|  |  |  |
| --- | --- | --- |
| JAVASCRIPT Documentation | | |
|  | | |
| TOPIC | EXPLAINATION |
| 1. String concatenation using “+” operator | With the help of “+” operator two strings can be combined.  Example:  let firstName = "Arijit"  let lastName = "Deb"  let fullName = firstName +" "+ lastName  console.log(fullName)  Output: Arijit Deb |
| 1. String concatenation using concat() function | The concat() method concatenates the string arguments to the calling string and returns a new string. Takes the items to be concatenated as argument separated by comas.  Example:  let firstName = "Arijit"  let lastName = "Deb"  let fullName = firstName.concate(“ ”, lastName)  console.log(fullName)  Output: Arijit Deb |
| 1. Template String | Helps to concat strings without ‘+’ operator.  Example:  let name = “Arijit”  let age =37  console.log(`${name} is a bad boy. His age is ${age} years. `)  Output: Arijit is a bad boy. His age is 37 years. |
| 1. Rules regarding variable names | 1. Can’t define a variable more than once. So, there will be error if:  let name = “Auritra”  let name = “Arijit”  2. Numbers can’t be variable name. So, it is not possible to have:  let 2 = 3  3. Variable names can only be a word or “$” sign before or after the word, or “-” sign before or after the word or word with a number without spacing.  4. Variable names cannot be reserved keywords. |
| 1. === | Equality operator.  \*Can be used for all data types |
| 1. !== | Not equality operator  \*Can be used for all data types |
| 1. < | Less than operator.  \*Can be used for numeric data types |
| 1. <= | Less than equal to operator.  \*Can be used for numeric data types |
| 1. > | Greater than operator.  \*Can be used for numeric data types |
| 1. >= | Greater than equal to operator.  \*Can be used for numeric data types |
| 1. If (true) {} | Code will always execute |
| 1. If (false) {} | Code will never execute |
| 1. Scoping in Javascript | 1. Lexical scoping (Static scoping)  2. There are two types of scope – global scope and local scope.  3. Global scope contains all the things outside of all the code blocks.  4. Local scope contains all the things inside a code block. |
| 1. Scoping Rules | 1. In a scope, you can access variables defined within the scope or in any parent/ancestor scope.  Example:  let name1 = “Arijit”  if (true) {  let name2 = “Auritra”  if (true) {  let name3 = “Abhi”  console.log(name1)  console.log(name2)  console.log(name3)  }  }  It is possible to access all the variables name1(defined in parent scope which is here the global scope) and name2 (defined inside the parent local scope) and name3 (defined inside the child local scope).  2. Scope of same level (two independent scope) cannot access variables of each other.  Example:  if (true) {  let name1 = “Arijit”  }  if (true) {  let name2 = “Auritra”  }  Here name1 cannot be accessed from the second if statement and name2 cannot be accessed from the first if statement.  3. Variables with same name can be defined in different scopes.  Example:  if (true) {  let name1 = “Arijit”  }  if (true) {  let name1 = “Auritra”  }  Here name1 is defined more than once in different scopes.  4. The value of a variable defined in parent scope can be changed in the child scope. This is known as variable shadowing in JavaScript.  Example:  let name = “Arijit”  if (true) {  let name = “Auritra”  if (true) {  console.log(name)  }  }  Output: Auritra  5. If a variable is not defined locally, it will search in the parent scope (if any) and if it is not found in the parent scope, it will search in the global scope. If the variable is not defined there also, it will automatically create a global variable. This is known as global leaking.  Example:  if (true) {  if (true) {  name = “Auritra”  console.log(name)  }  }  Output: Auritra  Here name is not declared using “let” but still the output is “Auritra” instead of an error. This happens due to global leaking. |
| 1. Function without argument | Defining:  let functionName = function () {  console.log (“Hello World”)  }  Calling:  functionName ()  Output:  Hello World |
| 1. Function with argument | Defining:  let functionName =function(num) {  console.log (num)  }  Calling:  functionName(3)  Output:  3 |
| 1. Function with return | Defining:  let squareNumber =function(num) {  let square = num\*num  return square  }  Calling:  console.log(squareNumber(3))  Output:  9 |
| 1. Objects | Collection of related data of primitive or reference/object data type in form of key-value pair.  Example:  let person = {  name: “Arijit”,  age: 37,  address: {  country: “Singapore”,  pin: 460208  }  }  Calling:  Console.log(person)  Output:  { name: “Arijit”, age: 37, address: {country: “Singapore”, pin: 460208}}  Calling:  Console.log(person.name)  Output:  Arijit  Calling:  Console.log(`${person.name} is ${person.age} years old.`)  Output:  Arijit is 37 years old. |
| 1. Passing Objects within functions | Example:  let book1 = {  title : “ABC”,  author: “Arijit”  }  let book2 = {  title : “XYZ”,  author: “Auritra”  }  let getDetails = function(book) {  console.log(`${book.title} by ${book.author}`)  }  Calling:  getDetails(book1)  getDetails(book2)  Output:  ABC by Arijit  XYZ by Auritra |
| 1. Functions returning Object | Example1:  let book1 = {  title : “ABC”,  author: “Arijit”,  page: 100  }  let book2 = {  title : “XYZ”,  author: “Auritra”,  page: 200  }  let getDetails = function(book) {  return {  details:`${book.title} by ${book.author}`  pageCount : `The ${book.title} is ${book.page} pages long`  }  }  Calling:  console.log(getDetails(book1))  Output:  { details: ABC by Arijit, pageCount: The ABC is 100 pages long.}  Example2:  let book1 = {  title : “ABC”,  author: “Arijit”,  page: 100  }  let book2 = {  title : “XYZ”,  author: “Auritra”,  page: 200  }  let getDetails = function(book) {  return {  details:`${book.title} by ${book.author}`  pageCount : `The ${book.title} is ${book.page} pages long`  }  }  Calling:  let book1Summary= getDetails(book1)  console.log(book1Summary.pageCount)  Output:  The ABC is 100 pages long. |
| 1. Functions as an attribute of an Object/ Object Methods | A method is just a object property whose value/attribute is a function.  Example:  let restaurant = {  name: “ABC”,  capacity: 100,  guestCount: 100,  checkAvailability: function(partySize) {  let seatsAvailable = this.capacity-this.guestCount  return seatsAvailable>=partySize  }  }  let status = restaurant.checkAvailability(4)  console.log(status)  Output:  True  Note:  “this.” inside the checkAvailability function refers to the object restaurant. |
| 1. Array | JavaScript arrays are used to store multiple values in a single variable. |
| 1. Array properties in JS | 1.arrayName.length – Returns/set the array length. |
| 1. Array manipulating methods in JS | 1. arrayName.toString() – Converts an array to a string of (comma separated) array values.  2. arrayName.join(“eparator”) – Joins all array elements into a string and in addition you can specify the separator. Seperator can be “\*”, “,”, “/’, “:” etc  3. arrayName.pop() – Removes the last element from an array. Returns the value that was “popped out”.  4. arrayName.push() – Adds a new element to an array (at the end). Returns the new array length.  5. arrayName.shift() – Removes the first array element and “shifts” all other elements to a lower index. Returns the string that was “shifted out”  6. arrayName.unshift() – Adds a new element to an array (at the beginning. Returns the new array length.  7. arrayName.splice(n,k, “item1”, “item2”,…., “itemn”) – Add/remove items to/from an array at a same time. The first parameter (n) defines the position where splice should be introduced. The second parameter (k) defines number of elements to be removed (If k>0, it will remove first k number of elements which are to the right of splice position(n)) . The rest of the parameters (“item1,.., “itemn”) define the new elements to be added to the array starting from nth position. This method returns an array with the deleted items.  8. arrayName.splice(n,k) – Remove items from an array from any position. The first parameter (n) defines the position where splice should be introduced. The second parameter (k) defines number of elements to be removed (If k>0, it will remove first k number of elements which are to the right of splice position(n)). Remove elements without leaving “holes” in the array.  9.arrayName.splice(n, k) – Slices out a piece of an array into a new array. It does not remove any elements from the source array. It starts slicing the array from nth position (first argument) upto kth position (excluding kth element). If the second argument is omitted, the slice() method slices out the rest of the array.  10. array1.concat(array2) – Merge two arrays. It always returns a new array and does not change the existing arrays.  11. array1.concat(array2,…..arrayn) – Merge multiple arrays. It always returns a new array and does not change the existing arrays. |
| 1. Array iteration methods in JS | 1. forEach()  arrayName.forEach(functionName(value, index){ }).   * currentValue – Required. The value of the current element * index – Optional. The array index of the current element   2. for()  3. while()  4. Extended for loop  5. map()  6. reduce()  7. every() |
| 1. Array element searching methods in JS | 1. arrayName.indexOf(“element”) – Searches an array for an element value and returns its position. Returns -1 if the value to search for never occurs. It is case sensitive.  2. arrayName.find (function(value){ }) – Returns the value of the first array element that passes a test function.  3. arrayName.findIndex(function(value,index){ }) – Returns the index of the first array element that passes a test function. Otherwise it returns -1. Does not execute the function for array elements without values. |
| 1. Array sorting methods in JS | 1. arrayName.sort() – Sorts an array alphabetically.  2. arrayName.sort(compareFunction()) – Specifies a function that defines the sort order. If omitted, the array elements are converted to strings, then sorted according to each character's Unicode code point value. Returns the sorted array. The array is sorted in place, and no copy is made. Return-type (-1) implies no swapping. Return-type (1) implies swapping.  arrayName.sort(function(a, b){return a - b}) – Sorts an array numerically. |
| 1. Array filtering methods in JS | 1. arrayName.filter(function(element){ }) – Creates an array filled with all array elements that pass a test (provided as a function. Does not execute the function for array elements without values. Does not change the original array. |
| 1. Object and Array reference/ address in JS | The value assigned to a variable In JS decides whether it is stored with assign-by-value or assign-by-reference.  Number, String, Boolean, Undefines, Null, Symbol are assign-by-value and hence are not changeable/immutable.  Example:  let num1= 7  let num2 = num1  num1++  console.log(num1)  console.log(num2)  Output:  8  7  Array and Objects are assigned-by-reference and hence are changeable/mutable.  Example1:  let array1 =[8,8,8]  let array2 =array1  array1.push(1)  console.log(array1)  console.log(array2)  Output:  [8,8,8,1]  [8,8,8,1]  Example 2:  let myAccount = {  income: 100,  initialExpenditure:0  }  let expenseTracker = function(account, newExpenditure){  let amountSpend = account.initialExpenditure  amountSpend = amountSpent + newExpenditure  return account  }  Let modifiedMyAccount = expenseTracker(myAccount, 25)  console.log(modifiedMyAccount)  console.log(myAccount)  Output:  {income: 100, initialExpenditure:25}  {income: 100, initialExpenditure:25} |
| 1. String Methods In JS | 1. stringName.length - Gives the size of the string  2. stringName.toUpperCase() – Converts a string to upper case  3. stringName.toLowerCase() – Converts a string to lower case  4. stringName.includes(“searchString”) – Determines whether a string can be found within another string. Returns boolean.  5. stringName.includes(“searchString”, position) – Determines whether a string can be found within another string. Returns boolean. The second argument is optional which specify the position within the string at which to begin searching for searchString. (Defaults to 0.)  6. stringName.trim() – Removes whitespace from both ends of a string. |
| 31. Number Methods In JS | 1. varName.toFixed(n), n=0 to 20 – Convert a decimal number up to nth decimal point |
| 32. Math Methods In JS | 1. Math.round(num) – Round up the number using mathematics rule.  2. Math.floor(num) – Erase the decimal part  3. Math.ceil(num) – Convert number to nearest integer  4. Math.random() – Generates a random number between 0 and 1(0 inclusive and 1 exclusive).  5. Math.floor(Math.ramdom()\*(max-min+1)) + min – Generates a random number between max and min number(say between 20 and 10) |
| 33. Constant Variable | If assigned, then value cannot be changed (will show error) but it can be manipulated in case of object.  Example 1:  const num =2  num = 3  console.log(num)  Output:  Error  Example 2:  const person = {  name: “Arijit”,  age: 35  }  person= {  name: “Auritra”,  age: 30  }  console.log(person)  Output:  Error  Example 3:  const person = {  name: “Arijit”,  age: 35  }  person.name= “Auritra”  console.log(person.name)  Output:  Auritra |
| 34. Installing live server | For Dos/Windows: npm install -g live-server  For Mac/Apple: sudo npm install -g live-server |
| 35. Check live server version | live-server --version |
| 36. Serving a particular project on live-server | cd projectDirectory  cd ..  live-server projectDirectory --port=“8090” |
| 37. Connecting JS with HTML | Crude way:  <body>  <script>*any JS code*</script>  </body>  Actual way:  <body>  <script src=”/path from current location/fileName.js”> </script>  </body>  Note:  Script tag should be inserted inside the body tag at the end. If not then all html codes below the script tag will not be available to JS in spite of the presence. |
| 38. Targeting HTML element using JS code | document.queryselector(“tagName”). For example, tagName can be p for paragraph tag  querySelector() targets only the first tag with same tags in HTML. |
| 39. Removing HTML element using JS code | Removing single tag:  let tag= document.queryselector(“tagName”)  tag.remove()  Removing multiple tags of same type:  let tags= document.queryselectorAll(“tagName”)  tags.foreach(function(tagName) {  tagName.remove()  }) |
| 40. Reading HTML element using JS code | Reading single tag:  let tag= document.queryselector(“tagName”)  tag.textContent  Reading multiple tags of same type:  let tags= document.queryselectorAll(“tagName”)  tags.foreach(function(tagName){  console.log(tagName.textContent)  }) |
| 41. Updating HTML elements using JS codes | Updating single tag:  let tag= document.queryselector(“tagName”)  tag.textContent =”\*\*\*\*\*\*\*\*”  Updating multiple tags of same type:  let tags= document.queryselectorAll(“tagName”)  tags.foreach(function(tagName){  tagName.textContent = “\*\*\*\*\*\*”  }) |
| 42. Adding HTML elements through JS | Comprises of three steps:  1.Telling JS what type of element you want to create  let newElement= document.createElement(“tagType”)  2.Entering the content for the element  newElement.textContent= “abcdefgh”  3.Positioning the element in JS  For this we need to select the tag in HTML where we want to place the element. By default, it will be added inside, at the end of the selected tag.  document.querySelector(“tagName”).appendChild(newElement) |
| 43. Adding Event Listener to any DOM elements | Attaches an event handler to the specified element. Many event handlers of the same type can be added to one element.  By id:  document.querySelector(“#id”).addEventListener(event,function(e){*defn*})  By class:  document.querySelectorAll(“.className”).addEventListener(event,function(e){*defn*}) |
| 44. Rendering filtered data | Steps:  1. Create a filter object with a single attribute and keep that empty to store latest filtered object –  const filteredObj = {  search : “”  }  2. Create a render function which includes 3 steps:  const renderFunction = function(arrayName, filteredObj){  **JS filter function to filter elements**  const filteredData = arrayName.filter(function(arrayElement){  return arrayElement.include(filteredObject.search)  })  **Removing duplicity in displaying notes**  document.querySelector(“#<div> id”).innerHTML= “”  **forEach method on filteredData array to display the filtered data to HTML.**  filteredData.forEach(function(arrayElement){  const element = document.createElement(“p”)  element.textContent = arrayElement  document.querySelector(“#<div> id”).appendChild(element)  })  }  Note:  For the first time (before entering any input in the input tag) the renderFunction will render the entire array to HTML as filterObj.search= “”.  3. Call the render function to execute all the functions inside the render function.  4. Again call this render function from inside the input-eventListener function to display only the array elements which will pass the filter |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |