Cheatsheet: Introduction to JavaScript Development

JavaScript Tag and Terminologies	Description	Code Example

<script>	Used to include the required JavaScript code in your HTML document.	`<body>`
		`  <p id="showname"></p>`
		`  <script>`
		`  document.getElementById('showname').innerHTML='Peter';`
		`  </script>`
		`</body>`

<script src>	Used to link the required JavaScript files in your HTML document.	`<script src="script.js"></script>`
var	var is a keyword used to declare variables.	`var num1=10;` `var num2=11;`

var & Scope	var has functional scope, allowing variable to be accessed within function only.	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`  <meta charset="UTF-8">`
		`  <meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`  <title>Document</title>`
		`</head>`
		`<body>`
		`  <p id="showname"></p>`
		`  <script>`
		`  function show() {`
		`  var name = 'Peter';`
		`  document.getElementById('showname').innerHTML = name;`
		`  }`
		`  </script>`
		`</body>`
		`</html>`

let	let is a keyword used to declare variables.	`let num1=20;` `let num2=21;`

let & Scope	let has block scope, allowing the variable to be limited to the block, statement,	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`  <meta charset="UTF-8">`
		`  <meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`  <title>Document</title>`
		`</head>`

	or expression in which it is defined, preventing redeclaration within the same scope.	<body>
		
		<script>
		{
		let emailId = 'test@example.com';
		document.getElementById('showemail').innerHTML = emailId;
		}
		</script>
		</body>
		</html>

const	const is a keyword used to declare variables.	const employeeId=120; cont employeeId=121;

const & Scope	It creates a constant whose value cannot be reassigned or redeclared.	<!DOCTYPE html>
		<html lang="en">
		<head>
		<meta charset="UTF-8">
		<meta name="viewport" content="width=device-width, initial-scale=1.0">
		<title>Document</title>
		</head>
		<body>
		
		<script>
		{
		const employeeId = 120';
		document.getElementById('showeEId').innerHTML = employeeId;
		}
		</script>
		</body>
		</html>

Arithmetic Operators	Arithmetic operators perform mathematical calculations like addition, subtraction, multiplication, division and modulus.	let x = 15;
		let y = 3;
		let sum = x + y; // Addition
		console.log(sum) //the answer is 8
		let difference = x - y; // Subtraction
		console.log(difference) //the answer is 2
		let product = x * y; // Multiplication
		console.log(product) //the answer is 8
		let quotient = x / y; // Division
		console.log(quotient) //the answer is 8
		let remainder = x % y; // Modulus
		console.log(remainder) //the answer is 0

Comparison Operators	Comparison operators compare values and return true/false based on the comparison.	let a = 5;
		let b = 7;
		let isEqual = a == b; // Equality
		let isNotEqual = a != b; // Inequality
		let isStrictEqual = a === b; // Strict equality
		let isGreaterThan = a > b; // Greater than

Logical Operators	Logical operators combine multiple conditions and return a boolean result.	`let hasPermission = true;`
		`let isMember = false;`
		`let canAccessResource = hasPermission && isMember; // Logical AND`
		`let canViewPage = hasPermission || isMember; // Logical OR`
		`let isDenied = !hasPermission; // Logical NOT`

Assignment Operators	Assignment operators assign values to variables. For example, =, +=, -=.	`let x = 10; // Assigns the value 10 to the variable x`
		`x += 5; // Equivalent to x = x + 5`
		`x -= 5; //Equivalent to x = x + 5`

Unary Operators	Unary operators act on a single operand, performing operations like negation or incrementing.	`let count = 5;`
		`count++; // Increment count by 1 (count is now 6)`
		`count--; // Decrement count by 1 (count is now 5 again)`

typeof Operator	typeof operator returns the data type of a variable or expression as a string.	`let num1 = 42;`
		`console.log(typeof(num1)); //the awnswer is Number`
		`let name = 'John';`
		`console.log(typeof(name)); //the awnswer is String`

if Statement	The if statement is used to execute a piece of block code if the given condition is true.	`let age = 25;`
		`if (age >= 18) {`
		`console.log("You are an adult.");`
		`} else {`
		`console.log("You are a minor.");`
		`}`

else if Statement	It allows you to test multiple conditions sequentially.If the condition is true then it will execute if statement block otherwise execute else statement block.	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`<meta charset="UTF-8">`
		`<meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`<title>Document</title>`
		`</head>`
		`<body>`
		`<p id="seasonmessage"></p>`
		`<script>`
		`let Seasonmonth = 'March to May';`
		`if (Seasonmonth == 'March to May') {`
		`document.getElementById("seasonmessage") = 'It is spring season';`
		`}`
		`else if (Seasonmonth == 'June to August') {`
		`document.getElementById("seasonmessage") ='It is summer season';`
		`}`
		`else if (Seasonmonth =='September to November') {`
		`document.getElementById("seasonmessage") = 'It is autumn season';`

		``
		        }
		        else {
		            document.getElementById("seasonmessage") = 'It is winter season';
		        }
		    </script>
		</body>
		</html>```

Nested if else Statement	This statement allows you to test multiple conditions and execute different blocks of code based on the results of those conditions.	```const temperature = 30;
		const isRaining = true;
		if (temperature > 30) {
		  if (isRaining) {
		    console.log("It's hot and raining. Stay inside.");
		  } else {
		    console.log("It's hot, but not raining. Enjoy the sunshine.");
		  }
		} else {
		  if (isRaining) {
		    console.log("It's not so hot, but it's raining. Take an umbrella.");
		  } else {
		    console.log("It's not hot, and it's not raining. Have a nice day.");
		  }
		}```

switch Statement	The switch statement is used for multiple conditional branches, allowing the execution of different code blocks based on the value of an expression.	```let month = "December";
		switch (day) {
		    case "December":
		        console.log("It's Christmas month.");
		        break;
		    case "November":
		        console.log("It's Thanksgiving month");
		        break;
		    default:
		        console.log("It's a regular month.");
		}```

Ternary Operator	The ternary operator is the simplest way to write conditional statements such as if else condition.	```let age = 20; let canVote = age >= 18 ? "Yes" : "No";```

for loop	A for loop is a control structure that allows to execute a block of code repeatedly for a specified number of times until a particular condition is met.	```for (let i = 1; i <= 5; i++) {
		    console.log(i);
		}```

While loop	A while loop is a control structure that allows to execute a block of code repeatedly as	```let limit = 50;
		let a = 0;
		let b = 1;
		while (a <= limit) {
		    console.log(a);
		    let temp = a + b;```

	long as a specified condition is true.	`a = b;`
		`b = temp;`
		`}`

do while loop	A "do...while" loop in allows you to execute a block of code repeatedly as long as a specified condition is true and guarantees that the code block will execute at least once, even if the condition is initially wrong.	`let roll = 1;`
		`do {`
		`console.log("Rolled a " + roll);`
		`roll++;`
		`} while (roll < 7);`

Function Declaration and Call	Function is a reusable block of code that can be defined and executed as many times as needed.	`function sayHello() {`
		`console.log("Hello!");`
		`} //function declaration`
		`sayHello();  //function call`

Non-Parameterized Functions	The functions that do not require any parameters to operate.	`function greet() {`
		`const greeting = "Hello, World!";`
		`console.log(greeting);`
		`}`
		`// Call the non-parameterized function`
		`greet(); // This will print "Hello, World!" to the console`

Parameterized Functions	The function that accepts one or more values that provide input data for the function to work with. These values in the function's declaration called parameters, and during calling of the function called arguments.	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`<meta charset="UTF-8">`
		`<meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`<title>Document</title>`
		`</head>`
		`<body>`
		`<p id="functiondata1"></p>`
		`<script>`
		`function add(a, b) {`
		`return a + b;`
		`}`
		`document.getElementById('functiondata1').innerHTML = add(3, 4);`
		`</script>`
		`</body>`
		`</html>`

Named Function	The functions with a specific name that can be called by that name.	`const add = function(a, b) {`
		`console.log(a+b);`
		`}`
		
		`//name of the function is add`
		`add(2, 3);`

IIFE	Immediately Invoked Function Expression is a function in JavaScript that's defined and executed immediately after its creation.	`(function sayWelcome() {`
		`  console.log("Welcome!");`
		`})();`

Arrow Function	Arrow functions in JavaScript are a concise way to write function expressions, using the => syntax.	`const arrowFunc = (a, b) => a + b;` `console.log(arrowFunc(5, 3));`

return	The return statement in JavaScript is used to end the execution of a function and specify the value that the function should return to the caller.	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`  <meta charset="UTF-8">`
		`  <meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`  <title>Document</title>`
		`</head>`
		`<body>`
		`  <p id="showmessage"></p>`
		`  <script>`
		`  function multiply(message) {`
		`  return message; // Returns the product of a and b`
		`  }`
		`  document.getElementById('showmessage').innerHTML = multiply('Hard work is the key');`
		`  </script>`
		`</body>`
		`</html>`

Function Closure	A function closure in JavaScript allows a function to access and remember variables from its outer scope even after that scope has finished executing.	`function outerFunction() {`
		`  const outerVar = "I am from the outer function";`
		`  function innerFunction() {`
		`  console.log(outerVar); // innerFunction can access outerVar`
		`  }`
		`  return innerFunction;`
		`}`
		`const closure = outerFunction();`
		`closure(); // This will log "I am from the outer function"`

Function Hoisting	Function hoisting means that function declarations are moved to the top of their containing scope during the compile phase, allowing them to be used before they are declared in the code.	`sayHello(); // This works even though the function is called before it's declared`
		`function sayHello() {`
		`  console.log("Hello!");`
		`}`

Function Hoisting for function expression	Function expressions where a function is assigned to a variable do not exhibit hoisting behaviour.	`greet(); // This will result in an error`
		`const greet = function() {`
		`  console.log("Greetings!");`
		`};`

addEventListener	addEventListener is a JavaScript method used to assign a function to execute when a specific event occurs on an element in the DOM.	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`  <meta charset="UTF-8">`
		`  <meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`  <title>Document</title>`
		`</head>`
		`<body>`
		`  <p id="btnclick"></p>`
		`  <button id="btn">Click Me</button>`
		`  <script>`
		`  // Get the element by its ID`
		`  const button = document.getElementById('btn');`
		`  // Add an event listener for the 'click' event`
		`  button.addEventListener('click', () => {`
		`  document.getElementById('btnclick').innerHTML = 'Button clicked!';`
		`  });`
		`  </script>`
		`</body>`
		`</html>`

onclick Event	A way of assigning a function directly to an HTML element to execute when it's clicked.	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`  <meta charset="UTF-8">`
		`  <meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`  <title>Document</title>`
		`</head>`
		`<body>`
		`  <button onclick="myFunction()">Click me</button>`
		`  <script>`
		`  function myFunction() {`
		`  alert('Button clicked!');`
		`  }`
		`  </script>`
		`</body>`
		`</html>`

Mouseover Event	The mouseover event is triggered when the mouse cursor enters an element.	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`  <meta charset="UTF-8">`
		`  <meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`  <title>Document</title>`
		`</head>`
		`<body>`
		`  <div id="myDiv" style="width: 200px; height: 200px; background-color: lightblue;"></div>`
		`  <script>`
		`  const myDiv = document.getElementById('myDiv');`
		`  // Adding a mouseover event listener`
		`  myDiv.addEventListener('mouseover', () => {`
		`  myDiv.style.backgroundColor = 'lightgreen';`
		`  });`
		`  </script>`
		`</body>`
		`</html>`

mouseout Event	The mouseout event in JavaScript is triggered when the mouse pointer moves out of an element, indicating that the mouse is no longer over that specific element.	`<!DOCTYPE html>`
		`<html lang="en">`
		`<head>`
		`<meta charset="UTF-8">`
		`<meta name="viewport" content="width=device-width, initial-scale=1.0">`
		`<title>Document</title>`
		`</head>`
		`<body>`
		`<div id="myDiv" style="width: 200px; height: 200px; background-color: lightblue;"></div>`
		`<script>`
		`const myDiv = document.getElementById('myDiv');`
		`// Adding a mouseover event listener`
		`myDiv.addEventListener('mouseover', () => {`
		`myDiv.style.backgroundColor = 'lightgreen';`
		`});`
		`myDiv.addEventListener('mouseout', () => {`
		`myDiv.style.backgroundColor = 'lightcoral';`
		`});`
		`</script>`
		`</body>`
		`</html>`

Keydown Event	The keydown event is triggered when a key on the keyboard is pressed down.	`<!DOCTYPE html>`
		`<html>`
		`<head>`
		`<title>Keydown Event Handling</title>`
		`</head>`
		`<body>`
		`<input type="text" id="myInput">`
		`<p id="output"></p>`
		`<script>`
		`const input = document.getElementById("myInput");`
		`const output = document.getElementById("output");`
		`input.onkeydown = function(event) {`
		`output.textContent = `Key pressed: ${event.key}`;`
		`};`
		`</script>`
		`</body>`
		`</html>`

Change Event	The change event is triggered when the value of an input element changes. Typically, it's used for form elements like text fields or dropdowns.	`<!DOCTYPE html>`
		`<html>`
		`<head>`
		`<title>Change Event Handling</title>`
		`</head>`
		`<body>`
		`<input type="text" id="myInput">`
		`<p id="output"></p>`
		`<script>`
		`const input = document.getElementById("myInput");`
		`const output = document.getElementById("output");`
		`input.onchange = function() {`
		`output.textContent = `Value changed to: ${input.value}`;`
		`};`
		`</script>`
		`</body>`
		`</html>`

onsubmit Event	The onsubmit event in HTML occurs when a form is submitted, either by clicking a submit button or by calling the submit().	`<!DOCTYPE html>` `<html>` `<head>` `<title>Form Submission Example</title>` `</head>` `<body>` `<form id="myForm" onsubmit="validateForm()">` `<label for="name">Name:</label>` `<input type="text" id="name" name="name">  ` `<label for="email">Email:</label>`

```
    <input type="email" id="email" name="email"><br><br>
    <input type="submit" value="Submit">
  </form>
  <script>
    function validateForm() {
      // Prevent the default form submission
      event.preventDefault();
      // Retrieve form values
      const name = document.getElementById('name').value;
      const email = document.getElementById('email').value;
      // Perform validation (for example, checking if fields are filled)
      if (name === '' || email === '') {
        alert('Please fill in all fields.');
        return false; // Prevent form submission if validation fails
      }
      // If validation passes, continue with form submission
      alert('Form submitted successfully!');
    }
  </script>
</body>
</html>
```