JavaScript Concepts

1. An Introduction to Functions in JavaScript

Functions are reusable blocks of code that perform a specific task. They allow you to organize your code, making it more modular and easier to maintain.

Defining a Function: A function is defined using the function keyword, followed by the function name, a list of parameters (enclosed in parentheses), and the function body (enclosed in curly braces).

Example:

```
function greet(name) {
  return `Hello, ${name}!`;
}
```

Calling a Function: You call a function by its name followed by parentheses, optionally passing arguments.

Example:

```
console.log(greet('Alice')); // Output: Hello, Alice!
```

2. Global and Local Variables

Global Variables: Variables declared outside any function have global scope, meaning they can be accessed from anywhere in the code.

Example:

```
let globalVar = 'I am a global variable';

function showGlobalVar() {
  console.log(globalVar);
}

showGlobalVar(); // Output: I am a global variable
```

Local Variables: Variables declared within a function are local to that function, meaning they can only be accessed within that function.

Example:

```
function showLocalVar() {
  let localVar = 'I am a local variable';
  console.log(localVar);
}

showLocalVar(); // Output: I am a local variable
console.log(localVar); // Error: localVar is not defined
```

3. Working with Functions

Parameters and Arguments: Functions can take parameters, which are variables listed as part of the function definition. Arguments are the values passed to the function when it is called.

Example:

```
function add(a, b) {
  return a + b;
}

console.log(add(2, 3)); // Output: 5
```

Default Parameters: You can set default values for parameters in case no argument is provided.

Example:

```
function greet(name = 'Guest') {
  return `Hello, ${name}!`;
}

console.log(greet()); // Output: Hello, Guest!
```

4. The Fundamentals of Error Handling

Error handling is a critical part of programming to manage and respond to runtime errors.

Try...Catch Statement: Allows you to catch and handle errors gracefully.

Example:

```
try {
  // Code that may throw an error
  let result = riskyOperation();
} catch (error) {
  console.log('An error occurred:', error.message);
}
```

Throw Statement: Allows you to create custom errors.

Example:

```
function checkAge(age) {
  if (age < 18) {
    throw new Error('You must be at least 18 years old.');
  }
  return 'Access granted';
}

try {
  console.log(checkAge(16));
} catch (error) {
  console.log(error.message); // Output: You must be at least 18 years old.
}
```

5. Creating Arrays

Arrays are used to store multiple values in a single variable.

Array Declaration: Arrays can be declared using square brackets.

Example:

```
let fruits = ['Apple', 'Banana', 'Cherry'];
```

6. Rest Parameters in JavaScript

The rest parameter syntax allows a function to accept an indefinite number of arguments as an array.

Syntax:

```
function sum(...numbers) {
  return numbers.reduce((total, num) => total + num, 0);
}

console.log(sum(1, 2, 3, 4)); // Output: 10
```

7. The Spread Syntax for Arrays

The spread syntax allows an iterable (like an array) to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected.

Example:

```
let arr1 = [1, 2, 3];
let arr2 = [...arr1, 4, 5, 6];

console.log(arr2); // Output: [1, 2, 3, 4, 5, 6]
```

8. Destructuring Arrays

Destructuring assignment allows you to unpack values from arrays or properties from objects into distinct variables.

Example:

```
let [first, second] = [1, 2, 3];
console.log(first); // Output: 1
console.log(second); // Output: 2
```

9. Copying Arrays

You can create a shallow copy of an array using the spread operator or Array.from.

Using Spread Operator:

Example:

```
let original = [1, 2, 3];
let copy = [...original];
```

Using Array.from:

Example:

```
let copy = Array.from(original);
```

10. Splicing and Slicing Arrays

Splice: Used to add or remove elements from an array.

Example:

```
let fruits = ['Apple', 'Banana', 'Cherry'];
fruits.splice(1, 1, 'Blueberry');
console.log(fruits); // Output: ['Apple', 'Blueberry', 'Cherry']
```

Slice: Returns a new array containing a portion of the original array.

Example:

```
let fruits = ['Apple', 'Banana', 'Cherry'];
let slicedFruits = fruits.slice(1, 3);
console.log(slicedFruits); // Output: ['Banana', 'Cherry']
```

11. Concatenating and Sorting Arrays

Concatenating: You can concatenate two or more arrays using the concat method or the spread operator.

Example:

```
let arr1 = [1, 2];
let arr2 = [3, 4];
let concatenated = arr1.concat(arr2);
console.log(concatenated); // Output: [1, 2, 3, 4]
```

Sorting: The sort method sorts the elements of an array.

Example:

```
let numbers = [4, 2, 5, 1, 3];
numbers.sort();
console.log(numbers); // Output: [1, 2, 3, 4, 5]
```