# 100 Fundamental JavaScript Concepts

**Introduction:** JavaScript is a widely-used and versatile programming language that powers the interactive aspects of web development. As a developer, mastering these 100 essential concepts will empower you to write efficient, scalable, and maintainable JavaScript code. Let's dive into each concept with detailed explanations and practical examples!

1. **Variables and Data Types:** Variables store data values, and JavaScript has various data types like strings, numbers, Booleans, etc.
   **Example:**
   ```
   let name = 'John';
   const age = 30;
   ```

2. **Functions:** Functions are blocks of reusable code that perform a specific task and can take parameters and return values.
   **Example:**
   ```
   function addNumbers(a, b) {
       return a + b;
   }
   ```

3. **Arrays:** Arrays are ordered collections of elements that can hold different data types.
   **Example:**
   ```
   const numbers = [1, 2, 3, 4, 5];
   ```

4. **Objects:** Objects are unordered collections of key-value pairs, where each key is a unique identifier.
   **Example:**
   ```
   const person = {
       name: 'Alice', age: 25
    };
   ```

5. **Conditional Statements:** Conditional statements execute different code blocks based on specified conditions.
   **Example:**
   ```
   if (age >= 18) {
     console.log('You are an adult.');
   } else {
     console.log('You are a minor.');
   }
   ```

6. **Loops:** Loops execute a block of code repeatedly until a specified condition is met.
   **Example:**
   ```
   for (let i = 0; i < 5; i++) {
     console.log(i);
    }
   ```

7. **DOM Manipulation:** The Document Object Model (DOM) allows interacting with HTML elements dynamically.

**Example:**

```
const element =
document.getElementById('myElement');
element.innerHTML = 'Hello, World!';
```

8. **Event Handling:** Event handling enables responding to user actions (e.g., clicks) on web pages.
   **Example:**

```
document.getElementById('myButton')
.addEventListener('click', function() {
 console.log('Button clicked!');
});
```

9. **Scope:** Scope determines the accessibility of variables within different parts of the code.
   **Example:**

```
function foo() {
  let x = 10; console.log(x);
}
```

10. **Closures:** Closures are functions that have access to variables from their outer (enclosing) function even after it has finished executing.
    **Example:**

```
function outer() {
  let count = 0;
  return function inner() {
    count++; console.log(count); };
 }
```

11. **Callbacks:** Callbacks are functions passed as arguments to another function to be executed later.
    **Example:**

```
function fetchData(callback) {
   // Fetch data from API
   callback(data);
 }
```

12. **Promises:** Promises are objects that represent the eventual completion or failure of an asynchronous operation.
    **Example:**

```
const fetchData = () => {
return new Promise((resolve, reject) => {
   // Fetch data from API and resolve or reject accordingly
 });
};
```

13. **Async/Await:** Async/await simplifies asynchronous code by making it look more like synchronous code.

**Example:**

```
async function getData() {
try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('Error fetching data:', error);
  }
 }
```

14. **Arrow Functions:** Arrow functions provide a concise syntax for writing function expressions.
    **Example:**

```
const addNumbers = (a, b) => a + b;
```

15. **ES6 Modules:** ES6 modules allow organizing code into separate files and sharing functionalities across scripts.
    **Example:**

```
// Exporting module
export const greeting = 'Hello';
// Importing module
import { greeting } from './greetings.js';
```

16. **Template Literals:** Template literals enable embedding expressions inside string literals, making string interpolation more readable.
    **Example:**

```
const name = 'Alice'; console.log(`Hello, ${name}!`);
```

17. **Destructuring:** Destructuring simplifies extracting values from arrays and objects.
    **Example:**

```
const { age } = person;
console.log(age);
```

18. **Spread Operator:** The spread operator allows expanding elements of an array or object into places where multiple elements are expected.
    **Example:**

```
const numbers = [1, 2, 3];
const newNumbers = [...numbers, 4, 5];
```

19. **Rest Parameters:** Rest parameters collect all remaining arguments into an array.
    **Example:**

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

20. **Object-oriented Programming (OOP):** OOP is a programming paradigm based on objects that interact with each other to solve complex problems.

**Example:**

```
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    console.log(`Hello, ${this.name}!`);
  }
}
const john = new Person('John');
john.greet();
```

21. **Prototypal Inheritance:** Prototypal inheritance enables objects to inherit properties and methods from other objects.

**Example:**

```
function Animal(name) {
  this.name = name;
}
Animal.prototype.sound = function() {
  console.log('Some generic sound.');
};
function Dog(name) {
  Animal.call(this, name);
}
Dog.prototype =
Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;
Dog.prototype.sound = function() {
  console.log('Woof! Woof!');
};
const dog = new Dog('Buddy');
dog.sound();
```

22. **Constructor Functions:** Constructor functions create and initialize objects using the "new" keyword.

**Example:**

```
function Car(make, model) {
  this.make = make;
  this.model = model;
}
const myCar = new Car('Toyota', 'Camry');
```

23. **Classes:** Classes are blueprints for creating objects, providing a more structured way of defining constructor functions.

**Example:**

```
class Rectangle {
 constructor(width, height) {
   this.width = width;
   this.height = height;
 }
 getArea() {
   return this.width * this.height;
 }
}
const rect = new Rectangle(5, 10);
console.log(rect.getArea());
```

24. **Prototype Chain:** The prototype chain allows objects to inherit properties from their prototype objects.
    **Example:**

```
const person = {
 name: 'Alice',
};
const programmer = {
 language: 'JavaScript',
};
programmer.__proto__ = person;
console.log(programmer.name);
```

25. **Hoisting:** Hoisting allows variables and function declarations to be used before they are declared.
    **Example:**

```
console.log(name); var name = 'Alice';
```

26. **Strict Mode:** Strict mode enforces stricter rules for writing JavaScript to avoid common mistakes and improve performance.
    **Example:**

```
'use strict';
```

27. **Type Coercion:** Type coercion is the automatic conversion of one data type to another.
    **Example:**

```
console.log(5 + '5'); // Output: "55"
```

28. **Truthy and Falsy Values:** JavaScript treats certain values as either truthy or falsy when evaluating conditions.
    **Example:**

```
if ('hello') {
 console.log('This will be executed.');
}
```

29. **Ternary Operator:** The ternary operator provides a shorthand way of writing if-else statements.

**Example:**

```
const age = 20;
const status = age >= 18 ? 'Adult' : 'Minor';
```

30. **Error Handling:** Error handling is essential to catch and handle exceptions gracefully.
    **Example:**

```
const data = '{"name":"Alice","age":30}';
const parsedData = JSON.parse(data);
console.log(parsedData.name);
```

31. **JSON (JavaScript Object Notation):** JSON is a lightweight data interchange format used to transmit and store data as text.
    **Example:**

```
const data = '{"name":"Alice","age":30}';
const parsedData = JSON.parse(data);
console.log(parsedData.name);
```

32. **Date and Time:**
    JavaScript provides Date objects for working with dates and times.
    **Example:**

```
const now = new Date();
console.log(now.toISOString());
```

33. **Regular Expressions (RegEx):** Regular expressions are patterns used to match and manipulate strings.
    **Example:**

```
const pattern = /[a-z]+/;
console.log(pattern.test('hello')); // Output: true
```

34. **Local Storage and Session Storage:** Local storage and session storage are mechanisms to store data on the client-side web browser.
    **Example:**

```
localStorage.setItem('username', 'Alice');
const username = localStorage.getItem('username');
```

35. **AJAX (Asynchronous JavaScript and XML):** AJAX enables updating parts of a web page without reloading the whole page.
    **Example:**

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.example.com/data', true);
xhr.onreadystatechange = function() {
  if (xhr.readyState === 4 && xhr.status === 200) {
    const data = JSON.parse(xhr.responseText);
    console.log(data);
  }
};
xhr.send();
```

36. **Fetch API:** The Fetch API simplifies making network requests and handling responses using promises.

**Example:**

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error fetching data:', error));
```

37. **Map:** The map method creates a new array with the results of calling a provided function on every element in the original array.

**Example:**

```
const numbers = [1, 2, 3];
const squaredNumbers = numbers.map(num => num * num);
```

38. **Filter:** The filter method creates a new array with all elements that pass the test implemented by the provided function.

**Example:**

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter(num => num % 2 === 0);
```

39. **Reduce:** The reduce method reduces the array to a single value by executing a reducer function on each element.

**Example:**

```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((acc, num) => acc + num, 0);
```

40. **ES6 Sets:** Sets are collections of unique values, which can be used to eliminate duplicate elements from an array.

**Example:**

```
const uniqueNumbers = new Set([1, 2, 2, 3, 3, 4, 5]);
```

41. **ES6 Maps:** Maps are collections of key-value pairs, similar to objects, but can have any data type as keys.

**Example:**

```
const person = new Map();
person.set('name', 'Alice');
person.set('age', 25);
```

42. **ES6 Symbols:** Symbols are unique and immutable data types often used as object property keys.

**Example:**

```
const mySymbol = Symbol('description');
const obj = {
  [mySymbol]: 'This is a symbol property',
};
```

43. **Object.assign():** Object.assign() copies the values of all enumerable properties from one or more source objects into a target object.

**Example:**

```
const source = { name: 'Alice' };
const target = {};
Object.assign(target, source);
```

44. **Array.prototype.forEach():** The forEach method executes a provided function once for each array element.
    **Example:**

```
const numbers = [1, 2, 3];
numbers.forEach(num => console.log(num));
```

45. **Array.prototype.some():** The some method tests whether at least one element in the array passes the test implemented by the provided function.
    **Example:**

```
const numbers = [1, 2, 3, 4, 5];
const hasEvenNumber = numbers.some(num => num % 2 === 0);
```

46. **Array.prototype.every():** The every method tests whether all elements in the array pass the test implemented by the provided function.
    **Example:**

```
const numbers = [2, 4, 6, 8, 10];
const allEven = numbers.every(num => num % 2 === 0);
```

47. **Array.prototype.find():**
    The find method returns the value of the first element in the array that satisfies the provided testing function.
    **Example:**

```
const numbers = [10, 20, 30, 40, 50];
const result = numbers.find(num => num > 30);
```

48. **Array.prototype.findIndex():** The findIndex method returns the index of the first element in the array that satisfies the provided testing function.
    **Example:**

```
const numbers = [10, 20, 30, 40, 50];
const index = numbers.findIndex(num => num > 30);
```

49. **Array.prototype.includes():** The includes method determines whether an array includes a certain element, returning true or false.
    **Example:**

```
const numbers = [1, 2, 3, 4, 5];
const hasNumber = numbers.includes(3);
```

50. **Array.prototype.sort():** The sort method sorts the elements of an array in place and returns the sorted array.
    **Example:**

```
const fruits = ['Banana', 'Apple', 'Orange'];
fruits.sort();
```

51. **Array.prototype.reverse():** The reverse method reverses the order of the elements in an array in place.
    **Example:**

```
const numbers = [1, 2, 3, 4, 5];
numbers.reverse();
```

52. **Array.prototype.slice():** The slice method returns a shallow copy of a portion of an array into a new array object.
    **Example:**

```
const numbers = [1, 2, 3, 4, 5];
const subArray = numbers.slice(1, 4);
```

53. **Array.prototype.splice():** The splice method changes the contents of an array by removing or replacing elements and/or adding new elements in place.
    **Example:**

```
const numbers = [1, 2, 3, 4, 5];
numbers.splice(2, 0, 6, 7);
```

54. **Array.prototype.concat():** The concat method returns a new array that combines the elements of the original array with additional arrays and/or values.
    **Example:**

```
const numbers = [1, 2, 3];
const combined = numbers.concat([4, 5], [6, 7]);
```

55. **Array.prototype.map():** The map method creates a new array with the results of calling a provided function on every element in the array.
    **Example:**

```
const numbers = [1, 2, 3];
const squaredNumbers = numbers.map(num => num * num);
```

56. **Array.prototype.filter():** The filter method creates a new array with all elements that pass the test implemented by the provided function.
    **Example:**

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter(num => num % 2 === 0);
```

57. **Array.prototype.reduce():** The reduce method reduces the array to a single value by executing a reducer function on each element.
    **Example:**

```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((acc, num) => acc + num, 0);
```

58. **Array.prototype.flat():** The flat method creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.
    **Example:**

```
const numbers = [1, [2, [3]]];
const flatNumbers = numbers.flat(2);
```

59. **Array.prototype.every():** The every method tests whether all elements in the array pass the test implemented by the provided function.
    **Example:**

```
const numbers = [2, 4, 6, 8, 10];
const allEven = numbers.every(num => num % 2 === 0);
```

60. **Array.prototype.some():** The some method tests whether at least one element in the array passes the test implemented by the provided function.
    **Example:**

```
const numbers = [1, 2, 3, 4, 5];
const hasEvenNumber = numbers.some(num => num % 2 === 0);
```

61. **String.prototype.length:** The length property returns the number of characters in a string.
    Example:

```
const message = 'Hello, World!';
console.log(message.length); // Output: 13
```

62. **String.prototype.charAt():** The charAt method returns the character at a specified index in a string.
    **Example:**

```
const message = 'Hello, World!';
const char = message.charAt(0); // Output: "H"
```

63. **String.prototype.concat():** The concat method combines two or more strings and returns a new concatenated string.
    **Example:**

```
const greeting = 'Hello';
const name = 'Alice';
const message = greeting.concat(', ', name);
```

64. **String.prototype.indexOf():** The indexOf method returns the index of the first occurrence of a specified value in a string.
    **Example:**

```
const message = 'Hello, World!';
const index = message.indexOf('World');
```

65. **String.prototype.lastIndexOf():** The lastIndexOf method returns the index of the last occurrence of a specified value in a string.
    **Example:**

```
const message = 'Hello, World!';
const index = message.lastIndexOf('o');
```

66. **String.prototype.slice():** The slice method extracts a section of a string and returns it as a new string.
    **Example:**

```
const message = 'Hello, World!';
const subString = message.slice(7, 12);
```

67. **String.prototype.substring():** The substring method extracts a section of a string and returns it as a new string.
    **Example:**

```
const message = 'Hello, World!';
const subString = message.substring(7, 12);
```

68. **String.prototype.replace():** The replace method replaces a specified value with another value in a string.
    **Example:**

```
const message = 'Hello, World!';
const newMessage = message.replace('World', 'Universe');
```

69. **String.prototype.split():** The split method splits a string into an array of substrings based on a specified separator.
    **Example:**

```
const sentence = 'JavaScript is awesome';
const words = sentence.split(' ');
```

70. **String.prototype.toLowerCase():** The toLowerCase method converts a string to lowercase.
    Example:

```
const message = 'Hello, World!';
const lowerCaseMessage = message.toLowerCase();
```

71. **String.prototype.toUpperCase():** The toUpperCase method converts a string to uppercase.
    **Example:**

```
const message = 'Hello, World!';
const upperCaseMessage = message.toUpperCase();
```

72. **String.prototype.trim():** The trim method removes whitespace from both ends of a string.
    **Example:**

```
const message = ' Hello, World! ';
const trimmedMessage = message.trim();
```

73. **String.prototype.startsWith():** The startsWith method checks if a string starts with a specified substring.
    **Example:**

```
const message = 'Hello, World!';
const startsWithHello = message.startsWith('Hello');
```

74. **String.prototype.endsWith():** The endsWith method checks if a string ends with a specified substring.
    **Example:**

```
const message = 'Hello, World!';
const endsWithWorld = message.endsWith('World!');
```

75. **String.prototype.includes():** The includes method checks if a string contains a specified substring.

**Example:**

```
const message = 'Hello, World!';
const hasHello = message.includes('Hello');
```

76. **String.prototype.match():** The match method searches a string for a match against a regular expression and returns the matches as an array.
    **Example:**

```
const message = 'Hello, World!';
const matches = message.match(/l+/g);
```

77. **String.prototype.search():** The search method searches a string for a specified value and returns the index of the first match. **Example:**

```
const message = 'Hello, World!';
const index = message.search('World');
```

78. **String.prototype.charCodeAt():** The charCodeAt method returns the Unicode value of the character at a specified index in a string.
    **Example:**

```
const message = 'Hello, World!';
const charCode = message.charCodeAt(0);
```

79. **Math.abs():** The abs method returns the absolute value of a number.
    **Example:**

```
const num = -5;
const absoluteValue = Math.abs(num);
```

80. **Math.ceil():** The ceil method rounds a number up to the nearest integer.
    **Example:**

```
const num = 4.3;
const roundedUp = Math.ceil(num);
```

81. **Math.floor():** The floor method rounds a number down to the nearest integer.
    **Example:**

```
const num = 4.7;
const roundedDown = Math.floor(num);
```

82. **Math.round():** The round method rounds a number to the nearest integer.
    **Example:**

```
const num = 4.5;
const rounded = Math.round(num);
```

83. **Math.random():** The random method generates a random floating-point number between 0 (inclusive) and 1 (exclusive).
    Example:

```
const randomNumber = Math.random();
```

84. **Math.max():** The max method returns the highest value from a list of numbers.

**Example:**

```
const numbers = [1, 5, 3, 7, 2];
const maxNumber = Math.max(...numbers);
```

85. **Math.min():** The min method returns the lowest value from a list of numbers.
    **Example:**

```
const numbers = [1, 5, 3, 7, 2];
const minNumber = Math.min(...numbers);
```

86. **Math.pow():** The pow method raises a base number to the power of an exponent.
    **Example:**

```
const base = 2; const exponent = 3;
const result = Math.pow(base, exponent);
```

87. **Math.sqrt():** The sqrt method returns the square root of a number.
    **Example:**

```
const num = 16; const squareRoot = Math.sqrt(num);
```

88. **Number.prototype.toFixed():** The toFixed method formats a number using fixed-point notation with a specified number of digits after the decimal point.
    **Example:**

```
const num = 3.14159;
const formattedNum = num.toFixed(2);
```

89. **Number.prototype.toPrecision():** The toPrecision method formats a number to a specified length.
    **Example:**

```
const num = 3.14159;
const formattedNum = num.toPrecision(3);
```

90. **Number.prototype.toString():** The toString method converts a number to a string.
    **Example:**

```
const num = 42;
const strNum = num.toString();
```

91. **Date.prototype.getFullYear():** The getFullYear method returns the year (4 digits) of a Date object.
    **Example:**

```
const date = new Date();
const year = date.getFullYear();
```

92. **Date.prototype.getMonth():** The getMonth method returns the month (0-11) of a Date object.
    **Example:**

```
const date = new Date();
const month = date.getMonth();
```

93. **Date.prototype.getDate():** The getDate method returns the day of the month (1-31) of a Date object.

**Example:**

```
const date = new Date();
const day = date.getDate();
```

94. **Date.prototype.getHours():** The getHours method returns the hour (0-23) of a Date object.
**Example:**

```
const date = new Date();
const hours = date.getHours();
```

95. **Date.prototype.getMinutes():** The getMinutes method returns the minutes (0-59) of a Date object.
**Example:**

```
const date = new Date();
const minutes = date.getMinutes();
```

96. Date.prototype.getSeconds(): The getSeconds method returns the seconds (0-59) of a Date object.
**Example:**

```
const date = new Date();
const seconds = date.getSeconds();
```

97. **Date.prototype.toISOString():** The toISOString method converts a Date object to a string representation in ISO format.
**Example:**

```
const date = new Date();
const isoString = date.toISOString();
```

98. **Date.prototype.getTime():** The getTime method returns the number of milliseconds since January 1, 1970 (Unix epoch).
**Example:**

```
const date = new Date();
const milliseconds = date.getTime();
```

99. **Date.prototype.getDay():** The getDay method returns the day of the week (0-6) of a Date object.
**Example:**

```
const date = new Date();
const dayOfWeek = date.getDay();
```

100. **Date.prototype.toUTCString():** The toUTCString method converts a Date object to a string representation in UTC format.
**Example:**

```
javascript const date = new Date();
const utcString = date.toUTCString();
```