpretation of code, enabling the flexibility to change the code during runtime. JavaScript's weak typing allows for dynamic switching of data types in variables. However, it is important to use this flexibility responsibly. JavaScript code execution occurs on a single thread in the browser.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: DYNAMIC VS WEAKLY TYPED**

JavaScript is a programming language that is widely used in web development. One of its key characteristics is its dynamic and weakly typed nature. This means that when working with data in JavaScript, you don't have to explicitly define the data type. Instead, the data type is automatically inferred based on the value assigned to a variable. This dynamic nature allows for flexibility, as data types can change from one line of code to another.

In other programming languages, you typically have to declare the data type of a variable before using it. However, in JavaScript, you can simply store data in a variable without specifying its type. If the data happens to be a number, then it is treated as such. This forgiving nature of JavaScript allows for more flexibility and ease of use.

Furthermore, JavaScript is weakly typed, which means it doesn't enforce strong type definitions in advance. This means you can store different types of data in the same variable without any issues. JavaScript is able to handle multiple data types and perform operations on them accordingly.

It's important to note that this dynamic and weakly typed nature of JavaScript may seem unusual if you're coming from a background in other programming languages. However, it is a fundamental aspect of JavaScript and will become more clear and relevant as you progress through this course and work with data and variables.

JavaScript runs on a host environment, with the most well-known environment being the browser. Modern browsers have JavaScript engines built-in, allowing them to execute JavaScript code. However, JavaScript can also be run in other environments, such as on the server side. This means you can execute JavaScript code on a computer without the need for a browser.

JavaScript was originally designed to make websites more dynamic by allowing changes to be made without reloading the page. It can closely work together with HTML and CSS to manipulate the content and appearance of a website. JavaScript can also be used to send background HTTP requests and fetch data without the need for a page reload.

However, there are certain limitations to what JavaScript can do in the browser environment. For security reasons, JavaScript cannot access the local file system or interact with the operating system. This is to prevent malicious actions that could compromise user data or system integrity. JavaScript is confined to a sandbox within the browser, which provides certain capabilities while restricting others.

JavaScript is a dynamic and weakly typed programming language that is commonly used in web development. Its dynamic nature allows for flexibility in working with data, as data types are automatically inferred. JavaScript can be executed in various environments, with the browser being the most common. It enables dynamic and interactive web experiences, but is limited in its access to the local file system and operating system for security reasons.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: JAVASCRIPT RUNS ON A HOST ENVIRONMENT**

JavaScript runs on a host environment, and while the browser is the most common environment for JavaScript, it is not the only one. Google developed the JavaScript engine called V8, which was later extracted to run JavaScript outside of the browser. This standalone tool, known as Node.js, allows developers to execute JavaScript directly on their machines. In this course, we will have a separate module dedicated to Node.js, where we will explore its features and how it can be used to build web backends and servers.

However, for the majority of this course, we will focus on the browser side of JavaScript development. This is because JavaScript is essential for web development, and the browser provides a visual environment where we can see the changes and interactions on a web page. Node.js, on the other hand, can be executed on any machine and is often used for server-side JavaScript development.

Node.js provides access to the local file system, allowing developers to read and write files. It can also interact with the operating system. However, unlike browser-side JavaScript, it does not have direct access to manipulate HTML or CSS. In essence, browser-side JavaScript and Node.js have inverse access capabilities.

Throughout this course, we will primarily focus on the browser side of JavaScript development. The syntax and concepts you learn will be applicable in any environment where JavaScript is executed. However, learning in the browser environment provides more visual feedback and allows us to see the effects of our code. We will cover the separate module on Node.js later in the course, where we will explore the different capabilities and use cases of Node.js.

In this course, we will cover three main blocks of JavaScript development. The first block is the core basics, where we will learn the fundamental concepts of JavaScript and its history. We will then dive into the core syntax and language basics, including variables, functions, and control structures like if statements and loops. We will also explore efficient development and debugging techniques, configuring the development environment, and finding and fixing errors.

The second block builds upon the core basics and focuses on building a strong foundation in JavaScript. We will delve deeper into functions, a critical concept in JavaScript development. We will also explore the Document Object Model (DOM) basics, which involves manipulating the web page we are working on and adding content dynamically.

The third block covers advanced concepts that will take your JavaScript development skills to the next level. We will explore more complex topics like object-oriented programming, asynchronous programming, and working with APIs. These advanced concepts will enable you to become a proficient JavaScript developer.

By the end of this course, you will have a solid understanding of JavaScript and its core concepts, as well as the ability to build interactive web applications. You will have a strong foundation in JavaScript development and the knowledge to tackle more advanced topics.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: PROGRAMME OUTLINE**

In this programme, we will cover a comprehensive outline of JavaScript fundamentals. The goal is to provide you with a solid understanding of the JavaScript syntax and core features. Whether you are new to JavaScript or already familiar with the basics, the programme will serve as a great resource.

We will start by exploring how to manipulate web pages, including adding and removing content. We will also delve into arrays and iterables, which are lists of data, and learn how to work with them effectively. Additionally, we will focus on objects, which are crucial data structures in JavaScript.

Once we have covered the core syntax and features, we will build a stronger foundation by diving into more advanced concepts. One module in the curriculum will focus on classes and object-oriented programming, taking object usage to the next level. We will explore constructor functions and prototypes, which are important core concepts in JavaScript.

Our exploration will extend to the Document Object Model (DOM) and how to work with the browser. We will learn how to tap into certain browser features and delve deeper into events, such as reacting to mouse clicks. Functions will also be a major focus, as we will explore recursion and more advanced function usage.

To gain a deeper understanding of JavaScript, we will examine number and string types in greater detail, including their restrictions and capabilities. Asynchronous code will also be covered, addressing the issue of long-running operations without freezing the web page.

We will then move on to background HTTP requests, also known as AJAX, which allow for data exchange behind the scenes without reloading the page. This will further enhance our understanding of using JavaScript in the browser.

With a solid foundation in place, we will explore more advanced concepts. We will learn how to work with third-party libraries and utilize JavaScript modules to efficiently split code across multiple files. Additionally, we will delve into tooling, specifically webpack and babel, to optimize our JavaScript code.

Other topics covered in this programme include browser storage, ensuring browser support, JavaScript frameworks (with a focus on React.js), metaprogramming, Node.js, security considerations, deploying code to servers, performance optimizations, and fixing memory leaks.

JavaScript allows developers to add interactivity and dynamic content to websites. Throughout the programme, we will dive into various concepts and techniques that will enable you to become proficient in JavaScript.

Before we begin, it is important to understand the importance of debugging in the development process. Debugging involves identifying and resolving errors in your code. To aid in this process, we have provided code snippets that you can use to compare your code with the instructor's code. If you encounter any issues, it is recommended to compare your code step by step with the instructor's code to identify any differences. Additionally, utilizing search engines and the built-in search feature in the course player can help you find common issues and solutions. However, if you are unable to resolve an issue or if a concept is unclear, feel free to ask questions in the Q&A section of the course. The instructor regularly monitors and responds to questions to provide guidance. It is also encouraged to assist others in solving problems, as this promotes active learning and problem-solving skills.

Now, let's delve into the history of JavaScript and its relationship with Java. Despite the similar names, JavaScript and Java are entirely different programming languages with distinct syntax and principles. JavaScript primarily runs in web browsers, allowing for client-side scripting. On the other hand, Java can be used for server-side web development and other non-web development contexts. Java follows strict object-oriented and strongly typed principles, whereas JavaScript offers flexibility and supports weak typing. Throughout the course, we will explore object-oriented concepts in JavaScript to deepen your understanding.

The name "JavaScript" was chosen to capitalize on the popularity of Java, but the two languages share few

similarities. It is crucial to remember that Java does not run directly in web browsers, while JavaScript does. This differentiation between client-side and server-side JavaScript is significant. Client-side JavaScript executes within the browser, enabling dynamic interactions and modifications to web pages. Server-side JavaScript, with the assistance of Node.js, runs on a server and can generate dynamic HTML content for users. Understanding this distinction will help you leverage the appropriate language for different scenarios.

This programme will equip you with the fundamental knowledge and skills required to become proficient in JavaScript.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: JAVA VS JAVASCRIPT**

JavaScript began in the mid-1990s as a client-side scripting language for web browsers. It allowed developers to manipulate web pages and interact with browser features through JavaScript engines provided by different browser vendors. For example, the V8 engine is used by Chrome.

However, the idea of JavaScript being limited to the browser led to the development of Node.js. Node.js is a runtime environment that allows JavaScript to be run on the server-side, outside of the browser. This means that developers can now use JavaScript for various tasks, such as working with the file system or handling incoming HTTP requests.

The syntax, concepts, and core features of JavaScript remain the same whether it is used on the client-side or the server-side. Therefore, what you learn about JavaScript in this course, which primarily focuses on the browser-side, can be applied to Node.js and other environments as well.

Learning JavaScript is essential for front-end web development because it is the only programming language that can be used in the browser. Once you have a solid understanding of JavaScript, you can easily transfer your knowledge to Node.js since they share the same syntax.

Understanding JavaScript's history is also important to see how it has evolved over time. In 1995, Netscape introduced LiveScript, which was later renamed JavaScript. Microsoft also released its own version of JavaScript for Internet Explorer in 1996. Although they had the same idea and generally the same syntax, there were also differences between the two versions.

JavaScript faced challenges in terms of fragmentation and limited capabilities. Different browsers required different scripts, and JavaScript was mainly used for spammy and annoying overlays and pop-ups. To address these issues, JavaScript was submitted to the ECMA committee for standardization in late 1996. The ECMA committee is responsible for standardizing JavaScript, ensuring that there is one standard that can be implemented by multiple browsers.

Standardization efforts continued until around 2005, with Microsoft eventually joining the process and supporting the standardized JavaScript version. Although there were still some differences, JavaScript became more commonly used with a more consistent approach.

Since then, there have been ongoing standardization efforts and significant progress in adding new features to JavaScript. Microsoft's involvement in the standardization process has contributed to the growth and improvement of the language.

JavaScript started as a client-side scripting language for web browsers and has evolved to be used on the server-side through Node.js. Learning JavaScript is crucial for front-end web development, and the knowledge gained can be easily transferred to other environments. Understanding JavaScript's history helps to appreciate its growth and standardization efforts.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: INTRODUCTION****TOPIC: HISTORY OF JAVASCRIPT**

JavaScript is a widely used programming language that is under active development and constantly evolving. It has come a long way since its inception and has become a standardized language that can be used across different browsers with some minor differences. In this course, we will focus on the most modern syntax of JavaScript and learn all the latest features.

The development of JavaScript has been ongoing, with significant improvements made since 2010-2011. New features are continuously being added, making it a better programming language. In the past, JavaScript was considered clunky and each browser had its own way of interpreting the language, despite standardization efforts. However, nowadays we have a more uniform language with a core set of features that can be used across different browsers. Although there are still some differences, we now have a standardized language that is great to use.

The organization responsible for managing the language is called Ecma International. They oversee the development of a language called ECMAScript, which is the actual language evolved by the organization. JavaScript, on the other hand, is the most famous implementation of ECMAScript and is implemented by browser vendors. Other implementations include ActionScript and JScript, but they are not as widely used as JavaScript.

Each browser comes with its own JavaScript engine, which determines the exact features supported. Browser vendors implement the new standard versions of ECMAScript into their JavaScript engines. While discussions may still be ongoing about the final implementation details, browser vendors may already implement certain features in their engines. This means that some browsers may support certain features earlier than others. We will discuss this further in the browser support module, where you will learn how to write next-generation JavaScript code that runs in all browsers.

JavaScript is constantly evolving, and browser vendors play a significant role in shaping its future. They contribute to the development of ECMAScript and influence how JavaScript looks and what new features are added. This active development ensures that JavaScript remains a powerful and up-to-date language.

In this course, we will focus on teaching you JavaScript using the most modern syntax. In 2015-2016, JavaScript underwent a major overhaul, introducing important changes. Many tutorials and courses still teach the old JavaScript syntax, which is still functional but has certain downsides. We will start with the modern syntax from the beginning and stick to it throughout the course. However, we will also cover the older syntax so that you can understand code snippets and tutorials that use it.

Before we begin, it is important to set up a proper development environment. Writing code in a plain text editor can be cumbersome and not very efficient. We recommend using Visual Studio Code as your code editor. It is free, fast, modern, and highly customizable. Visual Studio Code also has a wide range of extensions that can enhance your coding experience. We will install it together in just a moment.

Having a good development environment is crucial for writing and testing JavaScript code. In addition to Visual Studio Code, we will also explore how to test our web pages and code effectively.

JavaScript is a fundamental programming language used in web development. In this didactic material, we will explore the history of JavaScript and its importance in the field.

JavaScript was created by Brendan Eich at Netscape Communications in 1995. Originally known as LiveScript, it was later renamed to JavaScript to leverage the popularity of Java at the time. Despite the name similarity, JavaScript and Java are two distinct programming languages.

JavaScript was initially developed to provide interactivity and dynamic features to web pages. Before its introduction, web pages were static and lacked the ability to respond to user actions. With JavaScript, developers could add interactive elements, validate forms, and create dynamic content.

The introduction of JavaScript revolutionized web development, allowing developers to build more engaging and

user-friendly websites. Over the years, JavaScript has evolved and become a versatile language, powering not only web pages but also server-side applications, mobile apps, and even desktop applications.

When it comes to testing and running JavaScript code, a browser is required. While JavaScript can run in all modern browsers, Google Chrome is highly recommended for development purposes. Chrome offers a wide range of developer tools and features that facilitate the debugging and testing process. It provides an excellent environment for web development, making it the preferred choice for many developers.

JavaScript has a rich history and has played a crucial role in shaping the modern web. Its ability to add interactivity and dynamic features to web pages has made it an essential tool for web developers. Google Chrome is a recommended browser for JavaScript development due to its extensive developer features.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: GETTING STARTED****TOPIC: SETTING UP DEVELOPMENT ENVIRONMENT**

To get started with setting up your development environment for JavaScript, it is recommended to use Google Chrome as your browser, even if you typically use a different one. This will ensure that you have the same results as demonstrated in this material. Additionally, we will be using the developer tools built into Chrome to inspect and debug our code.

To begin, we will need a code editor or an Integrated Development Environment (IDE). One popular option is Visual Studio Code, which can be found at code.visualstudio.com. On this website, you can learn more about the features of Visual Studio Code and access the documentation for further exploration. To install Visual Studio Code, simply download the installer for your operating system (Mac, Windows, or Linux) from the website. The installer will guide you through a straightforward installation process.

Once you have installed Visual Studio Code, open the editor. The interface should resemble the screenshot provided in this material. If the appearance is different, you can change the theme by going to "Preferences" or using the shortcut provided. From the "Preferences" menu, select "Color Theme" and choose a theme from the available options. You can cycle through the themes using the arrow keys to find the one you prefer. Additionally, you can customize the elements displayed in the editor's interface by going to "View" and selecting "Appearance." For example, you can remove the status bar at the bottom if desired.

By default, the editor does not display any code. To open a project folder, click on "Open Folder" or choose "File" followed by "Open." Select the folder containing the starting code provided in this material. The project will be loaded into Visual Studio Code, and you should see the files and folders within the project. The colors of the files may appear different due to version control, which is not relevant at this stage.

The icons displayed in the editor's interface can be customized using extensions. Extensions are plugins that enhance the functionality of the editor. To access the extensions, click on "View" and select "Extensions." In the extensions view, you can search for and install various extensions to modify the editor's appearance and behavior. Two recommended extensions are the Material Icon Theme and Prettier. The Material Icon Theme provides consistent file icons, while Prettier helps with code formatting. Installing these extensions is optional but can enhance your development experience.

To configure your project and understand code formatting, open the "assets" folder and navigate to the "scripts" folder. Inside the "scripts" folder, open the "app.js" file. This file contains a script that will be used later in the material.

To set up your development environment for JavaScript, it is recommended to use Google Chrome as your browser and Visual Studio Code as your code editor. Install Visual Studio Code, customize its appearance using themes and extensions, and open the project folder containing the provided starting code. Familiarize yourself with the editor's interface and the script file within the project.

A code editor is essential for web development as it makes code more readable and provides helpful features like keyword highlighting and auto completion. While humans may find certain code formats difficult to read, JavaScript can still understand and execute them. To improve code readability, we can use auto formatting tools like Prettier.

To autoformat your code in Visual Studio Code, you can go to Code Preferences and search for "format document." Assign a shortcut key to this command, and you can use it to automatically format your code. Prettier will transform the code into a more readable version.

Visual Studio Code also allows you to customize keyboard shortcuts for various commands, not just formatting. You can refer to the official documentation for a comprehensive list of available shortcuts. Additionally, in the development and debugging section of this course, we will share useful hints to enhance your coding experience.

It is recommended to explore the preferences settings in Visual Studio Code. You can switch between user and

workspace settings. User settings apply to all projects managed with Visual Studio Code, while workspace settings only apply to the current project. By accessing preferences settings, you can configure various options, such as Prettier, to format your code according to your preferences.

In Chrome, it is recommended to use the incognito mode to avoid any distractions from installed extensions that might interfere with your web page. Chrome's developer tools are essential for web development and can be accessed by clicking on "View" and then selecting "Developer" and "Developer Tools." These tools provide various functionalities, such as the console for outputting development log messages and the elements tab for viewing the rendered HTML content.

To ensure consistent output and behavior in the console area of the developer tools, it is important to uncheck the "preserve log" option. This ensures that the log is emptied whenever the page is reloaded, providing a fresh log output for the currently running page.

Now that we have set up our code editor and familiarized ourselves with Chrome's developer tools, we are ready to dive into JavaScript. In this course module, we will explore the core syntax of JavaScript, including how to write valid JavaScript code. We will also cover variables, their creation, usage, and the types of data we can work with. Additionally, we will delve into operators, which allow us to perform calculations and execute code by combining values. Lastly, we will learn about functions, an important language construct used in JavaScript programs.

With a properly set up editor and a browser like Chrome, equipped with the necessary developer tools, we are all set to embark on our journey into JavaScript. Let's dive right into its core syntax and essential features, laying the foundation for our exploration of this amazing language.

In web development, JavaScript is a widely-used programming language that allows us to add interactivity and dynamic functionality to websites. In this didactic material, we will focus on the fundamentals of JavaScript, specifically on getting started and setting up the development environment.

Functions are an essential part of JavaScript, as they allow us to group and organize blocks of code that perform specific tasks. They are reusable and can be called multiple times within a program. In this material, we will explore what functions are, why we use them, and how to use them effectively.

To demonstrate the concepts we discuss, we will use a basic application as a context. This application will provide a visual representation on the screen, which we believe enhances the learning experience. We will be using a website for this purpose, as websites are inherently visual and interactive.

To begin, we need to set up a basic website where we can utilize JavaScript. To facilitate this, we have provided a starting project with the necessary code. You can download and extract the starting code, and then incorporate it into your development environment.

By following the instructions in this material, you will gain a solid understanding of the fundamental concepts of JavaScript and how to set up a development environment for web development. This knowledge will serve as a strong foundation as you continue to explore and expand your skills in JavaScript.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: GETTING STARTED****TOPIC: SYNTAX AND FEATURES**

To get started with JavaScript development, you will need to download and extract the provided starting code. Once extracted, open the folder using your preferred IDE or code editor. In this example, we will be using Visual Studio Code.

After opening the folder, you will see a folder structure that includes an "index.html" file. This file serves as the entry point for our JavaScript code.

It is important to note that the ".vscode" folder may not be present in your setup. This folder is created by Visual Studio Code to manage some settings, such as the zoom level for better code visibility. It is not a requirement for JavaScript development.

Additionally, there is a "gitignore" file included in the folder. This file is used when working with Git, a source control management system. However, it is not necessary for following along with JavaScript development. You can skip this file if you are not familiar with Git.

Now that you have the starting code set up, you are ready to begin writing JavaScript code and exploring its syntax and features. Feel free to refer to the provided code and experiment with different JavaScript concepts.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: GETTING STARTED****TOPIC: PROJECT SETUP**

In this module, we will be getting started with JavaScript fundamentals, specifically focusing on project setup. To begin, we have an index.html file that contains a basic HTML skeleton. It includes links to fonts and a style sheet located in the assets folder. Within the script section of the HTML file, there is a starting script that includes code we may not fully understand at this point. However, it is necessary for us to see something on the screen and not just have a static application.

To view the interface, you can double click on the index.html file in your file explorer (Windows Explorer or Mac Finder), which will open a new window in your browser. This will display a basic calculator interface, which will be the focus of our project throughout this module.

Currently, nothing happens when you type and hit enter because we haven't written the necessary logic yet. We will be diving into the basic features of JavaScript to build this calculator and add the necessary functionality. JavaScript is essential for adding dynamic features to our browser-based application.

To begin building the calculator, we need to write some JavaScript code that runs in the background when the page is loaded. This code will allow us to interact with the calculator when the user presses buttons and update the result accordingly. As mentioned earlier, we have a vendor.js file provided, but we won't be using it. Instead, we will create a new file named app.js, which will hold our calculator logic.

In app.js, we will start with a simple alert that says "This works." Although you may not fully understand the syntax at this point, we will explore it throughout this module. After adding the alert, reload the page and you will see that nothing happens. This is because the browser does not automatically scan all subfolders for scripts to execute. This is actually a good thing as it allows us to control when our scripts run.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: GETTING STARTED****TOPIC: ADDING JAVASCRIPT TO A WEBSITE**

To add JavaScript to a website, there are a couple of ways to import JavaScript into the HTML file. One way is to add a script tag in the head section of the HTML file. Between the opening and closing script tags, you can write JavaScript code. For example, you can use the alert function to display a message. However, if you include all your scripts in the HTML file, it can become difficult to manage as the file size increases.

A better approach is to import your script using the script tag's source attribute (src). By adding the src attribute with the location of your script file, you can separate your JavaScript code from the HTML file. For example, you can specify the location of your script file in the assets folder by adding "assets/scripts/app.js" as the src attribute. It's important to note that the script tag should have a separate opening and closing tag, unlike the self-closing tag used for other elements.

When importing the script, it's recommended to place the script tag at the end of the body section. This allows the browser to first render all the HTML content before executing the JavaScript code. By doing this, you prevent the script from blocking the rendering of the page. This way, the majority of the file is loaded before the script executes.

If your script depends on another script, it's important to import the dependent script first. In this case, you should import the "vendor.js" file before the "app.js" file. The order of importing scripts matters in JavaScript. By importing the dependent script first, you ensure that the code in the "app.js" file can work with the code in the "vendor.js" file.

Once you have imported the necessary scripts, you can start writing JavaScript code. In this example, the goal is to create a basic calculator where you can enter a number and add it to the last number entered. To achieve this, you will need to define variables and write the necessary logic for the calculator.

To add JavaScript to a website, you can import the script using the script tag's src attribute. It's recommended to place the script tag at the end of the body section to prevent blocking the rendering of the page. Import any dependent scripts before using them in your code. Finally, write the JavaScript code to achieve the desired functionality.

Variables and constants are core features in JavaScript, as well as in most programming languages. They serve as data containers or data storage in programs. For instance, in a calculator program, we may need to store the last result and the number entered by the user to derive the new result.

To define a variable in JavaScript, we use the "let" keyword. This keyword is followed by the variable name, which can be chosen by the programmer.

Variables can hold different types of data, such as numbers, strings, or boolean values. The data stored in a variable can be changed throughout the program execution.

In addition to variables, JavaScript also supports constants. Constants are similar to variables, but their value cannot be changed once it is assigned. To declare a constant, we use the "const" keyword.

Both variables and constants are useful for storing and manipulating data in JavaScript programs. They allow us to work with user input, perform calculations, and store results.

Variables and constants are fundamental concepts in JavaScript programming, enabling us to work with data efficiently and effectively.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: BASIC PROGRAMMING IN JAVASCRIPT****TOPIC: INTRODUCTION TO VARIABLES AND CONSTANTS**

In JavaScript, variables are created using the "let" keyword, followed by a chosen name and an equal sign, and then the desired value. Once a variable is created, it can be reassigned with a new value using only the name and the equal sign. The "let" keyword is only necessary when introducing a variable for the first time; subsequent assignments can be made using only the name.

A variable is a data container that can hold a value which may change over time. However, there is also a special form of variable called a constant. Constants are created using the "const" keyword instead of "let". Unlike variables created with "let", the value of a constant cannot be changed once it is assigned. If an attempt is made to change the value of a constant, an error will occur. Constants are useful for storing values that never change throughout the program and need to be initialized in a central place. By using constants, you can ensure that the same value is referenced throughout the code, making it easier to update if needed.

When creating variables or constants, there are certain rules and recommendations to follow. The name should start with a lowercase letter and follow the camel case convention, where each word within the name starts with an uppercase letter. This makes the name more readable and descriptive. It is also important to note that JavaScript is case-sensitive, so the capitalization of letters matters.

Variables are created using the "let" keyword, while constants are created using the "const" keyword. Variables can be reassigned with new values, while constants cannot be changed once assigned. Using meaningful and descriptive names for variables and constants is recommended, following the camel case convention.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: BASIC PROGRAMMING IN JAVASCRIPT****TOPIC: DECLARING AND DEFINING VARIABLES**

In JavaScript, declaring and defining variables is an essential part of programming. The way you write the variable name, including the casing, is important and can affect how the variable is interpreted. For example, "username" with a lowercase "u" is different from "Username" with an uppercase "U".

Variable names can be made up of letters, digits, and special characters like the dollar sign (\$) or underscore (_). The dollar sign and underscore can be used anywhere in the variable name, including at the beginning. However, it's important to note that variable names cannot start with a digit.

There are some naming conventions to follow when it comes to variables. It's recommended to use camel case notation, where the variable name starts with a lowercase letter and each subsequent word starts with an uppercase letter. For example, "myVariableName".

On the other hand, there is a convention called snake case, where words are separated by underscores. While technically valid in other programming languages, snake case is not recommended in JavaScript.

It's also important to avoid using special characters like dashes or white spaces in variable names. Only the underscore is allowed, as mentioned earlier.

Furthermore, using keywords such as "let" and "const" as variable names is not allowed. JavaScript would not be able to differentiate between the keyword and the variable name, leading to confusion.

When declaring a variable, you can choose to initialize it with a value or leave it uninitialized. An uninitialized variable is declared but does not have a value assigned to it. However, it's common practice to initialize variables with an initial value.

To conclude, when writing JavaScript code, you have the choice to end each line with a semicolon or not. While semicolons are generally optional, it's recommended to use them for consistency and to prepare for other programming languages where they are mandatory. However, some developers choose to omit semicolons in JavaScript code. Ultimately, the decision is up to the programmer.

In JavaScript, declaring and defining variables is a fundamental concept that allows us to store and manipulate data. When declaring a variable, we use the keyword "let" followed by the variable name. It's important to note that JavaScript is a case-sensitive language, so "myVariable" and "myvariable" would be considered two different variables.

To define a variable, we use the assignment operator "=" followed by the desired value. For example, we can declare and define a variable called "currentResult" and assign it the value of 10:

```
1. let currentResult = 10;
```

Once a variable is declared and defined, we can perform operations on it. In this case, we want to add 10 to the current value of "currentResult" and store the result back in the variable. We can achieve this by using the addition operator "+":

```
1. currentResult = currentResult + 10;
```

This line of code takes the current value of "currentResult" (which is 10) and adds 10 to it, resulting in 20. The new value is then stored back in the "currentResult" variable.

JavaScript provides various mathematical operators that allow us to perform different operations on variables. These include:

- Addition (+): Used to add two numbers together.
- Subtraction (-): Used to subtract one number from another.

- Multiplication (*): Used to multiply two numbers.
- Division (/): Used to divide one number by another.
- Modulus (%): Used to find the remainder of a division operation.

For example, if we have two variables "a" and "b" with values 5 and 2 respectively, we can perform different operations:

1.	let a = 5;
2.	let b = 2;
3.	
4.	let sum = a + b; // sum = 7
5.	let difference = a - b; // difference = 3
6.	let product = a * b; // product = 10
7.	let quotient = a / b; // quotient = 2.5
8.	let remainder = a % b; // remainder = 1

In the above example, "sum" stores the result of adding "a" and "b", "difference" stores the result of subtracting "b" from "a", "product" stores the result of multiplying "a" and "b", "quotient" stores the result of dividing "a" by "b", and "remainder" stores the remainder after dividing "a" by "b".

It's worth noting that JavaScript also supports other types of variables, such as strings, booleans, and objects. However, in this particular lesson, we focused on declaring and defining variables for numerical values.

Declaring and defining variables in JavaScript allows us to store and manipulate data. We can use various mathematical operators to perform operations on variables, such as addition, subtraction, multiplication, division, and finding the remainder of a division operation.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: BASIC PROGRAMMING IN JAVASCRIPT****TOPIC: WORKING WITH VARIABLES AND OPERATORS**

JavaScript Fundamentals - Basic programming in JavaScript - Working with variables and operators

In JavaScript, we can work with variables and operators to perform various mathematical operations. The mathematical operators in JavaScript include addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (**). These operators allow us to perform basic arithmetic calculations.

Additionally, JavaScript also has other operators, such as the assignment operator (=), which is used to assign a value to a variable. For example, we can use the assignment operator to assign a value to a variable named "currentResult".

When using operators in JavaScript, it's important to understand the order of execution. JavaScript follows the standard mathematical rules, where operations inside parentheses are executed first, followed by multiplication and division, and finally addition and subtraction.

To output the result of a calculation, we can use a function called "outputResult", which is provided in a separate file called "vendor.js". This function takes the value we want to output as an argument. By calling this function and passing in the "currentResult" variable, we can display the result on our HTML page.

In addition to basic mathematical operations, JavaScript allows us to perform more complex calculations by using parentheses to group different mathematical calculations together. This allows us to control the order of execution and achieve the desired result.

In JavaScript, variables can store various types of values, not just numbers. We can work with strings, booleans, objects, and more. However, in this lesson, we focused on working with numbers.

The "outputResult" function mentioned earlier is a type of function called a "callback function". It triggers a piece of code defined in the "vendor.js" file, which updates specific parts of our HTML code with the data we provide. This allows us to dynamically display the result of our calculations on the webpage.

By understanding how to work with variables and operators in JavaScript, we can perform a wide range of mathematical operations and display the results on our webpages.

JavaScript Fundamentals - Basic programming in JavaScript - Working with variables and operators

In JavaScript, there are different data types available for use. One of the basic data types is numbers, which can be positive or negative, and can also include decimal places. Numbers with decimal places are called floats or floating point numbers, while numbers without decimal places are called integers. Numbers are commonly used in code for calculations or when working with numerical values.

Another important data type in JavaScript is text, also known as strings. Strings are used when working with text-based data, such as user names or displaying messages to users. Strings can be created by enclosing the text in single quotes, double quotes, or backticks. All three options are valid, but it is important to be consistent in your choice throughout your code.

It is worth noting that backticks have a special purpose, which will be explained later. For now, it is recommended to use either single quotes or double quotes and maintain consistency.

When working with JavaScript, it is common to use a combination of numbers and strings to perform various tasks. For example, you might need to calculate a value based on user input or work with prices. By understanding how to work with numbers and strings, you can effectively manipulate and display data in your JavaScript code.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: BASIC PROGRAMMING IN JAVASCRIPT****TOPIC: NUMBER AND STRING DATA TYPES**

In JavaScript, there are two basic data types: numbers and strings. Numbers are used for mathematical calculations, while strings are used for text. In this program, we have an example of both types.

We start with a number, followed by a bunch of numbers. The thing labeled as "this" is actually a string, specifically an empty string. In JavaScript, strings can be enclosed in either single quotes or double quotes. In this case, the empty string is enclosed in single quotes.

We have an output result function that writes to two places on a web page. However, we are only interested in writing to one of the places, so we tell the function to write nothing to the other place.

To write to the other place, we can create a new variable, let's call it "calculation description". This variable will store a description of the calculation we just ran. The description includes a mathematical equation: $0 + 10 * 3 / 2 - 1$. However, it's important to note that JavaScript treats this equation as text, not as a mathematical operation. Therefore, JavaScript won't try to execute it as a mathematical equation.

We can pass the "calculation description" variable to the output result function, which will display the text on the web page. When the page is reloaded, the text will appear as entered, without any mathematical calculations being performed.

If we want to include the value of another variable, such as "current result", in the text, we need to use string concatenation. String concatenation is the process of combining multiple strings into one longer string. In this case, we can use the plus operator to concatenate the strings.

For example, we can concatenate the opening parenthesis, the value of "current result", and the closing parenthesis to create a longer string. To indicate that we want to use the value of "current result" and not treat it as text, we don't enclose it in quotes.

It's important to note that the plus operator behaves differently when used with numbers and strings. When used with numbers, it performs mathematical addition. When used with strings, it concatenates them.

To ensure readability of the final text, it's recommended to add whitespace between the plus operator and the value stored in "current result".

When reloading the page, the output may not be what we expect. This is because the value of "current result" has changed since it was first initialized. JavaScript executes code from top to bottom, so if we use "current result" after calculating a new value, we will get the updated value.

To print the initial value of "current result" (in this case, zero), we need to either execute the code before calculating a new value or store the initial value in another variable.

One way to store the initial value is by using a constant. Constants are variables whose values cannot be changed once they are assigned. By using a constant, we can ensure that the initial value of "current result" remains the same throughout the program.

JavaScript has two basic data types: numbers and strings. Numbers are used for mathematical calculations, while strings are used for text. We can use the output result function to display text on a web page. String concatenation allows us to combine multiple strings into one longer string. When using variables in strings, we need to be aware of their values at the time of concatenation.

In JavaScript, there are different data types that can be used to store and manipulate values. Two of the basic data types are numbers and strings.

Numbers in JavaScript can be integers or decimals. They are used to perform mathematical operations such as addition, subtraction, multiplication, and division. To declare a number, you can simply assign a value to a

variable using the "const" keyword. For example, `const number = 5;` declares a constant variable named "number" with a value of 5.

Strings, on the other hand, are used to represent text. They are enclosed in single or double quotation marks. To declare a string, you can assign a sequence of characters to a variable. For example, `const message = 'Hello, world!';` declares a constant variable named "message" with a value of "Hello, world!".

In JavaScript, you can also initialize a variable with the value of another variable or constant. This means that you can assign the value of one variable to another variable. For example, `const result = defaultResult;` assigns the value of the variable "defaultResult" to the variable "result".

It is important to note that when declaring a constant variable using the "const" keyword, the value assigned to it cannot be changed later in the program. This ensures that the variable remains constant throughout the execution of the program.

To summarize, in JavaScript, numbers and strings are fundamental data types used to store and manipulate values. Numbers are used for mathematical operations, while strings are used to represent text. Variables can be initialized with the value of another variable or constant. Constants, declared using the "const" keyword, cannot be changed after they are assigned a value.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: BASIC PROGRAMMING IN JAVASCRIPT****TOPIC: USING CONSTANTS**

In JavaScript, constants are used to store values that cannot be changed after they are declared. This is different from variables, which can be reassigned with new values. In this module, we will explore the concept of constants and how they can be used in JavaScript.

To declare a constant, we use the keyword 'const' followed by the name of the constant and its initial value. For example, 'const defaultResult = 0;' declares a constant named 'defaultResult' with an initial value of 0.

Unlike variables, constants cannot be reassigned. If we try to change the value of a constant, the browser will throw an error. However, we can still use the value of a constant in our code without modifying it.

In JavaScript, we can also use variables to store values that can be changed. To declare a variable, we use the keyword 'let' followed by the name of the variable and its initial value. For example, 'let currentResult = defaultResult;' declares a variable named 'currentResult' and assigns it the value of the 'defaultResult' constant.

In the provided code, we can see an example of using constants and variables together. The 'defaultResult' constant is used to initialize the 'currentResult' variable. We can refer to the value of the constant in our code without changing it. This allows us to build a text representation of our calculation using the constant value.

It's important to note that when we assign the value of a constant to a variable, we are creating a copy of the value. If we modify the variable, it does not affect the original constant. The constant and its value remain unchanged.

When working with strings in JavaScript, we can use either single quotes or double quotes to define a string. The choice is up to the developer's preference, but it's important to be consistent throughout the code. Mixing quotes within a string is not allowed and will result in an error.

In the provided code, there is an example of an unterminated string literal. The string is opened with a double quote but never closed, making it invalid JavaScript code. The browser will throw an error if we try to run this code.

To fix the error, we need to close the string with the same type of quote that we used to open it. Once the string is properly closed, the code will run without errors.

Constants are used to store values that cannot be changed, while variables can be reassigned with new values. When using constants, it's important to understand that assigning their value to a variable creates a copy and does not modify the original constant. Additionally, when working with strings, it's crucial to use consistent quotes and properly close the string to avoid errors.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: BASIC PROGRAMMING IN JAVASCRIPT****TOPIC: MORE ON STRINGS**

In JavaScript, strings are a fundamental data type used to represent text. In this lesson, we will explore more about strings and learn how to work with them effectively.

When working with strings, it is important to remember that they must be enclosed in either single quotes (') or double quotes ("). This is necessary to differentiate strings from other types of data. For example, if we want to output a single quote character within a string, we can enclose the string in double quotes and include the single quote character within it. This way, JavaScript will treat everything within the double quotes as text.

It is worth noting that using a single quote within a single quote or a double quote within a double quote will result in invalid JavaScript code. Therefore, it is recommended to use double quotes when you need to output a single quote character within a string.

In addition to single and double quotes, JavaScript also provides an alternative way to create strings using backticks (`). Backticks allow us to create template literals, which are strings that can span multiple lines and include dynamic values. To create a template literal, we enclose the string within backticks instead of single or double quotes.

Template literals offer a convenient way to embed dynamic values within a string using the `\${}` syntax. Inside the `\${}`, we can include variables, constants, or expressions that evaluate to a value. JavaScript will replace the `\${}` with the actual value when the string is evaluated.

For example, if we have a constant named `defaultResult` with a value of 0, we can use a template literal to include its value within a string. By using `\${defaultResult}`, JavaScript will replace `\${defaultResult}` with the value of `defaultResult` when the string is evaluated.

It's important to note that template literals only work with backticks. If we use single or double quotes, the `\${}` syntax will be treated as regular text and will not be replaced with the actual value.

Template literals, with their ability to include dynamic values and span multiple lines, are a powerful feature of JavaScript. They eliminate the need for manual string concatenation using the plus operator, making our code shorter and more readable.

When working with strings in JavaScript, we can use single quotes, double quotes, or backticks. Single and double quotes are used to create regular strings, while backticks are used to create template literals. Template literals allow us to include dynamic values within a string using the `\${}` syntax.

In JavaScript, strings are a fundamental data type used to represent text. In this lesson, we will explore some additional features and techniques related to working with strings.

One interesting feature of JavaScript strings is the ability to include line breaks and extra white space within the string itself. This can be achieved using template literals, which are enclosed in backticks (` `). Template literals allow for the inclusion of line breaks and extra white space by simply adding them within the string. This can improve readability, especially when dealing with long strings.

However, it's important to note that the inclusion of line breaks and extra white space is not purely visual. These line breaks and white spaces are actually part of the string itself. Depending on how the string is rendered and styled, the line breaks and white spaces may or may not be visible on the web page.

In contrast, when using normal strings with single or double quotes, line breaks and extra white space are not allowed within the string itself. If you want to split a normal string across multiple lines, you have two options. You can close the string with a single quote, concatenate it with a plus operator, and then add a line break. Alternatively, you can use a special character combination, `\\n`, to represent a line break within the string.

It's important to understand that line breaks and extra white space within normal strings are primarily for

developer readability and do not affect the actual string being constructed. When the string is rendered on a web page, it will not include the line breaks or extra white space.

To demonstrate the usage of line breaks within strings, let's consider an example of an error message. In this case, we can use a normal string with single or double quotes and add a line break using the ``\n`` character combination. This will result in a line break when the string is rendered.

JavaScript provides various ways to work with strings, including the ability to include line breaks and extra white space within template literals. However, it's important to understand that line breaks within normal strings are represented using the ``\n`` character combination. These features can be useful for improving code readability, but it's crucial to consider how the string will be rendered and styled on a web page.

In JavaScript, strings are a fundamental data type used to represent text. In this context, we will explore some important concepts related to strings.

A string is enclosed in either single quotes (') or double quotes ("). For example, 'hello' and "world" are both valid strings. However, it is important to note that if a string is enclosed in single quotes, you must use a backslash (\) to escape any single quotes within the string. The same applies to double quotes if the string is enclosed in double quotes.

Using a backslash in a string has a special meaning - it escapes the character that comes after it. This means that the character after the backslash is not treated as a normal character with its usual meaning, but instead, it is combined with the backslash to have a special meaning. For example, '\n' represents a line break, where the backslash and 'n' combined create a new line.

Another important combination is a backslash followed by a single quote (\'). This is helpful if you want to output a single quote within a string that is enclosed by single quotes. Without the backslash, you would encounter a syntax error because the single quote would be interpreted as an attempt to open a new string.

Alternatively, you can use double quotes to surround the string instead of single quotes. In this case, you can use single quotes within the double-quoted string without any issues or the need to escape them. This is because the double quote character is used to open and close the string, and JavaScript does not interpret a single quote as a closing character.

To output a backslash within a string, you need to escape it as well. A single backslash informs JavaScript that the character after it has a special meaning and should be escaped. To output a single backslash, you need to put two backslashes in front of it (\\). This tells JavaScript that you want to output a single backslash as part of the string.

In addition to these escape sequences, JavaScript provides a feature called template literals. Template literals allow you to interpolate content within a string using backticks (`). This feature is useful for easily adding line breaks and white space within a string.

It is important to note that you do not need to memorize all the escape sequences. However, it is useful to be aware of the most commonly used ones. These include '\n' for a line break, '\\' for a backslash, and '\'' for a single quote. Attached is a list of escapable characters that you may find useful.

Understanding these concepts is essential as you will encounter strings throughout your career as a web developer. Being aware of these escape sequences and the template literal feature will allow you to manipulate and output strings effectively.

In JavaScript, functions can have parameters and can also return values. Parameters are variables that are passed into a function, allowing the function to work with different values each time it is called. In the given example, "name" is a parameter of the function.

Additionally, functions in JavaScript can return values. This means that after performing some operations or calculations, a function can provide a result that can be used elsewhere in the code. However, the specific function mentioned in the example does not return anything.

Returning values from functions becomes particularly useful when we want to perform a specific task and obtain a result that we can use in our program. By returning values, we can store and manipulate the results of functions, making our code more dynamic and flexible.

In this module, we will explore the concept of returning values from functions in greater detail. Through practical examples and exercises, we will see how returning values can enhance the functionality and versatility of our JavaScript programs.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: INTRODUCING FUNCTIONS**

In JavaScript, functions play a crucial role in organizing and structuring our code. They allow us to define reusable blocks of code that can be executed whenever we need them. Functions also provide us with the flexibility to run code at specific times or in response to certain events.

When we define a function, JavaScript registers it and stores it in memory. The code inside the function does not run immediately when the script is executed. Instead, we can call the function later to execute the code inside it. To call a function, we simply use its name followed by parentheses. If the function takes any parameters, we can pass values for those parameters inside the parentheses.

Functions can have multiple parameters, separated by commas. This allows us to pass in different values each time we call the function. Each function execution is independent of previous executions, providing us with flexibility and modularity in our code.

One of the great advantages of functions is that they allow us to write code that runs later, rather than immediately. This opens up possibilities for creating interactive websites where code is executed only when certain events occur. For example, we can attach functions to buttons so that they run only when the button is pressed. This allows us to provide user interaction and control the flow of our code.

Functions can be triggered by a wide variety of events, not just button clicks. We can also manually call functions in our code whenever we need to. By using functions, we can create code that runs only when specific conditions are met, adding more interactivity and responsiveness to our web pages.

In the provided example, a function called "outputResult" is called. This function is defined in the "vendor.js" file and takes two parameters. When the function is called, it changes the text displayed on the web page by interacting with specific parts of the HTML code. The values passed as parameters to the function are used to update the web page dynamically.

To create our own functions, we can define them anywhere in our script. JavaScript will read the entire file before executing it, so it doesn't matter if we define a function before or after using it. However, for variables and constants defined with "let" and "const", this is not the case.

To better understand how functions work, let's create a function that adds two numbers. For example, we can create a function that adds the value entered in an input field to a current result. By calling this function, we can update the result dynamically based on user input.

Remember, functions allow us to encapsulate code, make it reusable, and execute it at specific times or in response to events. They provide us with the flexibility to create interactive and dynamic web pages.

In JavaScript, functions are an essential part of the language. They allow us to encapsulate a block of code and execute it whenever we need it. In this lesson, we will focus on introducing functions and understanding how to name them effectively.

When creating a function, it is important to choose a name that accurately describes the action the function will perform. For example, if we are creating a function that adds two numbers, a name like "add" or "addNumbers" would make sense. These names clearly convey the purpose of the function.

On the other hand, it is not recommended to use vague names like "result" for a function. Such names do not provide any information about what the function actually does. Remember, the goal is to have a name that describes the action performed by the function.

By giving our functions descriptive names, we make our code more readable and easier to understand. When someone else reads our code or when we revisit it after some time, the function names should provide a clear indication of their purpose.

Let's take a moment to consider the importance of naming functions effectively. It helps us and others who read our code to quickly grasp the functionality of the function without having to analyze the code in detail.

When creating functions in JavaScript, it is crucial to choose names that accurately describe the action performed by the function. This allows for better readability and understanding of the code.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: ADDING A CUSTOM FUNCTION**

A function in JavaScript is a reusable block of code that performs a specific task. Functions can be named and can optionally take parameters, which are values passed into the function for it to use. In this case, we have a function named "add" which is used to perform addition.

When defining a function, we use the "function" keyword followed by the function name and optional parameters. In JavaScript, parameters are enclosed in parentheses. The function body, where the actual code is written, is enclosed in curly braces. It's important to note that a semicolon is not required after the closing curly brace of a function.

Inside the function body, we can perform various operations. In this example, we define a constant named "result" and assign it the value of adding the two parameters, "num1" and "num2". These parameters are like variables and are automatically defined by JavaScript when the function is called. It's worth mentioning that these variables are only accessible within the function body and not outside of it.

The constant "result" is defined inside the function, which means it is only accessible within the function itself. This is known as the function's local scope. In JavaScript, variables/constants can be defined both at the top level of a script (global scope) and inside a function. It's important to understand where variables/constants are defined as it affects their accessibility.

The code is indented for readability, but the indentation is not required for the code to run. It's a common practice to have the function body indented to improve code readability. The amount of indentation can vary, and there are discussions about whether to use tabs or spaces and how many spaces to use. It's recommended to choose a consistent approach and configure your code formatting tool accordingly.

In this example, the "result" constant is calculated and stored within the function. We can then use this result in various ways. For instance, we can output it using the "alert" function, which is built into JavaScript and is used to display a message in a dialog box.

To summarize, a function in JavaScript is a reusable block of code that performs a specific task. It can be named and can have optional parameters. The function body contains the actual code to be executed. Variables/constants defined within a function are only accessible within that function. The code can be indented for readability, and the amount of indentation is a matter of personal preference.

In JavaScript, functions play a crucial role in organizing and reusing code. They allow us to encapsulate a set of instructions and execute them whenever needed. In this didactic material, we will focus on adding a custom function in JavaScript.

JavaScript provides several built-in functions, depending on whether you are running JavaScript in a browser or outside of it. One such built-in function available in the browser is the "alert" function. The alert function allows us to display a message to the user. To use the alert function, we provide the message as a parameter. For example, we can use the alert function to display the result of a calculation by concatenating the result with a message.

To define a custom function, we use the "function" keyword followed by the function name and parentheses. Inside the parentheses, we can specify any parameters the function requires. Parameters act as placeholders for values that will be passed to the function when it is called. In our example, we define a function called "add" which takes two parameters, "num1" and "num2".

To call a function, we simply use the function name followed by parentheses. Inside the parentheses, we provide the concrete values that the function should use for execution. When we call the "add" function with the values 1 and 2, the function is executed, and the result is displayed using the alert function.

It's important to note that a function can have as many parameters as needed, and it can also return a value. In our example, the "add" function does not return a value, but instead displays the result using the alert function.

However, functions can be used for various purposes, such as sending HTTP requests to a server or storing data.

By using functions, we can avoid repeating code and make our programs more efficient and maintainable. Instead of copying and pasting code multiple times, we can define a function once and call it whenever needed, providing different arguments each time.

To summarize, in JavaScript, functions are essential for organizing and reusing code. They allow us to encapsulate a set of instructions and execute them multiple times with different arguments. By using functions, we can make our code more modular, efficient, and easier to maintain.

In JavaScript, functions play a crucial role in organizing and reusing code. One important aspect of functions is their ability to return values. The "return" keyword allows us to specify the value that a function should produce as its result.

When we want a function to perform a specific task and produce a result, we can use the "return" keyword to indicate what that result should be. This is particularly useful when we want to perform calculations or operations on data and obtain a specific output.

For example, let's consider a function called "add" that takes two numbers as input and returns their sum. Inside the function, instead of using the "alert" function to display the result, we can simply use the "return" keyword followed by the expression that represents the sum of the two numbers.

By using the "return" keyword, we indicate that the function should produce a value as its result. This means that when we call the "add" function, instead of seeing any visual impact on the page, we will receive the sum of the two numbers as a value.

It's important to note that the "return" keyword immediately ends the execution of the function and sends the specified value back to the caller. This means that any code written after the "return" statement will not be executed.

To illustrate this concept, let's consider the following code:

1.	function add(a, b) {
2.	return a + b;
3.	}
4.	
5.	let result = add(3, 4);
6.	console.log(result); // Output: 7

In this example, we define a function called "add" that takes two parameters, "a" and "b". Inside the function, we use the "return" keyword to specify that the result of the function should be the sum of "a" and "b". When we call the "add" function with the arguments 3 and 4, the function returns the sum, which is then stored in the variable "result". Finally, we log the value of "result" to the console, which outputs 7.

By utilizing the "return" keyword, we can create functions that not only perform tasks but also produce specific results. This allows us to build more complex programs by combining and reusing functions in various ways.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: RETURNING VALUES IN A FUNCTION**

In JavaScript, functions are a fundamental concept that allow us to group and organize blocks of code that perform specific tasks. Functions can also return values, which means they can provide us with a result or output that we can use in our code.

To return a value from a function in JavaScript, we use the ``return`` keyword followed by the value we want to return. This value can be of any data type, such as numbers, strings, or even objects.

When a function returns a value, we can store that value in a variable or constant. This allows us to use the result of the function call in other parts of our code. For example, we can assign the result to a variable like this:

```
1. const additionResult = add(1, 2);
```

In the above example, we have a function called ``add`` that takes two parameters and returns their sum. We call the function with the values 1 and 2, and store the result in the ``additionResult`` constant.

By storing the result in a variable or constant, we can use it later in our code. For instance, we can use the ``additionResult`` to update the value of another variable:

```
1. currentResult = additionResult;
```

In this case, we assign the value of ``additionResult`` to the ``currentResult`` variable. This allows us to use the calculated result in further calculations or operations.

It's important to note that when declaring variables or constants in JavaScript, they need to be declared before they are used. If we try to use a variable or constant before declaring it, we will encounter an error. For example:

```
1. console.log(currentResult); // Error: currentResult is not defined
2.
3. const currentResult = 0;
```

In the above code snippet, we try to access the value of ``currentResult`` before declaring it. This will result in an error because the variable has not been defined yet. To avoid this error, we need to declare the variable before using it.

Returning values in JavaScript functions allows us to obtain results that we can use in our code. By storing the returned value in a variable or constant, we can manipulate and use the result in further calculations or operations.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: EXPLORING THE IMPORTANCE OF CODE ORDER**

In JavaScript, the order of code execution plays a crucial role in the outcome of our programs. When it comes to variables, the order matters. If we declare a variable and then use it, JavaScript will execute the code without any issues. However, if we try to use a variable before declaring it, we will encounter an error.

On the other hand, functions in JavaScript behave differently. Unlike variables, the order of function declarations does not affect the execution of the code. When the browser loads a script, it parses the entire script from top to bottom. It identifies any functions present and automatically moves them to the top, registering them before executing the rest of the script. This behavior allows us to call functions before defining them without causing any errors.

As developers, we have the flexibility to choose where to place our functions within our script. Some prefer to have functions towards the beginning of the file, allowing for easier readability and understanding of the code. Others prefer to have function definitions at the bottom of the file. Ultimately, there is no right or wrong approach, and it depends on personal preference and maintaining a consistent code style.

It is worth noting that the order of variables and constants does matter. However, for functions, we can place them anywhere in the script, and JavaScript will handle their registration automatically. It is important to find an approach that works best for you and stick to it within a project. However, in different applications or when refactoring existing code, you can always switch between placing functions at the top or bottom of the file.

In our example, the function is placed towards the beginning of the script. This choice allows developers to easily locate and understand the function when reading the file. However, the decision of where to place functions is entirely up to the developer.

Additionally, it is important to consider the scope of variables and constants. If a piece of data is only used within a function, it is recommended to define it within the function itself. This practice helps to encapsulate the data and make the code more self-contained. In our code, we have a constant defined within the function and a variable and constant defined outside of the function. This approach ensures that the variables and constants are accessible where they are needed and avoids polluting the global scope unnecessarily.

While it is possible to manipulate global variables within functions, it is generally not considered good practice. Modifying global variables inside functions can make code harder to understand and predict. It is advisable to limit the use of global variables and instead pass data as arguments and return values from functions to maintain better control and organization of the code.

The order of code execution is crucial in JavaScript. Variables must be declared before they are used, while functions can be called before they are defined. Developers have the freedom to choose where to place functions within their scripts, but it is important to maintain a consistent code style. It is also recommended to define variables and constants within functions if they are only used within that function to maintain encapsulation and avoid global scope pollution.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: INTRODUCTION TO GLOBAL AND LOCAL SCOPE**

Functions in JavaScript can have both global and local scope. Understanding the concept of scope is important for writing efficient and predictable code.

A function that is defined outside of any other function or block has global scope. This means that it can be accessed from anywhere in the code, including other functions. Global variables and constants can also be accessed within functions. However, it is generally considered best practice to avoid manipulating global variables within functions, as it can lead to unpredictable behavior.

On the other hand, variables and constants defined inside a function have local scope. They are only accessible within the function itself and cannot be used outside of it. This is known as block scope, which is marked by curly braces {}. Local variables and constants cannot be accessed from outside the function, even if the function has already been called.

The return statement is used to pass data from inside a function to the outside world. When a function encounters a return statement, it immediately stops executing and returns the specified value. Any code that follows the return statement will not be executed.

It is worth noting that a function can also return nothing, which is indicated by not specifying a return value. This is useful in scenarios where the function performs some calculations and then sends the result to a server via an HTTP request. In such cases, the function can be terminated conditionally using an if statement, which allows for the checking of a condition and exiting the function if the condition is true.

In JavaScript, it is possible to define a variable or constant with the same name as a global variable or constant within a function. This process is called shadowing. The local variable or constant will take precedence within the function, effectively "shadowing" the global variable or constant. The concept of shadowing will be explored further in the next lecture.

To summarize, functions in JavaScript have access to variables and constants in their surrounding context. However, variables and constants defined inside a function are only accessible within that function. The return statement is used to pass data from inside a function to the outside world, and it also terminates the function execution. Shadowing allows for the creation of local variables or constants with the same name as global variables or constants.

In JavaScript, functions play a crucial role in organizing and reusing code. They allow us to encapsulate a set of instructions into a single unit that can be called and executed multiple times. When working with functions, it is important to understand the concept of scope, which determines the visibility and accessibility of variables within a program.

JavaScript has two types of scope: global scope and local scope. Global scope refers to variables or functions that are accessible throughout the entire program, while local scope refers to variables or functions that are only accessible within a specific block of code.

Global scope variables are declared outside of any function and can be accessed from anywhere within the program. They have a global scope and are visible to all functions and code blocks.

On the other hand, local scope variables are declared inside a function or a code block and are only accessible within that specific function or block. They have a local scope and are not visible outside of their enclosing function or block.

When a function is called, a new local scope is created for that function. This means that variables declared within the function are only accessible within that function and not outside of it. Similarly, variables declared outside of a function are not accessible within the function unless they are explicitly passed as parameters.

It is important to note that when a variable is declared with the `var`, `let`, or `const` keyword inside a block of

code, it is scoped to that block of code. This means that it is only accessible within that block and not outside of it.

When a function is executed, JavaScript first looks for variables within the local scope. If a variable is not found, it then looks for it in the next outer scope, and so on, until it reaches the global scope. This is known as variable or function resolution.

To avoid unexpected behavior and potential conflicts, it is generally recommended to limit the use of global variables and instead use local variables within functions. This helps to encapsulate data and prevent unintended modifications.

Functions in JavaScript are essential for code organization and reusability. They allow us to encapsulate a set of instructions into a single unit that can be called and executed multiple times. Understanding the concept of scope is crucial for determining the visibility and accessibility of variables within a program. JavaScript has global and local scope, with global variables being accessible throughout the entire program and local variables being accessible only within specific functions or blocks of code.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: RETURN STATEMENT**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: EXECUTING FUNCTIONS INDIRECTLY**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: TYPE CONVERSION**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: SPLITTING CODE AND FUNCTIONS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: FUNCTIONS IN JAVASCRIPT****TOPIC: CLICKABLE BUTTONS WITH EVENT LISTENERS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: ADVANCING IN JAVASCRIPT****TOPIC: ADDING COMMENTS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: ADVANCING IN JAVASCRIPT****TOPIC: MORE OPERATORS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: ADVANCING IN JAVASCRIPT****TOPIC: MORE CORE DATA TYPES**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: ADVANCING IN JAVASCRIPT****TOPIC: ARRAYS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: OBJECTS IN JAVASCRIPT****TOPIC: OBJECTS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: OBJECTS IN JAVASCRIPT****TOPIC: ACCESSING OBJECT DATA**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: OBJECTS IN JAVASCRIPT****TOPIC: ADDING A REUSABLE FUNCTION THAT USES OBJECTS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: OBJECTS IN JAVASCRIPT****TOPIC: UNDEFIND, NULL, NAN**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: OBJECTS IN JAVASCRIPT****TOPIC: TYPEOF**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: OBJECTS IN JAVASCRIPT****TOPIC: SCRIPT TAGS, DEFER, ASYNC**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: OBJECTS IN JAVASCRIPT****TOPIC: SUMMARY**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: INTRODUCTION TO DEBUGGING AND EFFICIENT DEVELOPMENT**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: OVERVIEW OF EFFICIENT DEVELOPMENT WORKFLOW**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: THE IDE LOOK AND FEEL**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: USING SHORTCUTS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: AUTO COMPLETION AND HINTS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: EXTENSIONS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: WORKING WITH EDITOR SETTINGS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: USING DIFFERENT EDITOR VIEWS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: FINDING HELP**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: GOOGLING CORRECTLY FOR JAVASCRIPT HINTS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: DEBUGGING OVERVIEW**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: WORKING WITH ERROR MESSAGES**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: DEBUGGING LOGICAL ERRORS WITH CONSOLE.LOG**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: CHROME DEVTOOLS AND BREAKPOINTS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: TESTING CODE CHANGES IN DEVTOOLS**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: DEBUGGING VS VISUAL STUDIO CODE**

This part of the material is currently undergoing an update and will be republished shortly.

EITC/WD/JSF JAVASCRIPT FUNDAMENTALS DIDACTIC MATERIALS**LESSON: DEBUGGING AND EFFICIENT DEVELOPMENT****TOPIC: SUMMARY**

This part of the material is currently undergoing an update and will be republished shortly.