**Assignment-4**

1. **Write short note on arrow Functions with syntax?**

* **JavaScript Arrow Function:** Arrow function {() =>} is concise way of writing Javasript functions in shorter way. Arrow functions were introduced in the ES6 version. They make our code more structured and readable. Arrow function is one of the features introduced in the ES6 (ES6 stands for ECMA Script6. ECMA Script was created to standardize JavaScript, and ES6 is the 6th version of ECMA Script, it was published in 2015, and is also known as ECMA Script 2015.) version of JavaScript. It allows you to create functions in a cleaner way compared to regular functions. .

|  |
| --- |
| **Example:** // function expression  let x = function(x, y) { return x \* }   * can be written as   // using arrow functions  let x = (x, y) => x \* y; |

* **Arrow Function Syntax:** The syntax of the arrow function is:

|  |
| --- |
| **myFunction = (arg1, arg2, ...argN) => { statement(s) }**  Here, myFunction is the name of the function, arg1, arg2, ...argN are the function arguments, statement(s) is the function body. |
| **If the body has single statement or expression, you can write arrow function as:**  let myFunction = (arg1, arg2, ...argN) => expression |

1. **What is Template Strings OR Template literals with syntax?**

* **Template Strings OR Template literals** are literals delimited with backtick (`) characters, allowing for [multi-line strings](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals), [string interpolation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals) with embedded expressions, And special constructs called [tagged templates](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals). Template literals are sometimes informally called template strings, because they are used most commonly for [string interpolation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals) (to create strings by doing substitution of placeholders). However, a tagged template literal may not result in a string; it can be used with a custom [tag function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals) to perform whatever operations you want on the different parts of the template literal. **Synonyms:** Template Literals, Template Strings, String Templates, Back-Tics Syntax.
* **Syntax:** Template Literals use back-ticks (``) rather than the quotes ("") to define a string:

|  |
| --- |
| let text = `Hello World!`;  **OR**  let text = `He's often called "Johnny"`;  **OR Multi-line**  let text = `The quick brown fox jumps over the lazy dog`; |

1. **Explain with syntax rest parameter & Spread Operator?**

* **The rest parameter:** The rest parameter is an improved way to handle function parameters, allowing us to more easily handle various inputs as parameters in a function. The rest parameter syntax allows us to represent an indefinite number of arguments as an array. With the help of a rest parameter, a function can be called with any number of arguments, no matter how it was defined. Rest parameter is added in ES2015 or ES6 which improved the ability to handle parameter.

|  |
| --- |
| **Syntax:**  /... is the rest parameter (triple dots)  function functionname(...parameters)  {  statement;  } |

* **Spread Operator:** The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object. The spread operator ... is used to expand or spread an iterable or an array.

|  |
| --- |
| **Example:**  const arrValue = ['My', 'name', 'is', 'Jack']; console.log(arrValue); // ["My", "name", "is", "Jack"] console.log(...arrValue); // My name is Jack  **Output:**  [ 'My', 'name', 'is', 'Jack' ]  My name is Jack |

1. **What is Java Scripts Object Literals? Explain example & syntax?**

* **Object literals** are one of the most commonly used data structures in JavaScript. They are used to store collections of data, and can be used to represent complex data structures such as objects, arrays, functions, and even regular expressions. Object literals are also used to store information about a particular instance of an object, such as its state or behavior. Object literals are written in the form of key-value pairs, where each key is a string and each value can be any valid JavaScript data type.
* **For example, the following object literal stores information about a person:**

|  |
| --- |
| script.js  const person = {  name: 'John Doe', age: 30, address: '123 Main Street'  }; |

* **Example:**

|  |  |
| --- | --- |
| const person = {   firstName: "John",   lastName : "Doe",   id       : 5566,   fullName : function() {     return this.firstName + " " + this.lastName;   } };. | * In the example above, this refers to the person object. In this first Name means the first Name property of this. In this First Name means the firstName property of person. |

1. **Short note on JavaScript Object Destructuring & example?**

* **Destructuring**is a JavaScript expression that allows us to extract data from arrays, objects, and maps and set them into new, distinct variables. The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables. Destructuring allows us to extract multiple properties, or items, from an array​ at a time. JavaScript Object Destructuring is the syntax for extracting values from an object property and assigning them to a variable. The destructuring is also possible for JavaScript Arrays. By default, the object key name becomes the variable that holds the respective value. **Here is the old way of assigning array items to a variable:**

|  |
| --- |
| const vehicles = ['mustang', 'f-150', 'expedition'];  // old way  const car = vehicles[0];  const truck = vehicles[1];  const suv = vehicles[2]; |

* **Example:**ith destructuring:

|  |
| --- |
| const vehicles = ['mustang', 'f-150', 'expedition'];  const [car, truck, suv] = vehicles; |

* When destructuring arrays, the order that variables are declared is important. If we only want the car and suv we can simply leave out the truck but keep the comma:

|  |
| --- |
| const vehicles = ['mustang', 'f-150', 'expedition'];  const [car,, suv] = vehicles; |

1. **Explain JavaScript inheritance with syntax?**

* **Inheritance** refers to the act of inheriting something. It could be a child inheriting the property of their parents or a new species inheriting some property of the older species during evolution. Even in programming, the concept of inheritance exists. Inheritance in javascript aids a new class to have all the functionality of another class while having its own functionality as well. The inheritance in javascript primarily involves two segments:
* **Child class:** The class which inherits the properties of another class is known as the child class.
* **Parent class:** The class whose property is being inherited is known as the parent class. This can be done using the extends and super keywords. We use the extends keyword to implement the inheritance in ES6. Let’s see the ES6 version of inheritance:

|  |  |
| --- | --- |
| class Shape {      constructor( color ) {          this.color = color;     }       getColor() {          return this.color; }  }   class Rectangle extends Shape {      constructor( color, width, height ) {          super( color );          this.width = width;          this.height = height; }       getArea() {          return this.width \* this.height;      } }   let rectangle = new Rectangle( 'red', 5, 8 );  console.log( "Area:\t\t" + rectangle.getArea() );  console.log( "Color:\t\t" + rectangle.getColor() );  console.log( "toString:\t" + rectangle.toString() ); |  |

1. **Explain JavaScript split & substr with syntax & example?**

* **The split() method splits (divides)** a string into two or more substrings depending on a splitter (or divider). The splitter can be a single character, another string, or a regular expression. After splitting the string into multiple substrings, the split() method puts them in an array and returns it.

1. **String split() method:** The String.prototype.split() divides a [string](https://www.javascripttutorial.net/javascript-string/) into an [array](https://www.javascripttutorial.net/javascript-array/) of substrings: **split([separator, [,limit]]);**
2. **Separator :** The separator determines where each split should occur in the original string. The separator can be a string. Or it can be a [regular expression](https://www.javascripttutorial.net/javascript-regular-expression/). If you omit the separator or the split() cannot find the separator in the string, the split() returns the entire string.
3. **Limit :** The limit is zero or positive integer that specifies the number of substrings. The split() method will stop when the number of substrings equals to the limit. If the limit is zero, the split() returns an empty array. If the limit is 1, the split() returns an array that contains the string.

* **Let’s take some examples of using the split() method.**

1. **Splitting the strings into words example:**

|  |
| --- |
| let str = 'JavaScript String split()';  let substrings = str.split(' ');  console.log(substrings);Code language: JavaScript (javascript)  **Output:**  ["JavaScript", "String", "split()"]Code language:  JavaScript (javascript)  **Note**: space (‘ ‘) has been removed in the substrings. |

1. **Returning a limited number of substrings example:** The following example uses the split() method to divide a string into substrings using the space separator. It also uses the second parameter to limit the number of substrings to two:

|  |
| --- |
| let str = 'JavaScript String split()';  let substrings = str.split(' ',2);  console.log(substrings);  Code language: JavaScript (javascript)  **Output:**  ["JavaScript", "String"]  **Summary:** Use the JavaScript String split() to divide a string into an array of substrings by a separator.Use the second parameter (limit) to return a limited number of splits. |

1. **JavaScript  substr():** The substr() method extracts a part of a string. The substr() method begins at a specified position, and returns a specified number of characters. The substr() method does not change the original string. To extract characters from the end of the string, use a negative start position.
2. **Extracting a substring from the beginning of the string example:** The following example uses the substring method to extract a substring starting from the beginning of the string:

|  |
| --- |
| let str = 'JavaScript Substring';  let substring = str.substring(0,10);  console.log(substring);  **Output:**  JavaScript |

1. **Extracting a substring to the end of the string example:** The following example uses the substring() to extract a substring from the index 11 to the end of the string:

|  |
| --- |
| let str = 'JavaScript Substring';  let substring = str.substring(11);  console.log(substring);  **Output:**  Substring |

1. **Extracting domain from the email example:** The following example uses the substring() with the [indexOf()](https://www.javascripttutorial.net/javascript-array-indexof/) to extract the domain from the email:

|  |
| --- |
| let email = 'john.doe@gmail.com';  let domain = email.substring(email.indexOf('@') + 1); console.log(domain);  **How it works:** First, the indexOf() returns the position of the @ character, Then the substring returns the domain that starts from the index of @ plus 1 to the end of the string.  **Summary :**The JavaScript substring() returns the substring from a string between the start and end indexes. |

1. **What is Try...Catch And Throw block in JS?**

* **The try statement defines a code block to run (to try):** The catch statement defines a code block to handle any error. The finally statement defines a code block to run regardless of the result. The throw statement defines a custom error.
* **try{} statement:** Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{} block for taking suitable actions and handle the error. Otherwise, it executes the code written within.
* **catch{} statement:** This block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the built-in handler. This block executes only when any error-prone code needs to be handled in the try block. Otherwise, the catch block is skipped.
* **Throw Statement:** Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.

|  |
| --- |
| <html>  <head>Exception Handling</head>  <body>  <script>  **try** {  **throw new Error**('This is the throw keyword'); //user-defined throw statement.  }  **catch** (e) {    document.write(e.message); // This will generate an error message  }  </script>  </body>  </html> |

1. **How to Use JavaScript Date Methods explain?**

* **The date object** supports numerous date methods, but for this article, we only need the current date and will only use three methods:
* **getFullYear() –** we will use this method to get the year as a four digit number (yyyy), for example 2022.
* **getMonth() –** This gets the month as a number (0-11), for example 2 for March since it’s a zero based index (meaning it starts from 0).
* **getDate() –** gets the day as a number (1-31).
* **Let’s now put all these together based on the format in which we want our date to appear:**

|  |
| --- |
| const date = new Date();  let day = date.getDate();  let month = date.getMonth() + 1;  let year = date.getFullYear();  // This arrangement can be altered based on how we want the date's format to appear.  let currentDate = `${day}-${month}-${year}`;  console.log(currentDate); // "17-6-2022" |

* **Current Time in JavaScript:** Use the following script to get the current time using JavaScript in “H:i:s” format.

|  |
| --- |
| today = new Date();  var time = today.getHours() + ":" + today.getMinutes() + ":" + today.getSeconds();  getHours() – Provides current hour between 0-23.  getMinutes() – Provides current minutes between 0-59.  getSeconds() – Provides current seconds between 0-59. |

* **Use the following script to get the current date and time** using JavaScript in the “Y-m-d H:i:s” format. You can simply combine the output of the above JavaScript code in one variable as below:

|  |
| --- |
| var today = new Date();  var date = today.getFullYear()+'-'+(today.getMonth()+1)+'-'+today.getDate();  var time = today.getHours() + ":" + today.getMinutes() + ":" + today.getSeconds();  var dateTime = date+' '+time;  console.log(dateTime) |

1. **Explain JavaScript Math object?**

* **The JavaScript math object** provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors. In JavaScript, Math is a built-in object that allows you to perform mathematical operations on the Number type. Math is not a constructor function. It's a property of the implicit global object. You can access methods and constants of the Math object directly. For example, to use the mathematical PI constant, use Math.PI in your code.
* **JavaScript Math Methods:** Let's see the list of JavaScript Math methods with description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [**log()**](https://www.javatpoint.com/javascript-math-log-method) | It returns natural logarithm of a number. |
| [**trunc()**](https://www.javatpoint.com/javascript-math-trunc-method) | It returns an integer part of the given number. |
| [**sqrt()**](https://www.javatpoint.com/javascript-math-sqrt-method) | It returns the square root of the given number |
| [**tan()**](https://www.javatpoint.com/javascript-math-tan-method) | It returns the tangent of the given number. |

* **Math.sqrt(n):** The JavaScript math.sqrt(n) method returns the square root of the given number.

|  |
| --- |
| Square Root of 17 is: <span id="p1"></span>  <script>  document.getElementById('p1').innerHTML=Math.sqrt(17);  </script>  **Output:** Square Root of 17 is: 4.123105625617661 |

* **JavaScript Number Methods:** Let's see the list of JavaScript number methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [**isFinite()**](https://www.javatpoint.com/javascript-number-isfinite-method) | It determines whether the given value is a finite number. |
| [**isInteger()**](https://www.javatpoint.com/javascript-number-isinteger-method) | It determines whether the given value is an integer. |
| [**toPrecision()**](https://www.javatpoint.com/javascript-number-toprecision-method) | It returns the string representing a number of specified precision. |
| [**toString()**](https://www.javatpoint.com/javascript-number-tostring-method) | It returns the given number in the form of string. |

* **JavaScript Boolean: JavaScript Boolean** is an object that represents value in two states: true or false. You can create the JavaScript Boolean object by Boolean() constructor as given below. Boolean b=new Boolean(value);  The default value of JavaScript Boolean object is false.

|  |
| --- |
| **JavaScript Boolean Example:**  <script>   document.write(10<20); //true     document.write(10<5); //false  </script> |

1. **What is regular expression in JS, Explain with syntax?**

* **A regular expression**is a sequence of characters that forms a search pattern. The search pattern can be used for text search and text to replace operations. A regular expression can be a single character or a more complicated pattern. Regular expressions can be used to perform all types of text search and text replacement operations. The RegExp Object is a regular expression with added Properties and Methods.
* **Syntax:** /pattern/modifiers;
* **Example:** let patt = /I AM AI & DS Student/i;
* **Explanation:** I AM AI & DS Student/i is a regular expression. I AM AI & DS Student is the pattern (to be used in a search). I is a modifier (modifies the search to be Case-Insensitive).
* **Modifiers:** Modifiers are used to perform case-insensitive and global searches:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| [**g**](https://www.w3schools.com/jsref/jsref_regexp_g.asp) | Perform a global match (find all matches rather than stopping after the first match) |
| [**i**](https://www.w3schools.com/jsref/jsref_regexp_i.asp) | Perform case-insensitive matching |
| [**m**](https://www.w3schools.com/jsref/jsref_regexp_m.asp) | Perform multiline matching |

* **Brackets:** Brackets are used to find a range of characters:.

|  |  |
| --- | --- |
| **Expression** | **Description** |
| [**[abc]**](https://www.w3schools.com/jsref/jsref_regexp_charset.asp) | Find any character between the brackets |
| [**[^abc]**](https://www.w3schools.com/jsref/jsref_regexp_charset_not.asp) | Find any character NOT between the brackets |
| [**[0-9]**](https://www.w3schools.com/jsref/jsref_regexp_0-9.asp) | Find any character between the brackets (any digit) |
| [**[^0-9]**](https://www.w3schools.com/jsref/jsref_regexp_not_0-9.asp) | Find any character NOT between the brackets (any non-digit) |
| [**(x|y)**](https://www.w3schools.com/jsref/jsref_regexp_xy.asp) | Find any of the alternatives specified |

* **Metacharacters:** Metacharacters are characters with a special meaning:

|  |  |
| --- | --- |
| **Metacharacter** | **Description** |
| [**.**](https://www.w3schools.com/jsref/jsref_regexp_dot.asp) | Find a single character, except newline or line terminator |
| [**\w**](https://www.w3schools.com/jsref/jsref_regexp_wordchar.asp) | Find a word character |
| [**\W**](https://www.w3schools.com/jsref/jsref_regexp_wordchar_non.asp) | Find a non-word character |
| [**\d**](https://www.w3schools.com/jsref/jsref_regexp_digit.asp) | Find a digit |
| [**\D**](https://www.w3schools.com/jsref/jsref_regexp_digit_non.asp) | Find a non-digit character |
| [**\s**](https://www.w3schools.com/jsref/jsref_regexp_whitespace.asp) | Find a whitespace character |
| [**\S**](https://www.w3schools.com/jsref/jsref_regexp_whitespace_non.asp) | Find a non-whitespace character |
| [**\b**](https://www.w3schools.com/jsref/jsref_regexp_begin.asp) | Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b |
| [**\B**](https://www.w3schools.com/jsref/jsref_regexp_begin_not.asp) | Find a match, but not at the beginning/end of a word |
| [**\0**](https://www.w3schools.com/jsref/jsref_regexp_nul.asp) | Find a NULL character |
| [**\n**](https://www.w3schools.com/jsref/jsref_regexp_newline.asp) | Find a new line character |
| [**\f**](https://www.w3schools.com/jsref/jsref_regexp_formfeed.asp) | Find a form feed character |
| [**\r**](https://www.w3schools.com/jsref/jsref_regexp_carriagereturn.asp) | Find a carriage return character |
| [**\t**](https://www.w3schools.com/jsref/jsref_regexp_tab.asp) | Find a tab character |
| [**\v**](https://www.w3schools.com/jsref/jsref_regexp_vtab.asp) | Find a vertical tab character |
| [**\xxx**](https://www.w3schools.com/jsref/jsref_regexp_octal.asp) | Find the character specified by an octal number xxx |

* **Quantifiers:**

|  |  |
| --- | --- |
| **Quantifier** | **Description** |
| [**n+**](https://www.w3schools.com/jsref/jsref_regexp_onemore.asp) | Matches any string that contains at least one n |
| [**n\***](https://www.w3schools.com/jsref/jsref_regexp_zeromore.asp) | Matches any string that contains zero or more occurrences of n |
| [**n?**](https://www.w3schools.com/jsref/jsref_regexp_zeroone.asp) | Matches any string that contains zero or one occurrences of n |
| [**n{X}**](https://www.w3schools.com/jsref/jsref_regexp_nx.asp) | Matches any string that contains a sequence of X n's |
| [**n{X,Y}**](https://www.w3schools.com/jsref/jsref_regexp_nxy.asp) | Matches any string that contains a sequence of X to Y n's |
| [**n{X,}**](https://www.w3schools.com/jsref/jsref_regexp_nxcomma.asp) | Matches any string that contains a sequence of at least X n's |
| [**n$**](https://www.w3schools.com/jsref/jsref_regexp_ndollar.asp) | Matches any string with n at the end of it |
| [**^n**](https://www.w3schools.com/jsref/jsref_regexp_ncaret.asp) | Matches any string with n at the beginning of it |
| [**?=n**](https://www.w3schools.com/jsref/jsref_regexp_nfollow.asp) | Matches any string that is followed by a specific string n |
| [**?!n**](https://www.w3schools.com/jsref/jsref_regexp_nfollow_not.asp) | Matches any string that is not followed by a specific string n |

* **RegExp Object Properties:**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [**constructor**](https://www.w3schools.com/jsref/jsref_regexp_constructor.asp) | Returns the function that created the RegExp object's prototype |
| [**global**](https://www.w3schools.com/jsref/jsref_regexp_global.asp) | Checks whether the "g" modifier is set |
| [**ignoreCase**](https://www.w3schools.com/jsref/jsref_regexp_ignorecase.asp) | Checks whether the "i" modifier is set |
| [**lastIndex**](https://www.w3schools.com/jsref/jsref_regexp_lastindex.asp) | Specifies the index at which to start the next match |
| [**multiline**](https://www.w3schools.com/jsref/jsref_regexp_multiline.asp) | Checks whether the "m" modifier is set |
| [**source**](https://www.w3schools.com/jsref/jsref_regexp_source.asp) | **Returns the text of the RegExp pattern** |

* **RegExp Object Methods:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [**compile()**](https://www.w3schools.com/jsref/jsref_regexp_compile.asp) | Deprecated in version 1.5. Compiles a regular expression |
| [**exec()**](https://www.w3schools.com/jsref/jsref_regexp_exec.asp) | Tests for a match in a string. Returns the first match |
| [**test()**](https://www.w3schools.com/jsref/jsref_regexp_test.asp) | Tests for a match in a string. Returns true or false |
| [**toString()**](https://www.w3schools.com/jsref/jsref_regexp_tostring.asp) | Returns the string value of the regular expression |

1. **Describe Timing Events & async/await function in JavaScript?**

* **Timing Events:** The window object allows execution of code at specified time intervals. These time intervals are called timing events. The two key methods to use with JavaScript are:
* setTimeout(function, milliseconds). Executes a function, after waiting a specified number of milliseconds. setInterval(function, milliseconds).
* Same as setTimeout(), but repeats the execution of the function continuously.
* The setTimeout() and setInterval() are both methods of the HTML DOM Window object.
* **Timing Events: Stop the Execution:** The clearTimeout() method stops the execution of the function specified in setTimeout(). **window.clearTimeout(timeoutVariable)**

|  |
| --- |
| **Example:**  myVar = setTimeout(function, milliseconds);  clearTimeout(myVar); |

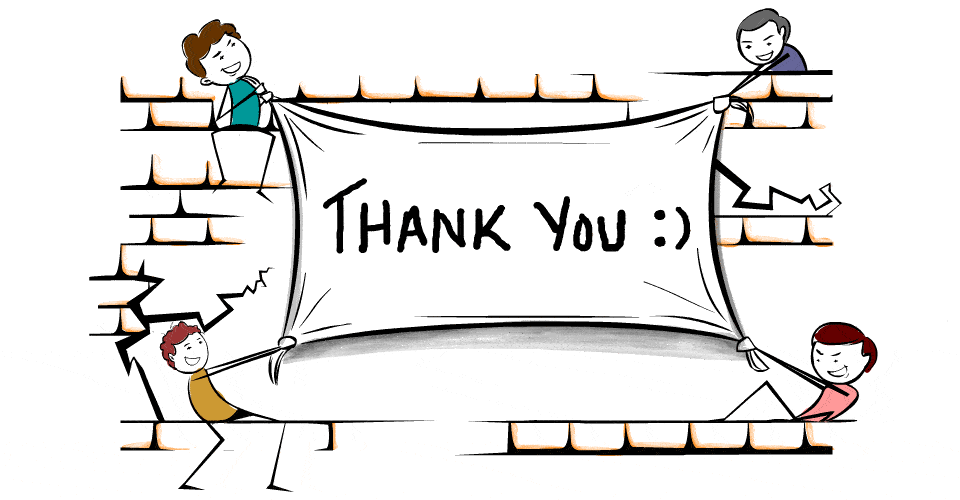
* **Javscript async/await:** We use the async keyword with a function to represent that the function is an asynchronous function. The async function returns a [promise](https://www.programiz.com/javascript/promise). The syntax of async function is:

|  |
| --- |
| async function name(parameter1, parameter2, ...paramaterN) { // statements }   * Here, **name**: name of the function. **Parameters**: parameters that are passed to the function |
| * **Here is how to use the Promise:**   myFunction().then  (   function(value) { /\* code if successful \*/ },   function(error) { /\* code if some error \*/ } ); |

* **Await Syntax:** The await keyword can only be used inside an async function. The await keyword makes the function pause the execution and wait for a resolved promise before it continues: **let value = await promise;**

|  |
| --- |
| **Basic Syntax :**  async function myDisplay() {   let myPromise = new Promise(function(resolve, reject) {     resolve("I love You !!");   });   document.getElementById("demo").innerHTML = await myPromise; } myDisplay(); |

**(PPT Give you more understanding than PDF)** The material for the PDF has been compiled from various sources such as books, tutorials (offline and online), lecture notes, several resources available on Internet. The information contained in this PDF is for general information and education purpose only. While we endeavor to keep the information up to date and correct, **we make no representation of any kind about the completeness and accuracy of the material.** The information shared through this PDF material should be used for educational purpose only.

****