Basics of JavaScript Code

JavaScript is a high-level, dynamic, and interpreted programming language. It is used to add interactivity and other dynamic elements to websites.

The console in JavaScript

The console in JavaScript is a tool used for debugging and testing purposes. It allows developers to log information to the browser console for viewing. This can be useful for checking the values of variables, examining the output of functions, and tracking down errors in code.

There are several methods to log information to the console in JavaScript:

console.log(): This method logs the specified data to the console.
For example:

```
console.log("Hello World"); // Output: Hello World
```

console.error(): This method logs an error message to the console.

For example:

```
console.error("This is an error"); // Output: Error: This is an error
```

console.warn(): This method logs a warning message to the console.
For example:

```
console.warn("This is a warning"); // Output: Warning: This is a warning
```

These methods can be used directly in the JavaScript code and the output can be viewed in the browser console by opening the Developer Tools in most modern browsers.

Here are some basics of JavaScript with code examples:

Variables:

variables are used to store values in JavaScript. You can declare a variable with the "let" keyword, like this:

```
let x = 10;
let name = "John";
let age = 30;
let isStudent = false;
let weight = 75.5;
let address = "123 Main St.";
```

JavaScript variables are containers that store data values. There are three main types of variables in JavaScript:

var: This is the original way to declare a variable in JavaScript. It is function scoped, which means it is accessible within the function it was declared in.

For example:

```
var name = "John";
function printName() {
  console.log(name);
}
printName(); // Output: John
```

let: This type of variable was introduced in ECMAScript 6 and is block scoped. It means that the variable is only accessible within the block of code it was declared in.

For example:

```
let age = 30;
if (true) {
  let age = 40;
  console.log(age); // Output: 40
}
console.log(age); // Output: 30
```

const: This type of variable is also block scoped and was introduced in ECMAScript 6. The difference between const and let is that const cannot be reassigned after it has been declared.

For example:

```
const country = "USA";
country = "Canada"; // Throws an error
```

It is important to note that the value stored inside a variable declared with const can still be mutable, meaning its properties can be changed but it cannot be reassigned to a completely new value.

JavaScript Comments

In JavaScript, you can add comments in your code to describe what it does or to temporarily ignore parts of the code. There are two types of comments: single-line comments and multi-line comments.

Single-line comment:

```
// This is a single-line comment
```

Multi-line comment:

```
/*
  This is a multi-line
  comment
*/
```

Here's an example that shows how comments can be used in a code:

```
// This is a single-line comment
/*
  This is a multi-line
  comment
*/

const name = "John"; // This is also a single-line comment
// The code below will print the value of the variable "name"
console.log(name);
```

Data Types:

JavaScript has several data types, including numbers, strings, and booleans. Here's an example of each:

```
let num = 10;
let str = "Hello World";
let bool = true;
let num1 = 20;
let num2 = -10;
let str1 = "Hello";
let str2 = 'JavaScript';
let bool1 = true;
let bool2 = false;
```

JavaScript Data Types

In JavaScript, there are seven basic data types:

Number: Used to represent numbers. Example: const num = 42;

String: Used to represent a sequence of characters. Example: const name = "John";

Boolean: Used to represent a logical value of either true or false. Example: const isMarried = true;

Undefined: Represents a value that has not been assigned. Example: const age; console.log(age); // Output: undefined

Null: Represents a value that is explicitly null. Example: const address = null;

Symbol: Used to create unique identifiers for objects. Example: const symbol = Symbol("description");

Object: Used to represent a collection of key-value pairs. Example: const person = { name: "John", age: 30 };

In JavaScript, variables have a dynamic type and can change their type based on the value assigned to them. The typeof operator can be used to check the type of a variable.

Example:

```
const num = 42;
console.log(typeof num); // Output: "number"

const name = "John";
console.log(typeof name); // Output: "string"

const isMarried = true;
console.log(typeof isMarried); // Output: "boolean"

const age;
console.log(typeof age); // Output: "undefined"

const address = null;
console.log(typeof address); // Output: "object"

const symbol = Symbol("description");
console.log(typeof symbol); // Output: "symbol"
```

```
const person = { name: "John", age: 30 };
console.log(typeof person); // Output: "object"
```

Arithmetic Operations:

JavaScript supports basic arithmetic operations like addition, subtraction, multiplication, and division. Here's an example:

```
let x = 10;
let y = 5;
let sum = x + y;
let difference = x - y;
let product = x * y;
let quotient = x / y;
let x = 10;
let y = 20;
let sum = x + y;
let difference = x - y;
let product = x * y;
let quotient = x / y;
let modulo = x % y;
let increment = x++;
let decrement = y--;
```

Conditional Statements:

Conditional statements are used to execute different blocks of code based on conditions. The if-else statement is the most common type of conditional statement in JavaScript. Here's an example:

```
let x = 10;
```

```
if (x > 5) {
  console.log("x is greater than 5");
} else {
  console.log("x is not greater than 5");
}
let grade = 85;

if (grade >= 90) {
  console.log("A");
} else if (grade >= 80) {
  console.log("B");
} else if (grade >= 70) {
  console.log("C");
} else {
  console.log("F");
}

let day = "Sunday";

switch (day) {
  case "Monday":
    console.log("Today is Monday");
    break;
  case "Tuesday":
    console.log("Today is Tuesday");
    break;
  case "Wednesday":
    console.log("Today is Wednesday");
    break;
  case "Thursday":
    console.log("Today is Thursday");
    break;
```

```
  case "Friday":
    console.log("Today is Friday");
    break;
  case "Saturday":
    console.log("Today is Saturday");
    break;
  default:
    console.log("Today is Sunday");
}
```

Functions:

Functions are blocks of code that can be executed when they are called. Here's an example:

```
function greeting() {
  console.log("Hello World");
}

greeting();
function add(x, y) {
  return x + y;
}

let result = add(10, 20);
console.log(result);

function greet(name) {
  console.log("Hello " + name);
}
```

```
greet("John");

function calculateArea(width, height) {
  return width * height;
}

let area = calculateArea(10, 20);
console.log(area);

function checkOddEven(num) {
  if (num % 2 === 0) {
    return "Even";
  } else {
    return "Odd";
  }
}

let result = checkOddEven(10);
console.log(result);

function generateRandomNumber() {
  return Math.floor(Math.random() * 100);
}

let randomNumber = generateRandomNumber();
console.log(randomNumber);
```

These are just some of the basics of JavaScript. There is much more to learn, but these basics will give you a good foundation to start building your skills.

JavaScript Loops:

JavaScript has two types of loops: for loops and while loops. Here are examples of each:

For Loop:

```
for (const i = 0; i < 5; i++) {
  console.log("Iteration " + (i + 1));
}
```

This code will output:

Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5

While Loop:

```
const i = 0;
while (i < 5) {
  console.log("Iteration " + (i + 1));
  i++;
}
```

This code will output:

Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5

In addition to these two loops, there is also the do-while loop, which is similar to the while loop, but with a slight difference in the way the condition is checked. Here's an example:

```
const i = 0;
do {
  console.log("Iteration " + (i + 1));
  i++;
} while (i < 5);
```

This code will output:

Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5

These are the basics of JavaScript loops. You can use them to repeat a block of code multiple times based on conditions.

JavaScript Arrays

An array in JavaScript is a data structure that stores a collection of values. You can access individual values of an array by referring to their index number. Arrays are declared using square brackets [] and items are separated by commas.

Here's an example of an array in JavaScript:

```
const fruits = ["apple", "banana", "cherry"];
```

JavaScript provides several built-in methods for working with arrays. Here are some commonly used methods with examples:

length property: returns the number of elements in an array.

```
const fruits = ["apple", "banana", "cherry"];
console.log(fruits.length); // Output: 3
```

push() method: adds an element to the end of an array.

```
const fruits = ["apple", "banana", "cherry"];
fruits.push("orange");
console.log(fruits); // Output: ["apple", "banana",
"cherry", "orange"]
```

pop() method: removes the last element from an array and returns it.

```
const fruits = ["apple", "banana", "cherry"];
const lastFruit = fruits.pop();
console.log(fruits); // Output: ["apple", "banana"]
console.log(lastFruit); // Output: "cherry"
```

unshift() method: adds an element to the beginning of an array.

```
const fruits = ["apple", "banana", "cherry"];
fruits.unshift("peach");
console.log(fruits); // Output: ["peach", "apple",
"banana", "cherry"]
```

shift() method: removes the first element from an array and returns it.

```
const fruits = ["apple", "banana", "cherry"];
const firstFruit = fruits.shift();
console.log(fruits); // Output: ["banana", "cherry"]
```

```
console.log(firstFruit); // Output: "apple"
```

splice() method: adds or removes elements from an array.

```
const fruits = ["apple", "banana", "cherry"];
fruits.splice(1, 0, "lemon", "lime");
console.log(fruits); // Output: ["apple", "lemon",
"lime", "banana", "cherry"]
```

These are some of the most commonly used array methods in JavaScript. You can use these methods to manipulate arrays and perform various operations on them.

JavaScript Objects

In JavaScript, an object is a collection of key-value pairs that store data. Objects are declared using curly braces {} and the keys and values are separated by colons.

Here's an example of an object in JavaScript:

```
const person = {
  name: "John",
  age: 30,
  location: "San Francisco"
};
```

You can access the values of an object using the dot notation or square bracket notation.

Here's an example of how to access the values of an object using the dot notation:

```
const person = {
  name: "John",
  age: 30,
```

```
  location: "San Francisco"
};
console.log(person.name); // Output: "John"
console.log(person.age); // Output: 30
console.log(person.location); // Output: "San
Francisco"
```

Here's an example of how to access the values of an object using the square bracket notation:

```
const person = {
  name: "John",
  age: 30,
  location: "San Francisco"
};

const nameKey = "name";
const ageKey = "age";
const locationKey = "location";
console.log(person[nameKey]); // Output: "John"
console.log(person[ageKey]); // Output: 30
console.log(person[locationKey]); // Output: "San
Francisco"
```

You can also add new properties or change the values of existing properties in an object.
Here's an example of how to add a new property to an object:

```
const person = {
  name: "John",
  age: 30,
  location: "San Francisco"
};
```

```
person.email = "john@example.com";
console.log(person); // Output: { name: "John", age:
30, location: "San Francisco", email:
"john@example.com" }
```

Here's an example of how to change the value of an existing property in an object:

```
const person = {
  name: "John",
  age: 30,
  location: "San Francisco"
};
person.age = 35;
console.log(person); // Output: { name: "John", age:
35, location: "San Francisco" }
```

Objects are widely used in JavaScript to store data and represent real-world objects. You can use objects to create more complex data structures and manage your data more effectively.

How to output a table into the console

console.table(): This method logs the data in a table format. For example:

```
console.table([{a:1, b:2}, {a:3, b:4}]);
// Output:
// ┌─────────┬───┬───┐
// │ (index) │ a │ b │
// ├─────────┼───┼───┤
// │    0    │ 1 │ 2 │
// │    1    │ 3 │ 4 │
```

// └──────────────┴────┴────┘

JavaScript String Methods

JavaScript provides several built-in methods for manipulating strings, some of the commonly used ones are:

length: Returns the length of the string.
Example: var name = "John"; console.log(name.length); // Output: 4

concat: Joins two or more strings together.
Example: var firstName = "John"; var lastName = "Doe"; console.log(firstName.concat(" ", lastName)); // Output: "John Doe"

toUpperCase: Converts the string to uppercase.
Example: var name = "John"; console.log(name.toUpperCase()); // Output: "JOHN"

toLowerCase: Converts the string to lowercase.
Example: var name = "John"; console.log(name.toLowerCase()); // Output: "john"

charAt: Returns the character at the specified index.
Example: var name = "John"; console.log(name.charAt(0)); // Output: "J"

indexOf: Returns the index of the first occurrence of the specified value, or -1 if it is not found.
Example: var name = "John"; console.log(name.indexOf("o")); // Output: 1

slice: Extracts a part of the string and returns it as a new string.
Example: var name = "John"; console.log(name.slice(0, 2)); // Output: "Jo"

replace: Replaces the first occurrence of the specified value with a new value.
Example: var name = "John"; console.log(name.replace("J", "j")); // Output: "john"

trim: Removes whitespaces from both ends of the string.
Example: var name = " John "; console.log(name.trim()); // Output: "John"

These are just a few of the many string methods available in JavaScript, each with its own specific use case. Understanding and utilizing these methods can greatly simplify string manipulation tasks in your code.

JavaScript Number Methods

JavaScript provides several built-in methods for working with numbers. Here are some common ones:

Number.isInteger(): This method returns true if the argument is an integer and false otherwise.

For example:

```
console.log(Number.isInteger(3)); // Output: true
console.log(Number.isInteger(3.14)); // Output: false
```

Number.parseFloat(): This method parses a string argument and returns a floating-point number.

For example:
```
console.log(Number.parseFloat("3.14")); // Output: 3.14
```

Number.parseInt(): This method parses a string argument and returns an integer.

For example:
```
console.log(Number.parseInt("3.14")); // Output: 3
```

Number.toFixed(): This method returns a string representation of a number with a specified number of decimal places.

For example:
```
console.log((3.14).toFixed(2)); // Output: 3.14
```

Number.toPrecision(): This method returns a string representation of a number with a specified number of significant digits.

For example:
```
console.log((3.14).toPrecision(2)); // Output: 3.1
```

Math.abs(): This method returns the absolute value of a number.

For example:
```
console.log(Math.abs(-3.14)); // Output: 3.14
```

Math.ceil(): This method returns the smallest integer greater than or equal to a number.

For example:
```
console.log(Math.ceil(3.14)); // Output: 4
```

Math.floor(): This method returns the largest integer less than or equal to a number.

For example:
```
console.log(Math.floor(3.14)); // Output: 3
```

These methods can be used to perform various operations on numbers in JavaScript.

JavaScript Math

JavaScript provides several built-in methods for working with mathematical operations. Here are some common ones:

Math.abs(): This method returns the absolute value of a number.

For example:
console.log(Math.abs(-3.14)); // Output: 3.14

Math.ceil(): This method returns the smallest integer greater than or equal to a number.

For example:
console.log(Math.ceil(3.14)); // Output: 4

Math.floor(): This method returns the largest integer less than or equal to a number.

For example:
console.log(Math.floor(3.14)); // Output: 3

Math.max(): This method returns the largest of zero or more numbers.

For example:
```
console.log(Math.max(3, 7, 4)); // Output: 7
```

Math.min(): This method returns the smallest of zero or more numbers.

For example:
```
console.log(Math.min(3, 7, 4)); // Output: 3
```

Math.pow(): This method returns the value of a number raised to the specified power.

For example:
```
console.log(Math.pow(3, 2)); // Output: 9
```

Math.random(): This method returns a random number between 0 (inclusive) and 1 (exclusive).

For example:
```
console.log(Math.random()); // Output: a random number between 0 and 1
```

Math.round(): This method returns the value of a number rounded to the nearest integer.

For example:

```
console.log(Math.round(3.14)); // Output: 3
```

These methods can be used to perform various mathematical operations in JavaScript.

JavaScript Classes

JavaScript added class syntax with ECMAScript 6 (ES6) as a way to write reusable code and create objects. A class is a blueprint for creating objects that have similar properties and methods.

Here's an example of a class in JavaScript:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log(`Hello, my name is ${this.name} and I
am ${this.age} years old.`);
  }
}
```

In the above example, the Person class has two properties: name and age, and a method greet that prints a greeting message.

To create an object from this class, you use the new operator and call the constructor function:

```
const person = new Person("John", 30);
person.greet();
```

```
// Output: Hello, my name is John and I am 30 years
old.
```
You can also extend a class to create a subclass and inherit properties and methods from the parent class:

```
class Student extends Person {
  constructor(name, age, major) {
    super(name, age);
    this.major = major;
  }

  study() {
    console.log(`I am studying ${this.major}.`);
  }
}

const student = new Student("Jane", 20, "Computer
Science");
student.greet();
// Output: Hello, my name is Jane and I am 20 years
old.
student.study();
// Output: I am studying Computer Science.
```

In the above example, the Student class extends the Person class and adds a new property major and a method study. The super keyword is used to call the constructor of the parent class and pass along the required properties.

Classes in JavaScript provide a way to write organized and reusable code, making it easier to maintain and extend your codebase.

Regular Expression (RegExp)

A Regular Expression (RegExp) is a pattern that specifies a set of strings. JavaScript provides built-in support for regular expressions with the RegExp object.

Here's an example of how to use regular expressions in JavaScript:

```
let string = "Hello, World!";
let pattern = /Hello/;
let result = pattern.test(string);
console.log(result); // Output: true
```

In this example, we define a string "Hello, World!" and a regular expression pattern /Hello/. We use the test() method of the RegExp object to test if the string matches the pattern. The result is a boolean value indicating whether the string matches the pattern.

You can also use the match() method of the String object to find all matches of a regular expression pattern in a string:

```
let string = "Hello, World! Hello, JavaScript!";
let pattern = /Hello/g;
let result = string.match(pattern);
console.log(result); // Output: [ "Hello", "Hello" ]
```

In this example, we add the g (global) flag to the pattern to find all matches in the string, instead of just the first match. The result is an array of strings containing all matches of the pattern in the string.

Regular expressions can be a powerful tool for matching and manipulating strings in JavaScript.