1. Javascript
2. Javascript datatypes
3. undefined and null
4. var,let,const
5. hoisting
6. temporal deadzone
7. this keyword
8. call,apply and bind
9. deep copy and shallow copy
10. NaN and IsNaN
11. Promise

Promise.all()

Promise.race()

Promise.any()

Promise.allSettled()

Promise chaining

1. Asyn/await and try/catch
2. Call back function
3. Normal and arrow and anoymous function
4. Closures
5. IIFE
6. Map,filter, forEach
7. slice and splice
8. debouncing and throatling

Program 1: WAP to count character frequencies in a string.

Program2: WAP to sort numbers array

Program3: WAP to sort character array

Program4: WAP to flatten nested array

Program5: WAP for fizz, buzz and fizzbuzz

**JavaScript Prototype**

In programming, prototype is a template or a model from which other objects are created. It defines the properties and methods that objects of a certain type should have. When we create new objects based on this prototype, they inherit these properties and method.

In JavaScript, every object has a hidden internal property called [[Prototype]].this property points to another object called its prototype. JavaScript objects have a special property that is either null or references another object.

When we try to access a property or method on object, JavaScript first looks for that property or method directly on the object. If it cant find it, it looks at the object prototype, and if it is still not found, it continues up the prototype chain until it either find property/method or reaches the end of the chain.

We can set proto type to proto type

const employee = {

  name: "john",

  language: "javascript",

  run: () => {

    alert("first run");

  },

};

console.log(employee);

let protoFunction = {

  run: () => {

    alert("run if method in not in object then it checks in protot");

  },

  protoRun: () => {

    alert("run if proto run method not in first appreance");

  },

};

employee.\_\_proto\_\_ = a.run();

employee.protoRun();

// define proto type for proto type

protoFunction.\_\_proto\_\_ = {

  email: "john@gmail.com",

};

console.log(employee.email);

In JavaScript, object can inherit properties and methods from other objects through a mechanism called prototype inheritance.

**Prototype inheritance**

Every object in JavaScript has a prototype object associated with it. When we try to access a property or method on object, and if it is not found directly on that object, JavaScript looks for it in the object prototype. If the property or method is still not found, JavaScript continues to look up the prototype chain until it either finds the property/method or reaches the end of the chain.

**Prototype chaining**

Prototype chaining is the process by which javascript traverses through the prototype chain to find properties or methods of object. if a property or method is not found directly on object, javascript looks for it in the object prototype.if it is still not found, it continues to look up the prototype chain until it either finds the property/method or reaches the end of the chain.

**2. JavaScript data types**

**Primitive data types**

They are simple and immutable data types, they can’t be changed once created. Primitive data types are stored directly in memory.

**Number**

**String**

**Boolean**

**Bigint**

**Null**

**Undefined**

**Symbol**

**Non-primitive data types/ reference data types**

Non primitive data types are more complex data types that can hold multiple values and have dynamic sizes. They are mutable, their contents can be changed after they are created.

Non-primitive data types are stored by reference, these variable does not directly contain the value but holds a reference to where the value is stored in memory.

**Object**

**Array**

**3. undefined and null**

**“undefined”** represents a variable that has been declared but has not been assigned a value, also the default value of uninitialized variables. When we access a variable that has not been assigned a value, it returns **‘undefined’.**

let x;

console.log(x)

**“null”** represents the intentional absence of any value. It is typically used to explicitly indicate that a variable does not point to any object or does not have a value.

It is often used as a placeholder value to indicate that a variable is intentionally empty or has no value.

let y = null;

console.log(y)

10. NaN and isNaN

NaN stands for “Not a Number”.it is a special value of the number data type. It indicates that a value is not a valid number.

NaN is returned when a mathematical operation cannot produce a meaning ful result.

Division of zero by zero

Multiplication of “Infinity” by “0”

Conversion of a non-numeric string to a number

Performing arithmetic operation involving “NaN”

Usage of NaN

Indicating Errors

When a function that expects a number as input receives an unexpected non-numeric input, it may return “NaN” to indicate that the input was not valid.

Type checking

“NaN” is considered a number data type in JavaScript. Hence, a type check using “typeof” for NaN will return “number”

“NaN” is not equal to any value, including itself. This behavior is due tp the IEEE754 standard fro floating point arithmetic. Therefore, comparison like “NaN===NaN” or “NaN==NaN” will always return “false”.

**isNaN()**

isNaN() is a built-in function in javascript that checks whether a value is “NaN”. It return “true” if the value is “NaN”, and false if value is not NaN.

It is important to note that isNaN() tries to coerce the value to a number before checking if it is “NaN”, which can lead to unexpected results for non-numeric values. To avoid this, it is often better to use “Number.isNaN()”, which performs a strict type check and doesnt coerce the value to a number before checking.

12. async/await, then / catch and try catch

**async/await**

it is a modern syntax for handling asynchronous code in javascript. It provides a more synchronous way of writing asynchronous code, making it easier to read and maintain compared to traditional callback based or promise based code

“async” keyword is used to define a function that will return a promise. Inside async function, we can use the await keyword to pause execution and wait for a promise to resolve before proceeding further.

“await” keyword is use to pause the execution of async function until a promise is settled. It can only be used inside async function.

async function fetchData() {

  try {

    const response = await fetch('https://api.example.com/data');

    const data = await response.json();

    return data;

  } catch (error) {

    console.error('Error fetching data:', error);

    throw error;

  }

}

then()/catch()

then() and catch() are methods available on promise in javascript. They are used to handle asynchronous operation and their result

then() method is used to specify what to do when a promise is resolved. It take a callback function as argument, which will be executed with the resolved value of the promise.

catch() method is used to specify what to do when a promise is rejected. It takes a callback function as argument, which will executed with the error value.

fetch('https://api.example.com/data')

  .then(response => response.json())

  .then(data => console.log(data))

  .catch(error => console.error('Error fetching data:', error))

try/catch

“try/catch” is a javascript construct used for error handling in synchronous code. It allows us to attempt a block of code that might throw error and handle that error gracefully.

“try” block contains the code that we want to try executing. If error occur within this block, it will be caught by the corresponding catch block.

Catch block is used to catch and handle any errors that occur within the try block. It takes error object as argument, which can be used to inspect the error.

try {

  const result = someSynchronousFunction();

  console.log(result);

} catch (error) {

  console.error("An error occurred:", error);

}