JS Basics

Introduction

JavaScript is a programming language that is used to add interactivity to web pages. It can be used to create things like games, animations, and interactive forms. JavaScript is a relatively easy language to learn, and it is a valuable skill for any web developer.

Characteristics of JavaScript:

- Change the appearance of a web page. You can use JavaScript to change the color of text, the size of images, and the layout of a web page.
- Add interactivity to a web page. You can use JavaScript to create games, animations, and interactive forms.
- Access and manipulate data. You can use JavaScript to access and manipulate data that is stored on the server or in the browser.
- Create web applications. You can use JavaScript to create web applications that can be used by users on their computers or mobile devices.

Where to add JS

JavaScript code can be written in the <script> element in the head or body of an HTML document. The <script> element can contain either inline JavaScript code or a reference to an external JavaScript file.

Inline JavaScript

Inline JavaScript is JavaScript code that is written directly in the <script> element. For example:

```
<script>
document.getElementById("myElement").innerHTML = "Hello, world!";
</script>
```

External JavaScript

External JavaScript is JavaScript code that is stored in a separate file. The file is typically named with the .js extension. To reference an external JavaScript file, use the src attribute of the <script> element. For example:

```
<script src="myscript.js"></script>
```

When to use inline JavaScript?

Inline JavaScript is a good choice for short, simple scripts that you want to keep close to the HTML code that they affect. For example, you might use inline JavaScript to change the text of an element or to make an element interactive.

When to use external JavaScript?

External JavaScript is a good choice for long, complex scripts or for scripts that you want to reuse on multiple pages. For example, you might use external JavaScript to create a game or to implement a form validation library.

Display JS

You can display JavaScript code execution in an HTML document using the console.log() method. The console.log() method is used to log messages to the console. The console is a tool that is used to debug JavaScript code.

For example, the following code will log the message "Hello, world!" to the console:

```
<script>
    console.log("Hello, world!");
</script>
```

When the browser loads the HTML document, it will also execute the JavaScript code. The console.log() method will then log the message "Hello, world!" to the console.

The console can be opened by pressing Ctrl+Shift+J (on Windows) or Cmd+Option+J (on Mac).

You can also use the console.log() method to log other types of data, such as variables, objects, and arrays. For example, the following code will log the value of the variable myVariable to the console:

```
<script>
    var myVariable = "Hello, world!";
    console.log(myVariable);
</script>
```

When the browser loads the HTML document, it will also execute the JavaScript code. The console.log() method will then log the value of the variable myVariable to the console.

The console is a powerful tool that can be used to debug JavaScript code. By logging messages to the console, you can track the execution of your code and identify any errors.

document.write()

The document.write() method is used to write text to the document. It is a deprecated method, which means that it is not recommended for use in new code. However, it is still supported by most browsers.

The document.write() method takes a single argument, which is the text that you want to write to the document. For example, the following code will write the text "Hello, world!" to the document:

```
document.write("Hello, world!");
```

The document.write() method can also be used to write HTML code to the document. For example, the following code will write a paragraph to the document:

```
document.write("<p>This is a paragraph.</p>");
```

The document.write() method can be used to write text or HTML code to the document at any time. However, it is important to note that the document.write() method will overwrite any existing content in the document.

For this reason, it is important to use the document.write() method with caution. It is best to use the document.write() method only when you are sure that you want to overwrite any existing content in the document.

As mentioned earlier, the document.write() method is a deprecated method. This means that it is not recommended for use in new code. There are better ways to write text and HTML code to the document. For example, you can use the innerHTML property.

The innerHTML property is used to set the inner HTML of an element. The inner HTML of an element is the text content of the element and its child elements.

For example, the following code will set the inner HTML of the element with the ID "myElement" to the text "Hello, world!":

```
document.getElementById("myElement").innerHTML = "Hello, world!";
```

The innerHTML property is a safer way to write text and HTML code to the document. It does not overwrite any existing content in the document.

JS Program

A JavaScript program is a set of instructions that are written in the JavaScript programming language. JavaScript programs are used to add interactivity to web pages. They can also be used to create web applications.

JavaScript programs are typically written in a text editor. The text editor can be any text editor, such as Notepad or Sublime Text.

Once the JavaScript program is written, it can be saved in a file with the .js extension. The .js extension tells the web browser that the file contains JavaScript code.

To run a JavaScript program, the web browser must have JavaScript enabled. JavaScript is enabled by default in most web browsers.

To add a JavaScript program to a web page, you can use the <script> element. The <script> element is used to embed JavaScript code in a web page.

The <script> element has two attributes:

src

The src attribute is used to specify the URL of the JavaScript file. For example, the following code specifies the URL of the file myscript.js:

```
<script src="myscript.js"></script>
```

type

The type attribute is used to specify the type of the JavaScript file. The type attribute must be set to "text/javascript". For example, the following code sets the type attribute to "text/javascript":

```
<script type="text/javascript" src="myscript.js"></script>
```

Once the JavaScript program is added to the web page, it will be executed by the web browser.

JS Statements:

A statement is a complete instruction that tells the JavaScript interpreter what to do. Statements can be simple, such as assigning a value to a variable, or they can be more complex, such as creating a function. There are many different types of statements in JavaScript. Some of the most common statements include:

- **Assignment statements**

Assignment statements are used to assign values to variables. For example, the following code assigns the value "Hello, world!" to the variable myVariable:

```
var myVariable = "Hello, world!";
```

- **Expression statements**

Expression statements are used to evaluate expressions. For example, the following code evaluates the expression 2 + 2 and prints the result to the console:

```
console.log(2 + 2);
```

- **Control flow statements**

Control flow statements are used to control the flow of execution of a JavaScript program. For example, the following code will only execute the statement inside the if statement if the value of the variable myVariable is equal to 10:

```
if (myVariable == 10) {
  console.log("The value of myVariable is 10.");
}
```

- **Function statements**

Function statements are used to create functions. For example, the following code creates a function called myFunction that takes one argument and returns the sum of the argument and 10:

```
function myFunction(x) {
  return x + 10;
}
```

- **Block statements**

Block statements are used to group statements together. For example, the following code groups the statements inside the { } curly braces together:

```
{
  console.log("This is a block statement.");
  console.log("This is another statement in the block.");
}
```

- **Comment statements**

Comment statements are used to add comments to JavaScript code. Comments are ignored by the JavaScript interpreter. For example, the following code is a comment:

```
// This is a comment.
```

Statements can be executed in any order, but it is usually best to execute them in a logical order. For example, you would typically want to assign values to variables before you use them in expressions.

Literals & Variables

In JavaScript, a literal is a value that is represented directly in the code. For example, the following are literals:

- The string "Hello, world!"
- The number 10
- The boolean value true
- The object {name: "John Doe", age: 30}
- The array [1, 2, 3]

Variables are names that are used to refer to values. For example, the following code creates a variable called myVar and assigns it the value "Hello, world!":

```
var myVar = "Hello, world!";
```

Once a variable is created, you can use its name to refer to its value. For example, the following code prints the value of the variable myVar to the console:

```
console.log(myVar);
```

Literals and variables are two of the most basic concepts in JavaScript. By understanding these concepts, you will be well on your way to learning JavaScript.

Here are some additional details about literals and variables:

- Literals can be of any type, including string, number, boolean, object, or array.
- Variables can be of any type, but they must be declared before they can be used.
- The value of a variable can be changed at any time.
- Variables can be used to store data, pass data to functions, and return data from functions.

Declaring Variable

JavaScript has three keywords for declaring variables: var, let, and const.

- var is the oldest and most basic keyword for declaring variables. It can be used to declare variables at any scope, including the global scope.
- let is a newer keyword that was introduced in ES6. It can only be used to declare variables in block scopes, such as the scope of a function or an if statement.
- const is the newest keyword for declaring variables. It can only be used to declare variables that cannot be changed.

Here is a table that summarizes the differences between var, let, and const:

Keyword	**Scope**	**Can be reassigned?**	**Can be deleted?**
var	Global, function, block	Yes	Yes
let	Block	No	Yes
const	Block	No	No

In general, it is best to use let whenever possible. It is more restrictive than var, but it also helps to prevent errors. For example, if you accidentally try to reassign a variable that was declared with let, you will get an error.

You should only use var if you need to be able to reassign a variable. For example, if you are using a variable to keep track of the current index of a loop, you will need to be able to reassign the variable.

You should only use const if you know that the variable will never need to be changed. For example, if you are declaring a constant, such as the pi constant, you should use const.

Operator in JS

Arithmetic operators

Arithmetic operators are used to perform arithmetic operations on numbers. The following are the arithmetic operators in JavaScript:

1. `+` - Addition
2. `-` - Subtraction
3. `*` - Multiplication
4. `/` - Division
5. `%` - Modulo

Assignment operators

Assignment operators are used to assign values to variables. The following are the assignment operators in JavaScript:

1. `=` - Simple assignment
2. `+=` - Addition assignment
3. `-=` - Subtraction assignment
4. `*=` - Multiplication assignment
5. `/=` - Division assignment
6. `%=` - Modulo assignment

Comparison operators

Comparison operators are used to compare values. The following are the comparison operators in JavaScript:

1. `==` - Equality
2. `!=` - Inequality
3. `<` - Less than
4. `>` - Greater than
5. `<=` - Less than or equal to
6. `>=` - Greater than or equal to

Logical operators

Logical operators are used to combine Boolean expressions. The following are the logical operators in JavaScript:

1. `&&` - Logical AND
2. `||` - Logical OR
3. `!` - Logical NOT

Bitwise operators

Bitwise operators are used to perform bitwise operations on numbers. The following are the bitwise operators in JavaScript:

1. `&` - Bitwise AND
2. `|` - Bitwise OR
3. `^` - Bitwise XOR
4. `~` - Bitwise NOT
5. `<<` - Bitwise left shift
6. `>>` - Bitwise right shift

Ternary operator

The ternary operator is a conditional operator that is used to evaluate a Boolean expression and return one of two values based on the result of the expression. The syntax of the ternary operator is as follows:

```
condition ? value_if_true : value_if_false
```

For example, the following code will assign the value "Hello, world!" to the variable myVar if the value of the variable is_morning is true, and it will assign the value "Good evening!" to the variable myVar if the value of the variable is_morning is false:

```
const is_morning = true;

const myVar = is_morning ? "Hello, world!" : "Good evening!";
```

Special operators

There are a few special operators in JavaScript that do not fit into any of the other categories. The following are the special operators in JavaScript:

1. `typeof` - Returns the type of a value
2. `new` - Creates a new object
3. `delete` - Deletes a property from an object
4. `in` - Checks if a property exists on an object
5. `instanceof` - Checks if an object is an instance of a particular class

Datatypes in JS

Primitive data types

Primitive data types are the basic data types in JavaScript. They are:

1. **Number -** A number can be any positive or negative integer, or any floating-point number.
2. **String -** A string is a sequence of characters.
3. **Boolean -** A boolean can be either true or false.
4. **Null -** The null value represents the absence of a value.
5. **Undefined -** The undefined value represents a value that has not yet been defined.

Object data types

Object data types are more complex than primitive data types. They are used to store collections of data.

1. **Object -** An object is a collection of properties. Each property has a name and a value.
2. **Array -** An array is a collection of elements. Each element can be of any type.
3. **Date -** A date represents a point in time.
4. **Math -** The Math object provides methods for performing mathematical operations.
5. **JSON -** JSON is a lightweight data-interchange format.

String

A string in JavaScript is a sequence of characters. Strings can be enclosed in double quotes or single quotes. For example, the following are all strings:

"Hello, world!"

'Hello, world!'

Strings can be used in many ways, such as:

- To store text data
- To display text on a web page
- To compare strings
- To search for strings
- To manipulate strings

JavaScript provides a few methods for working with strings. Some of the most common string methods include:

- **charAt() -** Returns the character at a specified index
- **charCodeAt() -** Returns the Unicode code point of the character at a specified index
- **concat() -** Concatenates two strings
- **indexOf()** - Returns the index of the first occurrence of a specified substring
- **lastIndexOf()** - Returns the index of the last occurrence of a specified substring
- **length -** Returns the length of a string
- **replace() -** Replaces all occurrences of a specified substring with another substring
- **slice() -** Extracts a substring from a string
- **split() -** Splits a string into an array of substrings
- **substring() -** Extracts a substring from a string
- **toLowerCase() -** Converts a string to lowercase
- **toUpperCase()** - Converts a string to uppercase
- **trim() -** Removes whitespace from the beginning and end of a string

JS Date object

The Date object in JavaScript represents a point in time. A Date object has properties that represent the year, month, day, hour, minute, second, and millisecond of the date.

The Date object can be used to:

- Get the current date and time
- Set a date and time
- Add or subtract time from a date and time
- Compare two dates and times
- Format a date and time for display

To create a Date object, you can use the new Date() constructor. The new Date() constructor takes no arguments, and it returns a new Date object that represents the current date and time.

For example, the following code creates a new Date object and stores it in the variable myDate:

```
const myDate = new Date();
```

You can use the properties of the Date object to get and set the date and time. For example, the following code gets the year from the Date object myDate:

```
const year = myDate.getFullYear();
```

The following code sets the month of the Date object myDate to January:

```
myDate.setMonth(0);
```

You can use the methods of the Date object to add or subtract time from a date and time, compare two dates and times, and format a date and time for display. For example, the following code adds one day to the Date object myDate:

```
myDate.setDate(myDate.getDate() + 1);
```

The following code compares the Date object myDate to the Date object otherDate:

```
const isEqual = myDate.getTime() === otherDate.getTime();
```

The following code formats the Date object myDate for display as a string:

```
const formattedDate = myDate.toLocaleString();
```

JS Math

The JavaScript **Math** object provides a collection of mathematical operations and constants. It is not a constructor and does not need to be instantiated. Here are some important methods associated with the **Math** object:

1. **Math.abs(x)**: Returns the absolute value of a number **x**.
2. **Math.ceil(x)**: Rounds a number **x** upward to the nearest integer.
3. **Math.floor(x)**: Rounds a number **x** downward to the nearest integer.
4. **Math.round(x)**: Rounds a number **x** to the nearest integer.
5. **Math.max(x1, x2, ..., xn)**: Returns the largest value among the given arguments.
6. **Math.min(x1, x2, ..., xn)**: Returns the smallest value among the given arguments.
7. **Math.random()**: Returns a random floating-point number between 0 and 1 (exclusive of 1).
8. **Math.pow(x, y)**: Returns **x** raised to the power of **y**.
9. **Math.sqrt(x)**: Returns the square root of **x**.
10. **Math.exp(x)**: Returns the exponential value of **x**.
11. **Math.log(x)**: Returns the natural logarithm (base e) of **x**.
12. **Math.sin(x)**, **Math.cos(x)**, **Math.tan(x)**: Returns the sine, cosine, and tangent of an angle **x**, respectively.
13. **Math.asin(x)**, **Math.acos(x)**, **Math.atan(x)**: Returns the arcsine, arccosine, and arctangent of a number **x**, respectively.
14. **Math.PI**: Represents the mathematical constant π (pi), the ratio of the circumference of a circle to its diameter.

These are just a few examples of the methods available in the **Math** object. The **Math** object provides a wide range of mathematical functions and constants for various calculations and computations in JavaScript.

Here's an example that demonstrates the usage of some methods from the **Math** object:

```
// Random number between 1 and 10

const randomNum = Math.floor(Math.random() * 10) + 1;

console.log(randomNum);

// Finding the maximum and minimum values

const numbers = [4, 9, 2, 7, 5];

const maxValue = Math.max(...numbers);

const minValue = Math.min(...numbers);

console.log(maxValue, minValue);
```

```
// Calculating the square root
const num = 16;
const squareRoot = Math.sqrt(num);
console.log(squareRoot);
// Calculating the sine and cosine
const angle = Math.PI / 4; // 45 degrees in radians
const sineValue = Math.sin(angle);
const cosineValue = Math.cos(angle);
console.log(sineValue, cosineValue);
// Rounding a number
const value = 3.7;
const roundedValue = Math.round(value);
console.log(roundedValue);
```

JS Conditional statements

Conditional statements in JavaScript are used to execute code based on certain conditions. There are two main types of conditional statements in JavaScript: **if** statements and **switch** statements.

If Statements An **if** statement is used to execute a block of code if a condition is true. The basic syntax of an **if** statement is as follows:

```
if (condition) {
  // code to execute if condition is true
}
```

Here's an example:

```
var x = 10;
if (x > 5) {
  console.log("x is greater than 5");
}
```

In this example, the condition **x > 5** is true, so the code inside the **if** statement is executed and "x is greater than 5" is logged to the console.

You can also use an **else** statement to execute a block of code if the condition is false:

```
var x = 2;
if (x > 5) {
  console.log("x is greater than 5");
} else {
  console.log("x is less than or equal to 5");
}
```

In this example, the condition **x > 5** is false, so the code inside the **else** statement is executed and "x is less than or equal to 5" is logged to the console.

Switch Statements A **switch** statement is used to execute different blocks of code depending on different cases. The basic syntax of a **switch** statement is as follows:

```
switch (expression) {
  case value1:
    // code to execute if expression equals value1
    break;
  case value2:
    // code to execute if expression equals value2
    break;
```

```
  ...
  default:
    // code to execute if none of the cases match
    break;
}
```

Here's an example:

```
var fruit = "banana";
switch (fruit) {
  case "apple":
    console.log("You like apples");
    break;
  case "banana":
    console.log("You like bananas");
    break;
  case "orange":
    console.log("You like oranges");
    break;
  default:
    console.log("You don't like any of the fruits");
    break;
}
```

In this example, the value of **fruit** is "banana", so the code inside the second case statement is executed and "You like bananas" is logged to the console.

It's important to note that the **break** statement is used to exit the switch statement once a matching case has been found. Without the **break** statement, all the following cases will also be executed.

Here are five questions on JavaScript conditional statements:

1. Write a JavaScript program that checks if a number is positive, negative, or zero. If the number is positive, display "Positive." If the number is negative, display "Negative." If the number is zero, display "Zero."
2. Write a JavaScript function that takes a string as input and determines whether the string is a palindrome (reads the same forwards and backwards). Return true if it is a palindrome, and false otherwise.
3. Write a JavaScript program that determines the grade of a student based on their score. The program should take a score as input and display the corresponding grade: "A" for scores 90 and above, "B" for scores 80-89, "C" for scores 70-79, "D" for scores 60-69, and "F" for scores below 60.
4. Write a JavaScript function that takes an array of numbers as input and returns the sum of all the positive numbers in the array.
5. Write a JavaScript program that determines if a year is a leap year. A leap year is divisible by 4, but not divisible by 100 unless it is also divisible by 400. The program should display "Leap year" if the year is a leap year, and "Not a leap year" otherwise.

Loops

Loops in JavaScript are used to execute a block of code repeatedly until a certain condition is met. There are three main types of loops in JavaScript: **for** loops, **while** loops, and **do-while** loops.

For Loops A **for** loop is used to execute a block of code a specified number of times. The basic syntax of a **for** loop is as follows:

```
for (initialization; condition; increment/decrement) {
 // code to execute
}
```

Here's an example:

```
for (var i = 0; i < 5; i++) {
 console.log("The value of i is: " + i);
}
```

In this example, the loop will run 5 times, starting with **i** equal to 0 and incrementing by 1 each time until **i** reaches 4. The output of the above code will be:

The value of i is: 0

The value of i is: 1

The value of i is: 2

The value of i is: 3

The value of i is: 4

While Loops

A **while** loop is used to execute a block of code while a certain condition is true. The basic syntax of a **while** loop is as follows:

```
while (condition) {
 // code to execute
}
```

Here's an example:

```
var i = 0;
while (i < 5) {
 console.log("The value of i is: " + i);
 i++;
}
```

In this example, the loop will run 5 times, starting with **i** equal to 0 and incrementing by 1 each time until **i** reaches 4. The output of the above code will be the same as the **for** loop example:

The value of i is: 0

The value of i is: 1

The value of i is: 2

The value of i is: 3

The value of i is: 4

Do-While Loops

A **do-while** loop is similar to a **while** loop, except that it will always execute the block of code at least once, even if the condition is initially false. The basic syntax of a **do-while** loop is as follows:

```
do {
  // code to execute
} while (condition);
```

Example:

```
var i = 0;
do {
  console.log("The value of i is: " + i);
  i++;
} while (i < 5);
```

In this example, the loop will run 5 times, starting with **i** equal to 0 and incrementing by 1 each time until **i** reaches 4. The output of the above code will be the same as the **for** loop and **while** loop example.

Loops are powerful tools for iterating over arrays, executing complex logic, and handling asynchronous operations.

Here are five questions on JavaScript loop statements:

1. Write a loop that prints numbers from 1 to 10 in the console.
2. Given an array of names, use a loop to iterate through the array and print each name on a new line.
3. Write a loop that calculates the sum of all even numbers from 1 to 50.
4. Create a loop that iterates from 10 to 1 and prints the current number in reverse order.
5. Given an array of numbers, write a loop that finds the largest number in the array and stores it in a variable.