|  |  |
| --- | --- |
| 0x00ES6 Basics: | |
| * Concepts:   Rest parameter, spread syntax, template literal, computed property name  **Spread syntax**:  Uses of Spread syntax:  Given : array1 = [1, 2, 3] array2 = [a, b, c]  Obj1 = { a: 1, b: 2} obj2 = {x: 3, y: 4}  String = “tom”   1. Array related:    1. shallow copy array   copiedArray1 = [. . . array1] >>> [1, 2, 3]   * 1. concatenate array   concatenated = [. . . array1, . . . array2]  >>> [1, 2, 3, a, b, c]   1. string related:    1. unpack: unpacked = [...string] >>> [‘t’, ‘o’ , ‘m’] 2. Object related:    1. Merger objects: Const newobj = {…obj1, …obj2} >>> { a: 1, b: 2, x: 3, y: 4}    2. Add new property to object   const addobj = {…obj1, c: 10}  >>> {a: 1, b: 2, c: 10}   1. As function parameter (as rest parameter)   FunctionName(… arguments)  Two types of copy shallow and deep   1. **Deep copy**: involves copying not only the top-level properties of an object but also ensuring that any nested objects within it are also copied and not just referenced. 2. **Shallow copy**: the outer object (or the first level of properties) is copied, the properties inside the object (nested objects or arrays) are still references to the original properties.   . const originalObject = {x: 1, y: {z: 2 } };  // Creating a deep copy  const deepCopyObject = JSON.parse(JSON.stringify(originalObject));  // creating shallow copy  Const shallowCopyObject = [... originalObject];  // if the properties are changed like this  originalObject.x = 10;  originalObject.y.z = 20;  console.log(originalObject); // {x: 10, y: { z: 20 }}  console.log(deepCopyObject); // {x: 1, y: { z: 2 }}  console.log(shallowCopyObject); // { x: 1, y: {z: 20}} | **Object Creation**  Syntax: **const myObject = { key: 'value'};**  when creating property for js object using object literal  the key can be created using three methods:  given : **const myKey = “key1”;**   1. **Const obj = {key1: value};//name variable** 2. **Const obj = {‘key1’: value};//single quote** 3. **Const obj = {“key1’: value};//double quote** 4. **Const obj = {[mykey]: value};// computed property (if it is calculated then it should be enclosed in [])**   **Object Methods**  **Object.entries()** method returns an array of a given object's own enumerable property [key, value]  **Object.keys()** method returns an array of a given object's own enumerable property names (keys).  **Use of [ ]**   1. Array creation: **let array = [ 1, 2, 3]** 2. Object property access **let valueName= obj[‘key name’]** 3. Destructuring : **let array = [ 1, 2, 3]**   **let [a, b, c] = array; >>> a = 1, b = 2, c = 3**   1. Regular expressions:   Let regex = /[abc]/ // matchs any a, b or c characters |
|  |  |

|  |  |
| --- | --- |
|  | |
|  |  |

|  |  |
| --- | --- |
| 0x02 ES6 Class | |
| **Type Checking with typeof:**  typeof(name) === "string";  // Validates if 'name' is a string  typeof(age) === "number"; // Validates if 'age' is a number   **Instance Checking:**  obj instanceof ClassName; // Checks if 'obj' is an instance of 'ClassName'   **Null or Undefined Checking:**  typeof variable !== 'undefined' && variable !== null; // Checks if variable is neither undefined nor null   **Checking Object Keys:**  obj.hasOwnProperty('keyName'); // Checks if 'obj' has a specific property  'keyName' in obj; // Another way to check if 'obj' has a specific property   **Array Checking**  Array.isArray(someArray); // Checks if 'someArray' is an array   **Checking for Functions:**  typeof someFunction === 'function'; // Checks if 'someFunction' is a function   **Truthy/Falsy Checking:**  if (variable) { /\* ... \*/ } // Checks if 'variable' is truthy (not null, undefined, 0, false, empty string, etc.)   **Regular Expression Validation:**  javascript   const regex = /^[a-zA-Z]+$/;  regex.test(someString); // Checks if 'someString' matches the regular expression pattern   **Type Checking using Object.prototype.toString:**  Object.prototype.toString.call(variable) === '[object Object]'; // Checks if 'variable' is an object   Custom **Validation Functions:** You can create custom functions to perform specific validation based on your requirements.   1. function isValidEmail(email) { 2. // Custom validation logic for email 3. } | **Static classes:**  Are functions that belong to the class itself rather than to instances created by the class. They are accessed using the class name rather than an instance of the class. These functions are declared with the static keyword within a class.  Symbol.toPrimitive method is a special symbol that allows objects to define their behavior when they are coerced to a primitive type (such as a number, string, or default). This method gets called by JavaScript internally when an object is converted to a primitive type explicitly or implicitly.  The Symbol.toPrimitive method takes a single argument hint, which represents the type of conversion being performed (e.g., 'number', 'string', or 'default').  There are 6 primitive types in js  Number, string, Boolean, null, undefined, symbol  Example: const class1 = class()  Class1.toString() ------- Symbol.toString() method is called  String(class1) ---when casted(coerced)—Symbol.toPremitive() method is called |

|  |  |
| --- | --- |
| 0x03. ES6 data manipulation | |
| **Map:** method creates a new array by calling a provided function on every element in the original array.  **Application:** Doubling each number in an array  const numbers = [1, 2, 3, 4];  const doubled = numbers.map(num => num \* 2);  **Filter:** method creates a new array with elements that pass a certain condition defined by a provided function.  **Application:** Doubling each number in an array  const n = [1, 2, 3, 4];  const even = arr.filter(n => n % 2 === 0);  **Reduce:** method applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value.  **Application:** Summing all numbers in an array  const ar = [1, 2, 3, 4];  const sum = arr.reduce((acc, n) => acc + nu, 0);  **Every():**method checks if all elements in an array pass a certain condition provided by a function. It returns true if all elements satisfy the condition; otherwise, it returns false.  **Application:** Check if all are positive  const arr = [1, 2, 3, 4];  const rslt = arr.every((n) => n > 0);  **find():**method is used to get the first element in an array that satisfies a provided testing function. It returns the value of the first element in the array that satisfies the provided function.  **Application:** find the first even number  const arr = [1, 2, 3, 4];  const rslt = arr.find((n) => n % 1 === 0);  output>>> 2;  **slice ():**method in JavaScript is used to extract a section of an array or a string and create a new array or string from that section. It doesn't modify the original array or string; instead, it returns a shallow copy of the selected elements or characters.  **Application:** find the first even number  const arr = [1, 2, 3, 4];  const rslt = arr.find((n) => n % 1 === 0);  output>>> 2;  **startsWith()** method in JavaScript is used to determine whether a string starts with the characters of a specified string, returning true or false as the result.  const str = 'Hello, World!';  console.log(str.startsWith('Hello')); True  console.log(str.startsWith('World', 7)); True | ITERATE or LOOP  Iterate across a dictionary in javascript   1. for (const [key, value] of Object.entries(dictionary)) {   console.log(`${key}: ${value}`);  }   1. Object.entries(dictionary).forEach(([key, value]) => {   console.log(`${key}: ${value}`);  **Using Template Literals:**  javascript  console.log(`${key}: ${value}`);  This uses template literals, which is a modern JavaScript feature that allows you to embed expressions within strings using ${...}. It provides a cleaner and more readable way to interpolate variables into strings.  **check if key exists in a dictionary**  const myObject = { k1: 'v1', k2: 'v2'};  if (myObject.hasOwnProperty(keyToCheck))  console.log(‘key exists”);  the above can create error by ESlinter so use this instead  if (Object.prototype.hasOwnProperty.call(myObject, keyToCheck)) {  console.log (`${keyToCheck} exists in the object.`);  Sets:   * Store unique values * Are indexed by their insertion order   **Array Buffer:**   * It does not have methods to directly interact with the data it holds; instead, it serves as a container for raw binary data * It allocates a specific length of memory in bytes but does not provide methods to directly read or write data.   **DataView** is an interface that provides a low-level interface for reading and writing multiple number types in an ArrayBuffer. It allows you to interpret and manipulate the raw binary data stored in an ArrayBuffer.  ArrayBuffer and DataView are two concepts in JavaScript used to handle binary data.  **Bytes for datatypes**  null and undefined: typically take up 4 bytes of memory.  String: 2 bytes (16 bits) for individual characters. due to JavaScript's use of UTF-16 encoding.  Number: 64 bits (8 bytes) for both float and integer.  Boolean: 2byte  Object and array: depends on the property they contain. |

|  |  |
| --- | --- |
| 0x04 Typescript | |
|  |  |

|  |  |
| --- | --- |
| 0x00 Html Advanced | |
| 1. <!DOCTYPE html>   declaration represents the HTML5 standard.   1. <html lang="en" dir="ltr">   dir="ltr" specifies the text direction as left-to-right (ltr stands for left-to-right)   1. <meta name="description" content="Techium is a digital agency">   help search engines understand the content of your webpage, potentially influencing how it appears in search results. Additionally, it provides a concise summary for users browsing the web.   1. <link rel="icon" type="image/x-icon" href="./favicon.ico">       <link rel="icon" type="image/png" href="./favicon.png">  By providing both formats (favicon.ico and favicon.png) in the <link> tags, you cater to a wider range of browsers and devices, ensuring that your website's favicon displays correctly regardless of the browser's preference or compatibility with different file types. | * Definitions:   <dl> definition list  <dt> term  <dd> defnition   * <blockquote>: long quote or part of a quote   <cite>: citation for quote   * <q>: short quote on a single line, no citation * <hr> : horizontal rule * <address> : to write address * <small> : tag is commonly used to display side comments, copyright information, disclaimers, etc., in a smaller font size than the surrounding text. * <pre> tag in HTML stands for "preformatted text." It defines preformatted text where whitespace, line breaks, and spaces are displayed exactly as they appear in the HTML code. * **<details>** element in HTML is used to create a disclosure widget that can be toggled open or closed, revealing or hiding additional content within a document. It's typically used to create collapsible sections of content, where the additional details are initially hidden but can be expanded by the user. |
| For accessibility to a table scope attribute is added  <table>  **<caption>Star Wars Trilogy Data</caption>**  <thead>  <tr>  <th **scope="col">**Title</th>  <th scope="col">Director</th>  <th scope="col">Release Date</th>  </tr>  </thead>  <tbody>  <tr>  <th scope="row">Movie 1</th>  <td>Director 1</td>  <td>2022</td>  </tr>  <tr>  <th scope="row">Movie 2</th>  <td>Director 2</td>  <td>2023</td>  </tr>  <!-- Additional rows -->  </tbody>  </table>   * The scope="col" attribute in the <th> tags within the <thead> section specifies that these headers represent columns. * The scope="row" attribute in the <th> tags within the <tbody> section signifies that these headers represent rows and contain row-specific data (in this case, the names of movies).   Using the scope attribute helps screen readers interpret table headers correctly, allowing users with disabilities to understand the structure and data within the table more effectively. | * VIDEO and AUDIO    <video>: Embeds the video.  <video controls loop poster="thumbnail.jpg">  <source src="video.mp4" type="video/mp4">  **fallback message** Sorry, your browser doesn't support HTML5 video.  </video>   * controls: Displays the video controls (play, pause, etc.). * loop: Sets the video to loop playback. * poster: Displays an image (thumbnail.jpg) while the video is loading. * <source>: Specifies the video source (BigBuckBunny.mp4) and its MIME type (video/mp4). * The text "Sorry, your browser doesn't support HTML5 video." acts as alternative content displayed if the browser cannot render the video. This provides a **fallback message** for users whose browsers do not support HTML5 video or if the video fails to load.   .Iframe: embed websites within our page  <iframe src="https://www.example.com" width="600" height="400" frameborder="0" title="Embedded Example Website"> **fallback message** </iframe>   * title="Holberton School": Provides a title for the iframe. * width="350" and height="200": Sets the width and height of the iframe. * src="https://www.youtube.com/embed/41N6bKO-NVI": Specifies the source URL for the embedded content. * frameborder="0": Removes the iframe border. * allowfullscreen: Allows the embedded content to be displayed in full-screen mode if supported. |

|  |  |
| --- | --- |
| **ACCESIBILITY** | |
| * Inside <svg> tag: use <Title> <title> element provides a title or description for the SVG graphic. It's usually used for accessibility purposes, allowing screen readers to announce the title of the icon to users who may be visually impaired. |  |