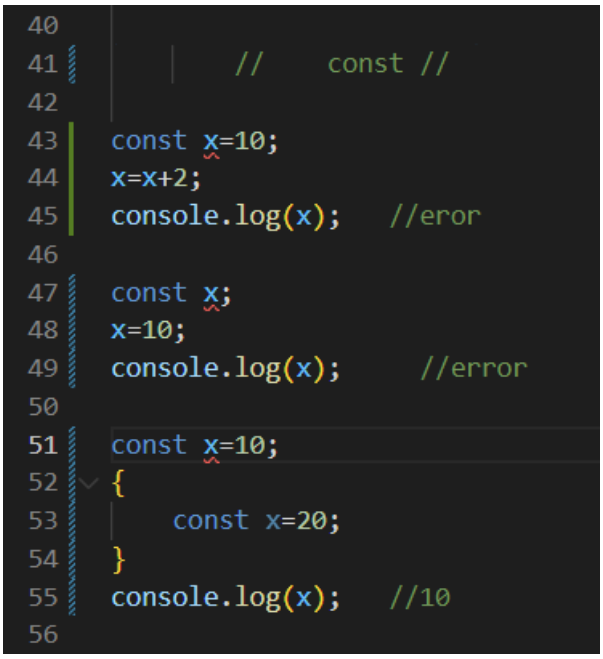
PR-1 Introduction – JavaScript

1. Write a JS program that demonstrate all examples of var, let and const variable declaration.

```
1
2                    //  Var//
3
4   var x=10;
5   var X=20;
6   console.log(X);   //20
7
8   var x;
9   console.log(x);     //undefined
10
11  var x=10;
12  {
13      var x=20;
14
15      //Block Scope or local scope
16  }
17  console.log(x);   //20
18
```

```
19
20                   // let//
21
22  let x=10;
23  let x=20;
24  console.log(x);   //error
25
26  let x=10;
27  {
28      let x=20;
29  }
30  console.log(x);    //10
31
32  let x;     //declaration or defined
33  x=10;       // Initialization
34  console.log(x);
35
36  let x=10;
37  x=x+2;
38  console.log(x);    //12
39
```

PR-1 Introduction – JavaScript



```
40
41          //    const //
42
43  const x=10;
44  x=x+2;
45  console.log(x);   //eror
46
47  const x;
48  x=10;
49  console.log(x);     //error
50
51  const x=10;
52  {
53      const x=20;
54  }
55  console.log(x);   //10
56
```

PR-1 Introduction – JavaScript

2. Write a JS program that demonstrate all examples of different operators.

```
// Arithmetic Operetor

let x = parseInt(prompt("Pls Enter value of x:"));
let y = parseInt(prompt("Pls Enter value of y:"));

let z;

z = x ** y;

console.log(z);
```

```

let x = parseInt(prompt("Pls Enter value of x:"));
let y = parseInt(prompt("Pls Enter value of y:"));

let z;

z = x % y;

console.log(z);
```

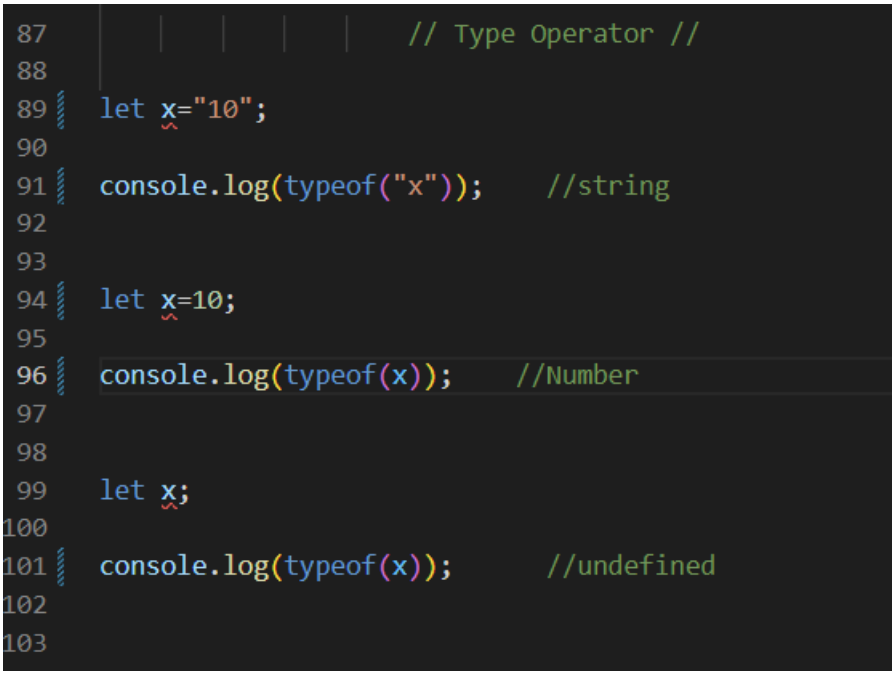
```
let x = parseInt(prompt("Pls Enter value of x:"));
let y = parseInt(prompt("Pls Enter value of y:"));

let z;

z = x / y;

console.log(z);


let x = parseInt(prompt("Pls Enter value of x:"));
let y = parseInt(prompt("Pls Enter value of y:"));

z = x + y;

console.log(z);
```

PR-1 Introduction – JavaScript

```
40
41                  //    Conditional Operators  //
42
43
44  let x=10, y=10;
45
46  console.log(x==y);      // True
47
48
49  let x=10, y="10";
50
51  console.log(x===y);    //False
52
53
54  let x=10, y=10;
55
56  console.log(x!=y);       //False
57
58
59  let x=10, y="10";
60
61  console.log(x!==y);      // True
62
63
64  let x=15, y=10;
65
66  console.log(x>y);         //True
67
68
```

```
69
70                  //  Logical Operator  //
71
72  let x=5, y=5;
73
74  x<=y && x==y ? console.log("True") : console.log("False");   //True
75
76
77  let x=4, y=5;
78
79  x<=y || x==y ? console.log("True") : console.log("False");    // True
80
81
82  let x=5, y=5;
83
84  x!=y ? console.log("True") : console.log("False");     // False
85
```

PR-1 Introduction – JavaScript



```
87                          // Type Operator //
88
89  let x="10";
90
91  console.log(typeof("x"));    //string
92
93
94  let x=10;
95
96  console.log(typeof(x));    //Number
97
98
99  let x;
100
101 console.log(typeof(x));      //undefined
102
103
```

PR-1 Introduction – JavaScript

- **Interview Questions:**

1. What is JavaScript?

Ans:

JavaScript is a programming language that is commonly used for creating interactive and dynamic websites. It is a high-level language, which means that it is designed to be easy for humans to read and write. JavaScript is also a client-side language, which means that it runs on the user's computer, rather than on a server.

JavaScript is often used in combination with HTML and CSS, the other two languages that make up the core of web development. With JavaScript, developers can add functionality to web pages, such as interactive forms, animations, and user interface components. It can also be used to create complex web applications, such as online games, social networks, and productivity tools.

JavaScript is supported by all major web browsers, including Chrome, Firefox, Safari, and Internet Explorer, making it a popular choice for web developers. Additionally, JavaScript has a large and active community of developers who contribute to libraries and frameworks that make it easier to build complex applications.

2. What is advantages and disadvantages of JavaScript?

Ans :

Advantages of JavaScript:	Disadvantages of JavaScript:
Interactivity: JavaScript makes websites more interactive by allowing developers to create dynamic user interfaces, add animations and effects, and respond to user input without having to reload the entire page.	Security concerns: JavaScript code can be vulnerable to attacks such as cross-site scripting (XSS) and cross-site request forgery (CSRF), which can compromise the security of a website or application.
Versatility: JavaScript can be used both on the client-side (in the user's browser) and on the server-side (using Node.js), which makes it a versatile language for building a wide range of applications.	Browser compatibility issues: JavaScript may behave differently across different web browsers, which can make it difficult to create consistent user experiences.
Large community: JavaScript has a large and active community of developers who contribute to open source libraries and frameworks, making it easier for developers to build complex applications.	Performance limitations: JavaScript performance can be slower compared to compiled languages like Java and C++, especially when dealing with large amounts of data or complex algorithms.

PR-1 Introduction – JavaScript

Easy to learn: JavaScript is relatively easy to learn compared to other programming languages, which makes it accessible to beginners who are just starting to learn how to code.	Lack of type checking: JavaScript is a dynamically-typed language, which means that it does not perform type checking at compile time. This can lead to errors and bugs that are difficult to catch during development.
Fast development: JavaScript allows for fast development and prototyping, as it can be added to existing HTML and CSS files without requiring a separate compilation step.	Accessibility concerns: JavaScript can be a barrier to accessibility for users who rely on screen readers or other assistive technologies, as it can interfere with the normal behaviour of these tools.

3. What is the purpose of the let keyword?

Ans:

The let keyword is used to declare block-scoped variables in JavaScript. Prior to the introduction of let in ECMAScript 6 (ES6), the only way to declare variables in JavaScript was to use the var keyword, which declared variables at the function scope or global scope.

The let keyword allows developers to declare variables that are only accessible within a specific block of code, such as inside a loop or a conditional statement. This can help to prevent bugs and improve code clarity, as it makes it clear which variables are only intended to be used within a specific scope.

For example, consider the following code:

```
function foo() {
  var x = 1;
  if (true) {
    var x = 2;
  }
  console.log(x); // outputs 2
}
```

PR-1 Introduction – JavaScript

4.Give the difference between var, let and const.

Ans:

'var'	'let'	'const'
'var' is function scoped, meaning that the variable declared with var is only accessible inside the function in which it is declared, or globally if it is declared outside of any function. var can be redeclared and reassigned within its scope, and it is also hoisted to the top of its scope.	'let' and 'const' are block scoped, meaning that they are only accessible within the block in which they are declared, including for-loops, if-statements, and functions. let can be reassigned but cannot be redeclared within its scope. const, on the other hand, cannot be reassigned or redeclared once it has been assigned a value.	'const' and 'let' are not hoisted to the top of their scope. Therefore, if you try to access them before they are declared, you will get a Reference Error.
Use 'var' when you need to declare a variable that is accessible within the function or globally.	Use 'let' when you need to declare a variable that has a limited scope and can be reassigned.	Use 'const' when you need to declare a variable that has a limited scope and should not be reassigned.

5. What is the difference between =, == and === operator?

Ans:

'='	'=='	'==='
= operator: assigns a value to a variable. For example, x = 5 assigns the value 5 to the variable x.	== operator: compares two values for equality, but performs type coercion if the values have different types. For example, 1 == "1" is true because the string "1" is converted to the number 1 before the comparison.	=== operator: compares two values for equality, but does not perform type coercion. It checks both the value and the type of the operands. For example, 1 === "1" is false because the two operands have different types.
Use = operator to assign a value to a variable.	Use == operator to compare two values for equality, with type coercion.	Use === operator to compare two values for equality, without type coercion.

PR-1 Introduction – JavaScript

6. What is identifier? Give the rules to declare identifier.

Ans:

- An identifier is a name given to a variable, function, or any other user-defined item. It is used to refer to that item in the code. Identifiers must follow certain rules for their declaration:
- An identifier can only contain letters (both uppercase and lowercase), digits, and underscores (_).
- An identifier must begin with a letter, underscore, or dollar sign ($). It cannot begin with a digit.
- Identifiers are case sensitive, meaning that 'myVariable' and 'myvariable' are two different identifiers.
- Identifiers cannot be reserved words, such as 'if', 'while', 'for', 'function', and so on.
- Identifiers should be meaningful and descriptive of the item they refer to. For example, 'firstName' is a more descriptive identifier than 'fn'.

Examples of valid identifiers:

```
myVariable
my_variable
$myVariable
```

Examples of invalid identifiers:

```
123var (starts with a digit)
my-variable (contains a hyphen)
function (reserved word)
```

PR-1 Introduction – JavaScript

7. List features of JavaScript.

Ans:

JavaScript is a powerful and versatile programming language that is widely used for web development. Here are some of its features:

- Dynamic and loosely typed: JavaScript is a dynamic language, meaning that variables can change their type during runtime. It is also loosely typed, allowing variables to be assigned values of different types.
- Client-side scripting: JavaScript runs on the client side of web applications, allowing it to interact with the user and modify the page dynamically.
- Event-driven: JavaScript is event-driven, meaning that it can respond to events such as clicks, input, and page loads.
- Functional programming: JavaScript supports functional programming concepts such as higher-order functions and closures.
- Object-oriented programming: JavaScript is also object-oriented, allowing for the creation of objects with properties and methods.
- Easy to learn: JavaScript has a relatively simple syntax and is easy to learn, making it a popular choice for beginners.
- Cross-platform: JavaScript can run on multiple platforms and devices, including desktops, mobile devices, and servers.
- Large ecosystem: JavaScript has a vast ecosystem of libraries, frameworks, and tools that make it easy to develop complex applications.
- Asynchronous programming: JavaScript supports asynchronous programming through callbacks, promises, and async/await, allowing for non-blocking I/O operations and better performance.
- To summarize, JavaScript is a versatile language that combines both functional and object-oriented programming paradigms and supports asynchronous programming, making it an ideal choice for building dynamic and responsive web applications.

PR-1 Introduction – JavaScript

8. What is the difference between null and undefined?

   Ans

'null'	'undefined'
'null', on the other hand, is a special value that indicates the intentional absence of any object value. It can be assigned to a variable to indicate that there is no object value, or it can be returned as a value by a function that intentionally does not return a value. To summarize, 'undefined' typically occurs when a variable or object property has not been assigned a value, whereas 'null' is explicitly assigned to indicate the absence of an object value.	'undefined' is a primitive value that indicates that a variable or object property has not been assigned a value. It is automatically assigned by JavaScript to a variable when it is declared but not initialized. It can also be returned when trying to access a non-existent object property or function argument.

9. What is the difference between window and document?

Ans

'window'	'document'
The 'window' object represents the browser window that contains the current web page. It is the top-level object in the browser's JavaScript environment and contains many properties and methods related to the window, including the ability to open and close windows, navigate to new pages, and manipulate the size and position of the window. In summary, the 'window' object represents the browser window, while the	The 'document' object, on the other hand, represents the web page that is currently loaded in the window. It provides access to the HTML content of the page, allowing developers to manipulate its structure, content, and styles. It also provides methods to query and manipulate the DOM (Document Object Model), which is a hierarchical representation of the HTML elements on the page.

PR-1 Introduction – JavaScript

'document' object represents the web page loaded in the window. The 'window' object provides methods to manipulate the window itself, while the 'document' object provides methods to manipulate the contents of the web page.	

10. What is local variable and global variable?

Ans

In programming, a variable is a named storage location that holds a value. Variables can be classified into two broad categories: local variables and global variables.

A local variable is a variable that is declared and used within the scope of a function or block of code. Local variables are only accessible within the function or block in which they are declared. Once the function or block has completed its execution, the local variable is destroyed, and its value is no longer available.

For example, consider the following JavaScript code:

```
function myFunction() {
  var x = 10;  // x is a local variable
  console.log(x);
}

myFunction();  // outputs 10
console.log(x); // Uncaught ReferenceError: x is not defined
```

In this example, x is a local variable declared inside the myFunction() function. It is only accessible within the function, and an attempt to access it outside the function will result in a ReferenceError.

PR-1 Introduction – JavaScript

A global variable, on the other hand, is a variable that is declared outside any function or block of code. Global variables are accessible from anywhere in the program, including within functions and blocks. Once declared, a global variable remains in memory for the entire duration of the program's execution.

For example, consider the following JavaScript code:

```
var y = 20; // y is a global variable

function myFunction() {
  console.log(y);
}

myFunction(); // outputs 20
console.log(y); // outputs 20
```

In this example, ‘y’ is a global variable declared outside any function. It is accessible from within the ‘myFunction()’ function and also from outside the function.

In general, it is good practice to minimize the use of global variables in a program to avoid naming conflicts and to make the program more modular and easier to maintain. Instead, variables should be declared with the smallest possible scope, such as local variables within functions or block-scoped variables using ‘let’ or ‘const’.

11. What is NaN property?

    Ans

    In JavaScript, ‘NaN’ stands for "Not a Number" and is a special value that represents the result of an operation that cannot be represented as a valid number.

    The NaN property is a global property that represents the value of NaN. It is a read-only property, which means that it cannot be assigned a new value.

PR-1 Introduction – JavaScript

Here is an example of using NaN in JavaScript:

```
var x = "Hello";
var y = parseInt(x);

console.log(y); // outputs NaN
console.log(typeof y); // outputs "number"
console.log(NaN === NaN); // outputs false
```

In this example, the ‘parseInt()’ function tries to convert the string "Hello" to a number. Since "Hello" cannot be represented as a number, the result is ‘NaN’.

Note that ‘NaN’ is considered a numeric data type in JavaScript, and ‘typeof NaN’ returns "number". However, ‘NaN’ is not equal to any other value, including itself, which means that ‘NaN === NaN’ evaluates to ‘false’.

Developers should be careful when working with ‘NaN’ in their code and use the ‘isNaN()’ function to check if a value is ‘NaN’ or not. The ‘isNaN()’ function returns ‘true’ if the argument is ‘NaN’, and ‘false’ otherwise.

12. Is JavaScript a case-sensitive language?

Ans

Yes, JavaScript is a case-sensitive language. This means that variables, functions, and other identifiers in JavaScript are distinguished by their capitalization.

For example, the following code creates two separate variables, ‘myVariable’ and ‘myvariable’:

```
var myVariable = 10;
var myvariable = 20;
```

PR-1 Introduction – JavaScript

In this code, ‘myVariable’ and ‘myvariable’ are two separate variables, and changing the value of one does not affect the value of the other.

Similarly, function and method names, as well as keywords and operators, are also case-sensitive in JavaScript. For example, ‘if’, ‘If’, and ‘iF’ are three different identifiers in JavaScript, and using the wrong capitalization can result in syntax errors or unexpected behavior.

Here's an example of a case-sensitive error in JavaScript:

```
var greeting = "Hello";

console.log(Greeting); // ReferenceError: Greeting is not defined
```

In this code, the ‘console.log()’ statement tries to access a variable named ‘Greeting’, which does not exist. This is because the variable is actually named ‘greeting’ with a lowercase "g".

Therefore, when writing JavaScript code, it is important to be consistent with capitalization and to pay close attention to spelling and naming conventions to avoid errors and bugs in your code.

13.What is ECMAScript?

Ans

ECMAScript is a standardized scripting language that forms the basis of JavaScript, as well as several other programming languages. It was created by the European Computer Manufacturers Association (ECMA) to provide a standard specification for scripting languages, and the first version of ECMAScript was released in 1997.

JavaScript is the most widely-used implementation of ECMAScript, and new versions of ECMAScript are often referred to by the year they were released. For example, ECMAScript 2015, also known as ES6, introduced several new features and improvements to the language, such as arrow functions, template literals, and let and const for declaring variables.

PR-1 Introduction – JavaScript

ECMAScript provides a standard set of rules and guidelines for writing JavaScript code, including syntax, data types, and control structures. This standardization helps ensure that JavaScript code can be run consistently across different browsers and platforms, and also provides a common language and toolset for web developers to use.

Today, ECMAScript is an essential part of web development, and new versions of the standard are released regularly to introduce new features and improvements to the language.

14. What are the benefits of initializing variables?

Ans :

Initializing variables means assigning a value to a variable before it is used in a program. The benefits of initializing variables are:

Preventing errors: If a variable is not initialized, it will contain garbage or undefined values. This can lead to errors or unexpected results when the variable is used. By initializing the variable with a specific value, you ensure that it has a predictable value, and can prevent errors.

Improved readability: Initializing variables makes code more readable, as it makes it clear what the value of the variable is supposed to be. This can help other programmers who read your code to understand it more easily.

Better debugging: Initializing variables can help with debugging, as it can help you identify the cause of errors more easily. If a variable is not initialized and is causing errors, it can be difficult to determine where the error is occurring. By initializing the variable, you can narrow down the potential causes of the error.

Optimization: In some cases, initializing variables can improve program performance. For example, if a loop variable is initialized to zero before the loop, the loop can run faster than if the variable is not initialized.

Overall, initializing variables is a good programming practice that can help prevent errors, improve readability and debugging, and sometimes even optimize program performance.