|  |  |  |
| --- | --- | --- |
| 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: 95,  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. 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. |
|  |  |
|  |  |
|  |  |
| 1. Installing live server | For Dos/Windows: npm install -g live-server  For Mac/Apple: sudo npm install -g live-server |
| 1. Check live server version | live-server --version |
| 1. 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. |
| 1. 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. |
| 1. 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”)  let removeTags= function(tagName){  tagName.remove()  }  tags.foreach(removeTags) |
| 1. Removing 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”)  let readTags= function(tagName){  console.log(tagName.textContent)  }  tags.foreach(readTags) |
| 35. 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”)  let updateTags= function(tagName){  tagName.textContent = ”\*\*\*\*\*\*”  }  tags.foreach(updateTags) |