Basic JavaScript Concepts : 2 months

- **Syntax and Variables**: Understanding basic syntax rules, declaring variables (var, let, const), and data types.
- **Operators**: Arithmetic, assignment, comparison, logical, and bitwise operators.
- **Control Flow**: Using if, else, switch, for, while, and do-while statements.
- **Functions**: Defining functions, function parameters, return statements, and function invocation.
- **Arrays and Objects**: Creating and manipulating arrays and objects.
- **Strings**: Basic string manipulation, concatenation, accessing characters, and string methods.
- **Scope and Hoisting**: Understanding block scope, function scope, and hoisting behavior.
- **DOM Manipulation**: Basic interaction with the Document Object Model (DOM), accessing and modifying elements.

Intermediate JavaScript Concepts : 2 months

- **ES6+ Features**: Arrow functions, template literals, destructuring assignments, spread/rest operators, etc.
- **Closures and Scopes**: Understanding closures, lexical scope, and closure use cases.
- **Asynchronous Programming**: Callbacks, Promises, and asynchronous functions (async/await).
- **Error Handling**: Using try-catch blocks, throwing and catching errors.
- **Event Handling**: Responding to user interactions with event listeners.
- **Modules and Module Systems**: Working with ES6 modules, CommonJS, AMD, or UMD.
- **Prototype and Inheritance**: Understanding prototypes, inheritance chains, and prototype property.
- **Regular Expressions**: Creating and using regular expressions for pattern matching.
- **Functional Programming**: Concepts like pure functions, immutability, higher-order functions, etc.

Advanced JavaScript Concepts : 2 months

- **Advanced DOM Manipulation**: Dynamic DOM manipulation, virtual DOM, complex event handling.
- **Memory Management**: Understanding garbage collection, memory leaks, optimizing memory usage.
- **Concurrency and Parallelism**: Web Workers, Shared Workers, and dealing with parallel execution.
- **Design Patterns**: Implementation of design patterns like Singleton, Observer, Factory, etc.
- **Optimization Techniques**: Performance optimization, code profiling, and identifying bottlenecks.
- **Metaprogramming**: Proxies, Reflection API, and altering behavior at runtime.
- **Security**: Understanding common security threats, best practices for secure coding.
- **APIs and WebSockets**: Consuming APIs, WebSockets for real-time communication.
- **Testing and Debugging**: Writing unit tests, end-to-end tests, and debugging complex applications.

LOOPS IN JAVASCRIPT :

For-of loop - loop is used to iterate over iterable objects, including arrays, strings, maps, sets, and more.

Here's a list of common data structures in JavaScript on which the `for...of` loop can be applied:

Arrays: The `for...of` loop is commonly used to iterate over the elements of an array.

```
const array = [1, 2, 3, 4, 5];
for (let element of array) {
    console.log(element);
}
```

Strings: Strings are iterable in JavaScript, so you can use the `for...of` loop to iterate over the characters of a string.

```
const str = "Hello";
for (let char of str) {
    console.log(char);
}
```

Maps: The `for...of` loop can iterate over the entries of a Map.

```
const map = new Map([
    ['a', 1],
    ['b', 2],
    ['c', 3]
]);
for (let [key, value] of map) {
    console.log(key, value);
}
```

Sets: The `for...of` loop can iterate over the elements of a Set.

```
const set = new Set([1, 2, 3]);
for (let item of set) {
    console.log(item);
}
```

Typed Arrays: Typed arrays like `Uint8Array`, `Float32Array`, etc., can be iterated over using the `for...of` loop.

```
const uint8Array = new Uint8Array([10, 20, 30]);
for (let num of uint8Array) {
    console.log(num);
}
```

Arguments object: The `arguments` object inside a function can be iterated over using the `for...of` loop (though it's recommended to use rest parameters instead).

```
function example() {
    for (let arg of arguments) {
        console.log(arg);
    }
}
example(1, 2, 3);
```

DIFFERENT CONTENT IN ARRAY DATA SRUCTURE :

// Numbers:

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

// Strings:

const strings = ["apple", "banana", "orange"];

// Booleans:

const booleans = [true, false, true, true];

// Objects:

```
const objects = [
  { name: "Alice", age: 30 },
  { name: "Bob", age: 25 }
];

// Nested Arrays:
const nestedArray = [
  [1, 2, 3],
  ["a", "b", "c"],
  [true, false]
];

// Mixed Data Types:
const mixedData = [1, "apple", true, { name: "Alice" }];

// Empty Arrays:
const emptyArray = [];
```

DIFFERENT CONTENT IN STRING DATA SRUCTURE :

```
// Alphabetic Characters (Uppercase):
const uppercaseLetters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

// Alphabetic Characters (Lowercase):
const lowercaseLetters = "abcdefghijklmnopqrstuvwxyz";

// Numbers:
const numbers = "0123456789";

// Special Characters:
const specialCharacters = "!@#$%^&*()-_+=[]{}|;:',.<>?/";

// Whitespace Characters:
const whitespace = " \t\n\r";

// Unicode Characters:
const unicodeCharacters = "😊🚀🎉";

// Control Characters:
const controlCharacters = "\b\f\v";

// Escape Characters:
const escapeCharacters = "\\\"\'";

// Combined Characters (Mixed Types):
```

```
const combinedCharacters = "ABC123!@#";

// Empty String:
const emptyString = "";
```

DIFFERENT CONTENT IN MAP DATA SRUCTURE :

```
// String Keys with Number Values:
const stringToNumberMap = new Map([
    ["one", 1],
    ["two", 2],
    ["three", 3]
]);

// Number Keys with String Values:
const numberToStringMap = new Map([
    [1, "one"],
    [2, "two"],
    [3, "three"]
]);
```

```
// Object Keys with Array Values:
const objectToArrayMap = new Map([
    [{ name: "Alice" }, ["apple", "banana"]],
    [{ name: "Bob" }, ["carrot", "orange"]],
    [{ name: "Eve" }, ["grape", "kiwi"]]
]);

// Mixed Data Types:
const mixedMap = new Map([
    ["key1", 123],
    [456, "value2"],
    [true, { name: "Alice", age: 30 }]
]);

// Nested Maps:
const nestedMap = new Map([
    ["outer", new Map([
        ["inner1", 10],
        ["inner2", 20]
    ])],
    ["another", new Map([
        ["nested1", "hello"],
        ["nested2", "world"]
    ])]
]);

// Empty Map:
const emptyMap = new Map();
```

DIFFERENT CONTENT IN SET DATA SRUCTURE :

```
// Numbers:
const numberSet = new Set([1, 2, 3, 4, 5]);

// Strings:
const stringSet = new Set(["apple", "banana", "orange"]);

// Booleans:
const booleanSet = new Set([true, false, true]);

// Objects:
const objectSet = new Set([{ name: "Alice" }, { name: "Bob" }]);

// Mixed Data Types:
const mixedSet = new Set([1, "apple", true, { name: "Alice" }]);

// Nested Sets:
const nestedSet = new Set([
  new Set([1, 2, 3]),
  new Set(["a", "b", "c"]),
  new Set([true, false])
]);

// Empty Set:
const emptySet = new Set();
```

DIFFERENT CONTENT IN TYPED ARRAY DATA SRUCTURE :

```
// Uint8Array (Unsigned 8-bit integers):
const uint8Array = new Uint8Array([10, 20, 30, 40, 50]);

// Int16Array (16-bit signed integers):
const int16Array = new Int16Array([-32768, 0, 32767]);

// Uint32Array (Unsigned 32-bit integers):
const uint32Array = new Uint32Array([4294967295, 0, 2147483648]);

// Float32Array (32-bit floating point numbers):
const float32Array = new Float32Array([1.5, 2.25, 3.75]);

// Float64Array (64-bit floating point numbers):
const float64Array = new Float64Array([1.5, 2.25, 3.75]);

// Int8Array (8-bit signed integers):
const int8Array = new Int8Array([-128, 0, 127]);

// Uint16Array (Unsigned 16-bit integers):
const uint16Array = new Uint16Array([65535, 0, 32768]);

// Int32Array (32-bit signed integers):
const int32Array = new Int32Array([-2147483648, 0, 2147483647]);
```