ADVANCE JS:

In-Depth JavaScript Learning Guide :

Autor: Deep Navale

1. Functions in JavaScript

Functions are the heart of any JavaScript program. They are reusable blocks of code that perform specific tasks.

Types of Functions

- **Function Declarations:** Defined using the `function` keyword, followed by a name (optional parameters), and a code block that outlines the function's behaviour.

JavaScript

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

greet("Alice"); // Output: Hello, Alice!
```

- **Function Expressions:** Assigned to variables, offering more flexibility in function definition.

JavaScript

```
const greet = function(name) {
  console.log("Hello, " + name + "!");
};

greet("Bob"); // Output: Hello, Bob!
```

- **Arrow Functions (ES6+):** Provide a concise syntax for defining functions, particularly for single-expression functions.

JavaScript

```
const greet = (name) => {
  console.log("Hello, " + name + "!");
};

greet("Charlie"); // Output: Hello, Charlie!

// For single expressions, omit braces and `return`
const square = x => x * x;
console.log(square(4)); // Output: 16
```

Function Parameters and Arguments

- **Parameters:** Variables listed within the function definition that hold the values passed when the function is called.
- **Arguments:** The actual values passed to the function during its invocation.

JavaScript

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

console.log(add(3, 4)); // Output: 7

// Default parameters (ES6+)
function greet(name = "Guest") {
  console.log("Hello, " + name + "!");
}

greet(); // Output: Hello, Guest!
greet("Alice"); // Output: Hello, Alice!

// Rest parameters   (ES6+)
function sum(...numbers) {
  return numbers.reduce((total, num) => total + num, 0);
}

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

Function Scope and Closures

- **Lexical Scoping:** JavaScript employs lexical scoping, meaning functions inherit the variable scope from where they were defined, not where they are called.

JavaScript

```
function outer() {
  let x = 10;
  function inner() {
    console.log(x);
  }
  return inner;
}

const closureExample = outer();
closureExample(); // Output: 10
```

In this example, `inner` forms a closure, "remembering" the environment (the value of `x`) in which it was created.

Callbacks and Higher-Order Functions

- **Higher-Order Functions:** Functions that take one or more functions as arguments or return a function. They allow powerful abstractions and code organisation.

JavaScript

```
function doOperation(x, y, operation) {
  return operation(x, y);
}

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

function multiply(a, b) {
```

```
  return a * b;
}

console.log(doOperation(5, 3, add)); // Output: 8
console.log(doOperation(5, 3, multiply)); // Output: 15
```

2. Objects in JavaScript

Objects in JavaScript are collections of key-value pairs. They are fundamental to the language and can represent real-world entities.

Creating Objects

- **Object Literals**

JavaScript

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

console.log(person.firstName); // Output: John
console.log(person.fullName()); // Output: John Doe
```

- **Constructor Functions**

JavaScript

```
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
  this.fullName = function() {
    return this.firstName + " " + this.lastName;
  }
```

Continue from where we left off in the previous response:

2. Objects in JavaScript

Objects in JavaScript are collections of key-value pairs. They are fundamental to the language and can represent real-world entities.

Creating Objects

- **Object Literals**

JavaScript

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

console.log(person.firstName); // Output: John
console.log(person.fullName()); // Output: John Doe
```

- **Constructor Functions**

JavaScript

```
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
  this.fullName = function() {
    return this.firstName + " " + this.lastName;
  }
};

const   john = new Person("John", "Doe", 30);
console.log(john.fullName()); // Output: John Doe
```

- **Object.create()**

JavaScript

```
const personPrototype = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

const john = Object.create(personPrototype);
john.firstName = "John";
john.lastName = "Doe";

console.log(john.fullName()); // Output: John Doe
```

Accessing and Modifying Properties

JavaScript

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

// Dot notation
console.log(person.name); // Output: John

// Bracket notation
console.log(person["age"]); // Output: 30

// Adding new properties
person.job = "Developer";
person["salary"] = 50000;

// Modifying properties
person.age = 31;

// Deleting properties
delete person.salary;
```

Prototypes and Inheritance

JavaScript uses prototypal inheritance. Objects can inherit properties and methods from other objects.

JavaScript

```
function Animal(name) {
  this.name = name;
}

Animal.prototype.speak = function() {
  console.log(this.name + " makes a sound.");
};

function Dog(name)    {
  Animal.call(this, name);
}

Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;

Dog.prototype.bark = function() {
  console.log(this.name   + " barks.");
};

const dog = new Dog("Rex");
dog.speak(); // Output: Rex makes a sound.
dog.bark(); // Output: Rex barks.
```

3. Arrays in JavaScript

Arrays are ordered collections of elements. They are versatile and have many built-in methods for manipulation and iteration.

Creating Arrays

JavaScript

```
// Array literal
const fruits = ["apple", "banana", "orange"];

// Array constructor
const numbers = new Array(1, 2, 3, 4, 5);

// Array.from() method
```

```
const arrayFromString = Array.from("Hello");
console.log(arrayFromString); // Output: ['H', 'e', 'l', 'l', 'o']
```

Array Methods

- **Adding and Removing Elements**

JavaScript

```
const arr = [1, 2, 3];

// Add to end
arr.push(4);
console.log(arr); // Output: [1, 2, 3, 4]

// Remove from end
const lastElement = arr.pop();
console.log(lastElement); // Output: 4
console.log(arr); // Output: [1, 2, 3]

// Add to beginning
arr.unshift(0);
console.log(arr); // Output: [0, 1, 2, 3]

// Remove from beginning
const firstElement = arr.shift();
console.log(firstElement); // Output: 0
console.log(arr); // Output: [1, 2, 3]

// Splice - add/remove elements at any position
arr.splice(1, 1, 4, 5); // At index 1, remove 1 element, add 4 and 5
console.log(arr); // Output: [1, 4, 5, 3]
```

- **Iterating Arrays**

JavaScript

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

// forEach
numbers.forEach(num => console.log(num));

// map
const doubled = numbers.map(num => num * 2);
```

```
console.log(doubled); // Output: [2, 4, 6, 8, 10]

// filter
const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // Output: [2, 4]

// reduce
const sum = numbers.reduce((acc,   num) => acc + num, 0);
console.log(sum); // Output:   15
```

- **Other Useful Array Methods**

```
JavaScript

const arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5];

// sort
arr.sort((a, b) => a - b);
console.log(arr); // Output: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

// find
const firstGreaterThanFour = arr.find(num => num > 4);
console.log(firstGreaterThanFour); // Output: 5

// some and every
console.log(arr.some(num => num > 5)); // Output: true
console.log(arr.every(num => num > 0)); // Output: true

// indexOf and lastIndexOf
console.log(arr.indexOf(5)); // Output: 6
console.log(arr.lastIndexOf(5)); // Output: 8
```

4. DOM Manipulation

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a document as a tree-like hierarchy.

Selecting Elements

```
JavaScript

// By ID
const element = document.getElementById("myElement");
```

```
// By class name
const elements = document.getElementsByClassName("myClass");

// By tag name
const paragraphs = document.getElementsByTagName("p");

// Using CSS selectors
const firstButton = document.querySelector("button");
const allButtons = document.querySelectorAll("button");
```

Modifying Elements

JavaScript

```
const element = document.getElementById("myElement");

// Changing content
element.textContent = "New text content";
element.innerHTML = "<strong>New HTML content</strong>";

// Modifying attributes
element.setAttribute("class", "newClass");
element.id = "newId";

// Changing styles
element.style.color = "red";
element.style.fontSize = "20px";

// Working with classes
element.classList.add("highlight");
element.classList.remove("oldClass");
element.classList.toggle("active");
```

Creating and Removing Elements

JavaScript

```
// Creating elements
const newDiv = document.createElement("div");
newDiv.textContent = "I'm a new div";

// Appending elements
document.body.appendChild(newDiv);
```

```
// Inserting before another element
const referenceElement = document.getElementById("referenceElement");
document.body.insertBefore(newDiv, referenceElement);

// Removing elements
const elementToRemove = document.getElementById("removeMe");
elementToRemove.remove();
// Or
elementToRemove.parentNode.removeChild(elementToRemove);
```

5. Event Handling

Event handling allows you to make your web pages interactive by responding to user actions.

Adding Event Listeners

JavaScript

```
const button = document.getElementById("myButton");

button.addEventListener("click", function(event) {
  console.log("Button clicked!");
  console.log("Event   type:", event.type);
  console.log("Target element:", event.target);
});
```

Common Events

- click: Occurs when an element is clicked
- submit: Occurs when a form is submitted
- keydown, keyup: Occur when a key is pressed or released
- mouseover, mouseout: Occur when the mouse enters or leaves an element
- load: Occurs when a page or image has finished loading

Event Delegation

Event delegation allows you to attach a single event listener to a parent element that will fire for all descendants matching a selector, whether those descendants exist now or are added in the future.

JavaScript

```
const list = document.getElementById("myList");

list.addEventListener("click", function(event) {
  if (event.target.tagName === "LI") {
    console.log("Clicked on:", event.
```

Continue from where we left off in the previous response:

5. Event Handling

Event handling allows you to make your web pages interactive by responding to user actions.

Adding Event Listeners

JavaScript

```
const button = document.getElementById("myButton");

button.addEventListener("click", function(event) {
  console.log("Button clicked!");
  console.log("Event   type:", event.type);
  console.log("Target element:", event.target);
});
```

Common Events

- click: Occurs when an element is clicked
- submit: Occurs when a form is submitted
- keydown, keyup: Occur when a key is pressed or released
- mouseover, mouseout: Occur when the mouse enters or leaves an element

- load: Occurs when a page or image has finished loading

Event Delegation

Event delegation allows you to attach a single event listener to a parent element that will fire for all descendants matching a selector, whether those descendants exist now or are added in the future.

JavaScript

```
const list = document.getElementById("myList");

list.addEventListener("click", function(event) {
  if (event.target.tagName === "LI") {
    console.log("Clicked on:", event.target.textContent);
  }
});
```

Preventing Default Behavior

JavaScript

```
const form = document.getElementById("myForm");

form.addEventListener("submit", function(event) {
  event.preventDefault(); // Prevents the form from submitting
  console.log("Form submission prevented");
});
```

6. Asynchronous JavaScript

Asynchronous programming in JavaScript allows you to perform long-running operations without blocking the execution of other code.

Callbacks

JavaScript

```
function fetchData(callback) {
```

```
  setTimeout(() => {
    callback("Data fetched successfully");
  }, 2000);
}

fetchData((result) => {
  console.log(result);
});
```

Promises

Promises provide a more structured way to handle asynchronous operations.

JavaScript

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Data fetched successfully");
      // or reject("Error fetching data");
    }, 2000);
  });
}

fetchData()
  .then(result => console.log(result))
  .catch(error => console.error(error));
```

Async/Await

Async/await is syntactic sugar built on top of promises, making asynchronous code look and behave more like synchronous code.

JavaScript

```
async function fetchAndLogData() {
  try {
    const result = await fetchData();
    console.log(result);
  } catch (error) {
    console.error(error);
  }
}
```

```
fetchAndLogData();
```

7. Error Handling

Proper error handling is crucial for creating robust JavaScript applications.

Try...Catch Statement

JavaScript

```
try {
  // Code that might throw an error
  throw new Error("Something went wrong");
} catch (error) {
  console.error("Caught an error:", error.message);
} finally {
  console.log("This always runs");
}
```

Custom Error Types

JavaScript

```
class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = "ValidationError";
  }
}

try {
  throw new ValidationError("Invalid   input");
} catch (error) {
  if (error instanceof ValidationError) {
    console.log("Validation error:", error.message);
  } else {
    console.log("Unknown error:", error);
  }
}
```

8. JavaScript Modules

Modules allow you to organize your code into separate files and control which parts of your code are accessible to other parts.

Exporting

JavaScript

```
// math.js
export function add(a, b) {
  return a + b;
}

export function subtract(a, b) {
  return a - b;
}

export const PI = 3.14159;
```

Importing

JavaScript

```
// main.js
import { add, subtract, PI } from './math.js';

console.log(add(5, 3)); // Output: 8
console.log(subtract(10, 4)); // Output: 6
console.log(PI); // Output: 3.14159
```

Default Exports and Imports

JavaScript

```
// person.js
export default class Person {
  constructor(name) {
    this.name = name;
  }

  sayHello() {
    console.log(`Hello, my name is ${this.name}`);
  }
```

```
}

// main.js
import Person from './person.js';

const john = new Person("John");
john.sayHello(); // Output: Hello, my name is John
```