```
Notes Prepared By : Ajinkya Ban
Email : ajinkyaban@gmail.com
====================================
*Javascript*
------------------------------
Javascript is the most popular scripting langauge in the world. Javascript can work on both server side as well client side for example to validate form data.

What is mean by scripting langauge?
---------------------------------------------------
1. compiler : compiler scan the code at once. It displays all the error at once.

What is the work of compiler?
----------------------------------------------
To translate human readable code into machine understandable format.Machine can only understand binary langauge(in the form 0's and 1's).

Ajay ---> 00111001100 0011100101110011001100011000110 1



2. interpreter : Interpreter can execute the code line by line. It display error one at a time.
        Interpreter require more time then compiler to execute the code. This an issue raised by developer team. To solve this problem the team introduced new


What is mean by scripting language?
---------------------------------------------------
-A script is a program that consist of a sequence of statements. It is wriiten by using the syntax or commands.
- It is emeded in html web page, interpreted and executed by a certain scripting engine.

Why we use script?
--------------------------
Script is written for various intentions such as automating the process on local computer as well as remote computer, to create the web page on web.

Which are the famous scripting langauge?
------------------------------------
1. Javascript
2. VBScript
3. ASP(active server pages)
4. php(hypertext preprocessor)
5. PERL(Practical Extraction and Reporting language)
6. JSP(Java server pages)
7. Python
8. Ruby

What is mean by script and scripting?
---------------------------------------------------
The term script refers to a program.
The term scripting refers to programming.


What are the key features of Scripting language?
-----------------------------------------------------------------
1. Scripting langauges can be directly embedded into an application.
2. They are quite easy to learn and use.
3. They provide all the basic features common to modern programming languages, such as powerful variable types, basic operations like addition, subtraction, lo
4. The scripted code is not usually compiled. It is simply interpreted by the interpreter at runtime.
5. Scripting languages are free and open source, that means a programmer can have full control to view and edit it.
6. The syntax of scripting language is familiar and similar to the syntax of other programming langauge such as c, c++, java.
7. Scripting languages also provides object orianted functionality, like encapsulation, inheritance, polymorphism etc.
8. Scripting languages require less memory from the system because they are not compiled but they are interpreted.
9. Scripting languages are portable and cross platform, as they  can run on remote server or in the visitors web browser on any operating system.
10. Scripting languages uses the concept of exception handling as like other programming langauge.

What are the different types scripting languages?
1. client side script
2. server side script

*What are the advantages of scripting langauges over Programming languages?*
----------------------------------------------------------------------------------------------------
1. Most of the scripting languages free and open source as well as lightweighted, as result anyone from  anywhere around the world can utlize it.

2. Most of the scripting langauges dynamically typed.

3. The coding of scripting langauge is easy to learn and understands. To learn Scripting language there is not required much knowledge of web technology.

4. The main benifit of scripting language is they provide rapid code development.





*What are the disadvantages of scripting languages over programming languages?*
------------------------------------------------------------------------------------------------------
The main disadvantage of scripting language is that they are slower then the compiled languages.



*What are the applications of scripting languages?*
-------------------------------------------------------------------
1. Scripting languages is mostly used in web applications, it can be used in both side that is from client side or server side.
2. They are used in system administration.
3. Scripting languages can be used in game development as well as to develop multimedia system.
4. It can be used to develop plugins and extensions.
5. Using Scripting language we can dynamically change the content of web page.
6. They can be triggered by event.
7. We can easily create graphical user interface using script.

*History of javascript*
-----------------------------------------------
1993  ----> Mosaic ---> static
1994  --->  Netscape Navigator --> Company Name -->American independant computer service company --> The developer was Marc Andersson --> within 10 days develo

Need of netscape:
------------------------
1989 ---> 1996 -->
in java there is one concept that is called Applet Programming ---> To run the applet we need browser --->In 1994 there is only one browser present that is cal
```

```
The java developed by Games Gossling ---> Bill gates  --> Gudio Van Roosum(Python)

- The First GUI based language developed by microsoft team and that language name was visual Basic 6.0.
- Games gossling introduced the concept called applet. This applet developed within 5 days. But there is   problem with applet, applet need browser to run.
- At that time the microsoft has the browser.
- Games gossling developed one browser that is called  Netscape Navigator within 10 days. This browser   developed by brendan eich.
- Todays we called Netscape Navigator as Mozilla Firefox.
- The first name of javascript was Mocha.
- Later the name was changed to LiveScript.
- Due to some treadmark issue the was change to javascipt.




*how to impliment javascript with html?*
--------------------------------------------------
There are four ways to add or embbed javascript into html page.

1. Directly in html page using <script></script>
2. As an event handler using html tag attribute
3. Via the pseudo-URL javascript: syntax within some link (href)
4. by using  .js file.

-------------------------------------------------------------------------------------
1. Directly in html page using <script></script>:
---------------------------------------------
- This tag was introduced by Netsscape Navigator in version 2.
- We can place any number of <script> tags within a web page.
- All the script tag are interpreted in the order they defined.


*Atrributes Added to Script tag*
---------------------------------
1. async : used with external js file
2. src : This attribute add the reference of external js file location.
3. type : This attribute tells us which type of file load on page.
4. defer : This attribute load first when something change happend in js file then it will load html page.
5. charset: utf-8
6. language: This is deperecated in latest js. This attribute introduced in javascript v 1.2.

----------------------------------------------------------------------------------------------------

*2. As an event handler using html tag attribute:*
-------------------------------------------------
- To make a web page more interactive, we can also add javascript code into a event handler using html tag attribute.
- It is the second way to execute javascript code via an event handler.
- We define an event handler to trigger the execution of script in response to user activity such as key press, mouse event.
- All event handler attributes starts with word 'on'.


3. Via the pseudo-URL javascript: syntax within some link (href)
-------------------------------------------------------------------
- This is the third way to pass javascript code via pseudo-URL.

4. by using  .js file
--------------------
By using internal js or by using external js file we add javascript code into html page.

========================================================================================================================

defer attribute full explaination
---------------------------------
- In javascript, the defer attribute is used with <script></script> tag in html to load script async while ensuring that they maintain their execution order.

- When you add defer to a script tag the script file is download in parllel with html document, but is executed by only the html document has been completely p

*key points*
----------------------
1.  script with defer attribute are executed in order they maintain.
2. The script executed after the document has been fully parsed, but before the DOMContentLoaded event.

Example:
1. with defer
2. without defer


*When to use defer*
------------------------------
use defer when you want to ensure that the script does not block the rendering of the html doccuments.


*What is mean by html parsing*
---------------------------------------
"HTML document has been completely parsed".

It means that the browser has finished reading and interpreting the entire html content of the web page.  The entire html page converted into DOM.(Document obj

*Break down the point*
---------------------------
1. Loading : browser downloading the html files
2. Parsing : reading the html tags, text, imgs
3. DOM Construction
4. complete parsing
========================================================================================================================
*Data types and how we can declare the variables in javascript*
----------------------------------------------------------------

What is variables?
----------------------------
- The variable is the memory location in which we can store the value for temporary purpose to perform some operations in    future.
- The values of the variables can stored in temporary memory location called as RAM(Random Access Memory).

What is the syntax of declaring the variable in javascript?
-------------------------------------------------------------
- As we know the javascript is dynamic typed language.
- Which means that we dont need to provide the data type while declaring the variables.
- So in the javascript we can use three keywords to declare the variable.
 a) var
 b) let
 c) const

*What is the difference between var, const and let keyword?*
----------------------------------------------------------------------
1. var:
---------------
- The var keyword first declared in 1995, which is implemented in Netscape Navigator v.2.
- The javascript is also called ECMAScript or it is called ES12.
- The biggest history happen in ES6 in the year 2015. So the several new features were added in ES6.

The following are new features which are introduced in ES6
1. let and const Declarations
2. Arrow Functions
3. Template literals
4. Default parameter to functions
5. Rest & sprade operator
6. classes
7. Modules(import/export)
8. promise

 a) Scope: var is function scoped, if declared inside the a function. It is accessible throughout that function. If it declared           outside the function
 b) hoisting: It will reference a var variable undefined.
 c) Re-declarations : You can re-declare and update a var variable within its scope.

*Example*
----------------------
var name = "html"; //global declarations
var name = "css"
function test()
{
  // local declaration / function scope declaration
  // var name = "java";
  // var name = "python
  console.log(name);
}

test();
console.log(name)

===================================================================================================

2. Let:
 a) Scope: let is block scoped. It is only accessible within the block where it is declared.
 b) hoisting: Accessing the variable before its declaration will result in a ReferenceError.
 c) Re-declarations : You can not re declare a let variable in the same scope, but you can update its value.

*Example*
---------------------------
let name = "html"
function test()
{

  let name = "java"
  let name = "css"; // i cant re declare the variable which has same scope
  name = "python" // i can update the value
  console.log(name);
}
console.log(name);
test();

==========================================================================================================
3. const: Suppose we dont want change the value of the variable then we can use const.

 a) scope : Like let const is block scope.
 b) hoisting : Accessing the variable before its declaration will result in a ReferenceError.
 c) Re-declarations: we can not re declare the value of const. But in case of array and object we can change the value.




Q. What is the difference between undefined and undeclared variable?
--------------------------------------------------------------------------------
undefined : The variable which you can declare but dont provide any specific value to the variable.
undeclared : Those variables which is not present in your code and you trying to print them.



hoisting:
------------------
This is the default present in javascript.  In which variable and function declarations are move to top.

Example:
---------------
console.log(num);  #ReferenceError: num is not defined . TDZ start
var num = 50; # TDZ ends
console.log(num)

*In above declarations we called it TDZ(Temporal Dead Zone):*
--------------------------------------------------------------------------------
TDZ is the time between entering the scope and the actual declartion of let, const and var variables. During this time any reference error that is called TDZ.

*shadowing Example*
-------------------------
- we can not declare the same name in the same scope. But we achive it by using the concept called shadowing.
- The shadowing concept can used with let and const keyword.

let personName = 'abc';

function getData() {
  let personName = 'xyz'; // shadowing
  console.log(personName);
}
```

```
getData();

o/p
-----
xyz


OR using const keyword
---------------------------
const personName = 'abc';

function getData() {
  const personName = 'xyz'; // shadowing
  console.log(personName);
}

getData();
=========================================================================================================

*Data Types in javascript*
----------------------------------
A data type in JavaScript specifies the specific type of value that a variable can store. In other words, a data type is simply a ”kind” of value (i.e. data) s

Internally, a variable represents a memory location where data stores. Data is information that we store in computer programs. For example, name, age, the colo

The data type in JavaScript determines how much memory will allocate for the data stored in a variable.

*What is Dynamic Data Types in JavaScript?*
----------------------------------------------
avaScript is a dynamic and loosely typed, or duck typed language. It means that we do not need to specify the type of variable because JavaScript engine dynami

In other words, the value we assign to the variable determines its data type. JavaScript automatically handles different types of values and automatically conv

That is, it does not prohibit a variable from changing the type of value while programming.

JavaScript simply allows for creating a generic variable using the keyword var. The actual data type used depends on the initial value that we will assign to t

Consider the following examples:

var FirstName = ”Shubh” // Holding string.
var AccountNumber = 123499 // Holding number.

*Types of Data types in JavaScript*
--------------------------------------------
Like any other programming language, JavaScript language allows us to work with the seven data types. A list of all data types are as follows:

Numbers
Strings
Booleans
Null
-------------
- Null in javascript means empty value and it is also primitive type.
- The variable which assign null contains no value or it is called empty value.
- When we check the type null then it will return object type. This is open issue by javascript.

*Why is Null an object*
-------------------------------
- Basically, null is primitive type of variable in javascript. Many people consider it a bug in javascript.
- But javascript considered it as an object.
- Changing this bug will break the existing code.
- The history behind this is that,
- When the null is with arithmatic operator

10  ---> 1010 ---> 32bit ---> 0000 0000 0000 0000 0000 0000 0000 1010

Undefined
Objects
Arrays

*JavaScript Primitive Data Types*
-----------------------------------

Primitive data types are the set of all basic building blocks for which data represents. They are manipulated directly by their values. In JavaScript, there ar

Numbers
Strings
Boolean

Primitive data types in JavaScript can only store a single value within the associated variable. It cannot store additional properties or methods along with th

By default, primitive data types are immutable in JavaScript, meaning that once a primitive value is created, the value itself cannot be altered. Any operation

For example, adding to a string results in a new string, because the original string cannot be changed. However, a newly created value can be initialized to an

1. Numbers:
----------------
Numbers in JavaScript are used to represent both integers and floating-point numbers. For example, we can represent your age as a number. Numbers look like thi

Like other languages like C, C++, Java, JavaScript language do not provide special or separate data types for integers and floating-point numbers. It does not

JavaScript language represents numbers by using the 64-bit floating-point format defined by the IEEE 754 standard. Let’s consider a very simple example program

Program code 1:
------------------------

<script>
      var x1 = 5;
      var x2 = 10;
      var x3 = 15;
      var x4 = 20.25;
     document.write(x1 + "<br>"); // It simply prints or display on the current webpage.
     document.write(x2 + "<br>");
     document.write(x3 + "<br>");
     document.write(x4);
</script>
Output:
```

```
            5
            10
            15
            20.25

2. Strings:
----------------------
A string is another type of data that comprises a sequence of alphanumeric characters. It is basically a line of text surrounded by quotation marks.

For example, “Hello World!”, “666-2829”, “KA23VABSN”, etc. Enclosing a group of characters, or a line of text in a single or double quote shows it is a string

var name = "Shubh"; // String.
alert( name ); // This will alert "Shubh".
When wrapping a number in quotes and assigning it to a variable, JavaScript will consider the number as a string. For example:

var num = "5";
alert( num + num ); // This will alert "55".


We know that plus sign (+) is used to add numbers. When we will use plus sign (+) with strings, the plus sign concatenates strings together like this:

var firstName = "Shubh";
var lastName = "Deep"
document.write(firstName + lastName); // It will print ShubhDeep.
If we add a number and string, JavaScript will assume the number as a string. For example:

1. var x = "5";
    var num = "five";
    document.write(x + num); // It will print 5five.

2. var num = 25 + 30 + "five";
    document.write(num); // It will print 55five because JavaScript will treat 25 and 30 as numbers. That’s why it is adding.

3. var num = "five" + 20;
    document.write(num); // Print five20.

4. var num = "five" + 20 + 30;
    document.write(num); // Print five2030 because the first operand is a string.
   // Therefore, JavaScript interpreter will consider all operands as strings.


3. Boolean:
--------------------------
We can also assign a variable with boolean values (true or false). JavaScript provides built keywords true and false to define boolean values, so quotation mar

For example:

var booleanValue = true; // The variable booleanValue is now true.
We often use boolean values in conditional testing. For example, consider the following statements:

If (num > 18) {
   // do something
}
In this example, we have used a boolean expression (num > 18) within the if statement to check whether the code within braces will execute.

If the variable num is greater than 18, the Boolean expression evaluates to true; otherwise, it evaluates to be false.


*JavaScript Non-primitive Data Types (Reference Data Types)*
-------------------------------------------------------------------

There are three non-primitive data types in JavaScript. Non-primitive data types are also called composite or reference data types in JavaScript. They are as f

Objects
Arrays
Functions
These non-primitive data types are manipulated by reference. Non-primitive data types in JavaScript are capable of storing multiple values under a single varia

Let’s understand all three non-primitive data types in JavaScript with example programs one by one.

1. Objects:
--------------------------
Objects in JavaScript are a set of properties, each of which can contain a value. We write these properties as name: value pairs (or key: value pairs), separat

Each value stored in the properties can have a value, another object, or even a function. We can also define our own objects or can use several built-in object

Javascript objects are created with curly braces. Let’s understand them with help of some examples.

a) var myObject = { }; // an empty object.

b) An object with several properties:
    var person = {
                              firstName: "Ivaan",
                              lastName : "Sagar",
                              age: 04,
                              eyeColor: "blue",
                              hairColor: "black"
                            };
The preceding code example creates an object called person, that has five properties: firstName, lastName, age, eyeColor, and hairColor.

The values contained in each property are Ivaan, Sagar, 04, blue, and black respectively.

The .operator is used to access the property of an object. For example, we can access the lastName property of the person object like this:

document.write(person.lastName); // It will print Sagar.
The complete script code structure is as follows:

<script>
var person = {
               firstName: "Ivaan",
               lastName: "Sagar",
               age: 04,
               eyeColor: "blue",
               hairColor: "black"
    };
  document.write("Full name : " +person.firstName + " " +person.lastName+ "<br>");
  document.write("Age: " +person.age +"<br>");
  document.write("Eye color: " +person.eyeColor+ "<br>");
  document.write("Hair color: " +person.hairColor)
</script>
Output:
           Full name: Ivaan Sagar
           Age: 4
           Eye color: blue
           Hair color: black


2. Arrays:
---------------------
An array is a group of multiple values that we can assign into a single variable. The values in an array are unsigned integers called array index. The array in

Therefore, the index number 0 contains the first member, the index 1 contains the second member, index 2 has the third member, and so on. We write an array wit

Let’s understand it with the help of an example program.

<script>
 var num = [20, 40, 50];
 document.write(nums[0], "<br>");
 document.write(nums[1], "<br>");
 document.write(nums[2]);
</script>
Output:
           20
           40
           50

3. Functions:
------------------------
A function is a block of JavaScript code that is written once and execute when “someone” calls it. JavaScript provides some pre-defined functions and we can al

The definition of a function looks like this:

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

----------------------------------------------------------------------------------------------------
*Type Conversion and Type Coercion:*
--------------------------------------
There are two types of conversion in javascript
1. explicit conversion : This explicit conversion means to convert one data type into another data type by using predefined functions.
Example:
-----------------------------
let numValue = 1234;
let strValue = String(numValue);


console.log(typeof numValue);
console.log(typeof strValue);

2. implicit conversion: In this conversion you dont need to specify the predefined functions, javascript automatically take care of this.

Example:
-------------------
let strValue = "100";
let numValue = 5;

let result = strValue + numValue;
console.log(typeof result);

---------------------------------------------------------------------------------------------------


*3. Operators and Expressions*
------------------------------------
*What is mean by operator?*
-------------------------------
Def : Operator can be used for to perform the operations on oprands.

Example:
------------
var a =30
var b = 40

console.log(a + b)

*Depend on the oprands there three types of operators*
-------------------------------------------------------
a. unary operator  : This operator work on single value.
b. binary operator : This operator can work on two values
c .ternerry operator : This operator work on three values.

*Types of javascript operator:*
----------------------------
1.  Arithmatic operator
2. logical operator
3. Relational / Comparision operator
4. Bitwise operator
5. Assignment Operator
6. String operator


1.  Arithmatic operator/ Mathematical Operator:
--------------------------------------------
Arithmetic operators in javascript can be used for to perform most common mathematical calculation.

List of operator:
---------------------
1. + --> to perform the addition
2. - --> to perforom subtraction
3. * ---> to perform the multiplication
4. / ---> to perform the dividation
```

```
5. % --> it calculate the reminder

- Javascript automatically perform type conversion when working with mathemetical operators.



*2. Bitwise operator:*
------------------------------
- Bitwise operator can work on bit level. Bit level means either the value present in 0's and 1's.
- JavaScript bitwise operators only work with 32bit integer numbers.
- In bitwise operators the internal representation of integer number is represented by binary number system.

*Types of bitwise operators in javascript*
-----------------------------------------
1. & ---> bitwise And
2. | ---> bitwise OR/ pipe operator
3. ^ ---> bitwise xor / cap operator / carret operator
4. ~ --> bitwise NOT / tild operator


1. & ---> bitwise And
-------------------------------
Q. 10 & 11 ---> ?

- In bitwise operator when we pass the decimal number the decimal number first converted into binary number then process the number and finally provide the out

Truth table of AND(multiplication) operator
------------------------------------
X Y Result
0 0 0
1 0 0
0 1 0
1 1 1

//decimal to binary conversion

// let decimalNum = 10;
// let binaryNum =  decimalNum.toString(2);
// console.log(binaryNum);
-------------------------------------------------------

//binary to decimal

let binaryNum = 1010;
let decimalNum = parseInt(binaryNum,2);
console.log(decimalNum);

Example:
------------------
let a = 20;
let b = 11;

console.log(a & b);

/*
1. The decimal number convert into binary
10 ---> 8 bit ---> 0000   1  0   1     0
11 ---> 8bit --->  0000   1  0    1     1
----------------------------------------------
Result ----->      0000   1010

20 ---> 00010100
11 --->  00001011
----------------------------------------
            00000 00
*/



*2. | ---> bitwise OR/ pipe operator*:
---------------------------------------------
Truth table of OR operator
----------------------------
X Y Result
0 0 0
1 0 1
0 1 1
1 1 1

*Example*
-------------------
let a = 20;
let b = 11;

console.log(a | b);

/*
1. The decimal number convert into binary
10 ---> 8 bit ---> 0000   1  0   1     0
11 ---> 8bit --->  0000   1  0    1     1
----------------------------------------------
Result ----->      0000    1011

20 ---> 00010100
11 --->  00001011
----------------------------------------
                00011111
*/


*3. ^ ---> bitwise xor / cap operator / carret operator:*
---------------------------------------------------------
The truth of table of XOR
--------------------------
X Y Result
0 0 0
```

```
1 0 1
0 1 1
1 1 0

Example:
-----------------------
let a = 20;
let b = 11;

console.log(a ^ b);

/*
1. The decimal number convert into binary
10 ---> 8 bit ---> 0000   1  0   1     0
11 ---> 8bit --->  0000   1  0    1     1
----------------------------------------------
Result ----->      0000    0001

20 ---> 00010100
11 --->  00001011
----------------------------------------
                        00011111
*/


4. ~ --> bitwise NOT / tild operator:
----------------------------------
0 -->  1
1 ---> 0

*Example*
------------------
let a = 10;
let b = 11;

console.log(~(a));
console.log(~(b));

/*
10 ---> ~ 1010 ---> 0101
*/

------------------------------------------------------------------------------------------------

*3. Assignment Operator:*
--------------------------
- An operator that is used for to assign/update the value of the variable.
- In javascript there is no concept of increment and decrement operator like in other programming language, that's why we use
   the assignment operators.
- The most common assignment operator is equal operator "=", which assign the right hand value to the left hand variable.

- The general syntax
 *variable = value;*

*Types of Assignement operator in javascript*
--------------------------------------------------
1. simple assignment operator
2. compund assignment operator


*1. simple assignment operator*
-----------------------------------------
A simple assignment operator can be used for to store or assign a value into variable or to store a value into another variable.

Syntax:
--------------
variable = value;


Example:
----------------
var num = 20;
var num2 = num;



*2. compound assignment operator*:
------------------------------------------
The compound assignment operator can only work with binary operators. The compound assignment operator is also called as
shorthand operators.

Syntax:
----------------
var variable = expression

*Example*
------------------
var x = 30;

I want to add 10 into x,

x = x + 10 or you can do this in another way,
x += 10   ---> shorthand operator

 *In assignment operator we can use the following operators,*
 ---------------------------------------------------------------
 1. = ---> equal assignment operator
 2. +=  ---> addition assignment operator
 3. -=  ---> subtraction assignement op
 4. *= ---> multiplication assignement op
 5. /=  ---> division assignment op
 6. %=  ---> reminder assignment op.

 Example:
 ------------------
let x = 30;
let y = 50;
y -=x;
console.log(y);



*4.Comparison or Relational Operators*
---------------------------------------------
- Comparision operator in javascript are those operator that are used for to comparing between two or more then two values.
- The comparision operator always return true or false value.
- The comparision operator mainly used with decision making statements like if, else if...etc.

*Types of comparision operators*
----------------------------------
1.  ==    ---> equals to
2. !=     ---> Not equal
3. >      ---> greater then
4. <     ----> less then
5. >=   ----> greater then or equal
6. <=   ----> less then or equal
7. === ---> strict equals then compare data types as well.
8. !===  --> not strict equals / strictly not equals

*Examples*
-----------------------
let x = 10;
let y = 4;
let z = 30;

console.log(x == y);
console.log(x > y)
console.log(x > y && y > z);
console.log("java" > "python");// assci values -->
console.log("python" < "ruby");// assci values -->
console.log("Suresh" > "Rahul" && "Rahul" < "Yogesh");
console.log(true > false)
console.log(true === false)

-----------------------------------------------------------------------------

*Comparision operators and Data Conversions*
---------------------------------------------------
- We can also have two operands of different data types within single comparision. In this case, javascript always tries to convert
each operand into number before running the comparision.

Case 1:
--------------------
If one operand is type of string and another is type of number. JavaScript is to tries to convert string into number.

Example:
----------------
let num1 = 30;
let num2 = '30';

var result = num1 == num2;
console.log("Result is = "+result);

Case2 :
---------------------------
If the string can not be converted into number, comparision always return false.


Example:
--------------------
let num1 = 30;
let num2 = 'java';

var result = num1 == num2;
console.log("Result is = "+result);

Case 3:
----------------------
If one operand is boolean type and another is a number, JavaScript convert the boolean to a number.

Example:
----------------
let num1 = 1;
let num2 = 'java';
let num3 = true // here true is 1 and false is 0

var result = num1 == num3;
console.log("Result is = "+result);


Case 4:
----------------------
If one value is boolean type and another value is string type, The javascript convert boolean into number and string into number.

Example:
----------------
let num1 = 1;
let num2 = '1';

var result = num1 == num2;
console.log("Result is = "+result);



5. Logical Operators:
-------------------------------
Logical operator in javascript are those operator which are used to form compound conditions by combining two or more simple
conditions.
- In other word, logical operator combine the comparision into one condition group.
- The logical operator is sometine called boolean operators, because they always return either true or false value.
- With the help of boolean operator we can perform logical operations.
```

```
*Types of logical operators*
-----------------------------
1. &&   ---> logical and
2. ||     ---> Logical OR
3. !      ---> Logical Not


6. Conditional operators:
-------------------------------
- Conditional operators in javascript provides one line approach for creating a simple conditional statement.
- It is often used as shorthand method for if-else. It makes the code much more simple.
- The conditional operator(?:) is also known as ternary operator.
- The basic syntax of ternary operator as below,
Syntax:
----------------
condtional_expression ? Value1 : Value2

Or

Variable = conditional_expression ? Value1 : Value2


- The above syntax, either return true value or false value.

*Example*:
------------------

let x = 20;
let y = 40;

// if (x > y) {
//   console.log(x);
// } else {
//   console.log(y);
// }

let z = x > y ? x : y;
console.log(z);

Example-2
-----------------
let age = prompt("Enter the age = ")
let check_eligible = age >= 18? "You are eligible for voting":"You are not eligible voting";
console.log(check_eligible);

========================================================================================================================

*4. Basic input and output in javascript*
----------------------------------------------
- In javascript basically 3 types dialouge box are present,
1. alert()
2. prompt()
3. confirm()

*1. prompt()* :
------------------
- The prompt() method in javascript is used to display prompt box that accept user input.
- It is generally used to take the input from the user.
- It can be written without using window prefix or object.
- When the prompt box pops up, we have either click "Ok" or "Cancel" button.
- The box is displayed using prompt() method, it will take two arugmuments: The first argument is label which display text, and       second argument is the de


Example -3
------------------
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="text/javascript">
        function myFun()
        {
            var result = prompt("Please enter your name");
            document.getElementById("prompt").innerHTML = "Hello, " + result;
            document.getElementById("prompt").style.color = "blue";
            document.getElementById("prompt").style.fontSize = "24px";
            document.getElementById("prompt").style.backgroundColor = "lightgray";
        }
    </script>
</head>
<body>
    <button type="button" onclick="myFun()">Click Me</button>
    <p id="prompt"></p>
</body>
</html>


*2. Confirm():*
-------------------------
- JavaScript confirm method invokes a function that asks the user for a confirmation on particular action.
- The confirm() method uses window object.
- The confirm() method has two buttons one is "Ok" and other is "Cancel".
- If the user click on the ok button it will execute further action,otherwise it returns on the same page.

*What are the usage of confirm()*
-------------------------------------
1. The javascript confirm() method is used to display the specific  message on a browser with OK and Cancel option to confirm user action.
2. It stop all the actions until the confirmation window is closed.
3. For dealing crud operations.

*Examples*
------------------
let name = prompt("What is your name?");
```

```
if(name != null && name !== "")
{
    let isCorrect = confirm("Is your name " + name + "?");
    if(isCorrect)//In the if block the value of the variable is true means 1
    {

        alert("Hello " + name + "!");
    }
    else{
        alert("name is not correct");
    }
}
else{
    alert("No name entered");
}

========================================================================================================================

*5. Conditional Statements/ Control Statements:*
------------------------------------------------
- A simple javascript program consists of set of statements that generally contain expressions and ends with semicolon.
- When we execute a javascript program, at time only one statement executed by javascript interpreter. This is called sequentail   statement execution.
- These statements executed from top to bottom one by one, from this we can say that the execution takes place from top to     bottom.
- However, if we want to change the flow of the program we can use the control statements.
- Flow control statements in javascript are those statement that change the flow execution in program accroding to requirement    of the user.

*Types of Control Statments*
----------------------------------
There are two types of control statements present javascript.
1. Condtional Statement.
----------------------------
 a) simple if
 b) if-else
 c) else if ladder
 d) switch statements

2. Unconditional Statements
-----------------------------
 a) break
 b) continue
 c) return
- These conditional statements is also known as selection statements.
- The conditional statements use the boolean expression for condition test.

*1. simple if:*
------------------
- An if statement in javascript is the simplest decision making statement that allows to specify an alternative path of execution in program.
- It is used to change the execution flow of the program.
- This statements executes a block of statement when the condition is true, and skips them if the condition become false.
- The if stateement is also called as single selection statement.

*Syntax of If Statements*
------------------------------
if(Test_condition)
{
 Logic
}

*Examples*
---------------------
html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple If </title>
</head>
<body>

    <script type="text/javascript" src="../conditional_statements/condittional_js/simple_if.js"></script>
</body>
</html>

js code:
------------
let radius, pi, area;

radius = 5;
pi = 3.14;
if(radius > 0)
{
    area = pi * radius * radius;
    document.write("The area of the circle is " + area);

}

*Types of selection Statements*
--------------------------------------
1. One way if statements
2. Two way if-else statements
3. Nested if statements
4. Multi way if else statements
5. Switch Statements


*Two way if-else statements*:
--------------------------------------
1. If else in JavaScript is a two-way conditional statement or double selection statement. It tests a condition and executes one of two sets of code, depending

2. In other words, if else statement executes the first statement if the condition is true. Otherwise, it executes the second statement if the condition is fal

3. A one-way if statement executes a statement if and only if the specified condition is true. If the condition is false, nothing will be done.

4. But, assume we want to take alternative action when the specified condition is false. In this case, we will use a two-way if-else statement.
```

```
*Syntax of If else statement in JavaScript*:
-------------------------------------------
if(expression or condition)
{
    statement to be executed if the condition is true.
}
else {
     statement to be executed if the condition is false.
}


*Multi way if else statements*:
-----------------------------------
If-else if ladder in JavaScript is a multi-way decision structure that we can use to decide among three or more actions.
The general syntax for if-else if ladder statements in JavaScript are as follows:

if(condition-1)
       statement1; // It will execute if condition-1 is true.
else if(condition-2)
       statement2; // It will execute if condition-2 is true.
else if(condition-3)
       statement3;
...
...
else if(condition-n)
      statementn; // It will execute if condition-n is true.
else
    statement; // It will execute if none of the condition is true.

*How if-else if Ladder works in JavaScript*:
-----------------------------------------------
*If-else if ladder works in the following steps that are, as follows:*

1. The first specified condition-1 evaluates to true. If the condition is true, statement1 inside if block will execute and the rest part of else if ladder wil

2. If the specified condition-1 is false, the second if condition-2 evaluates to true. The second condition-2 is true, statement2 will execute, and the rest pa

3. If the second condition-2 is false, the third condition-3 and the rest of the conditions (if required) will evaluate (or test) until a condition meets or al

4. If all the conditions are false, the final else block (i.e., statement) associated with the top if statement will execute.

5. The statement associated with final else block acts as a default condition. That is, if all of the above conditional tests fail, the statement associated wi

*If there is no final else block and none of the conditions is true, in this case, no action will take place.*



*Switch Statements*:
-------------------------
A switch statement in JavaScript is a multi-way decision statement that executes one statement from multiple conditions.
In other words, a switch statement executes statements based on the value of a variable or an expression, against a list of case values.If a match found, a blo

The switch statement is basically an enhanced version if-else if ladder statement. It is more convenient to use when there are multiple conditions or expressio

*Syntax of Switch Statement in JavaScript*:
-------------------------------------------
switch(expression)
{
    case value-1:
        // statement sequence
        break;
   case value-2:
      // statement sequence
      break;
   .
   .
   .
  case value-n:
       // statement sequence
       break;
  default:
        // default statement sequence
}
statement-x;

a) In the above switch syntax, an expression must be of integer, string, boolean, or character type. The expression must be type compatible with each of the va

b) value-1, value-2, value-3, … are constants or literals. These constants have called case labels (or case clauses).

c) A switch can have multiple case clauses depending on requirements and must be unique within a switch statement. The case labels must end with a colon (:).

d) The statement sequence is a list of statements and may contain zero or more statements.

e) After each case, a break statement is necessary inside the switch block to come out from the switch block. Break statement terminates the loop.

f) The default statement is optional and execute when none of the previous cases matched. In other words, the default clause handles the case when no match of


*How Switch statement works?*
----------------------------------
JavaScript switch statement works as follows:

a) When the switch statement execute, the value of the expression has successively compared with each case value like value-1, value2-, …

b) If a match found, the statement sequence following that case label executed. For example, the value of expression is 1, then statements associated with case

c) If the value of expression does not equal to value-1, value-2, …., then none of the statement sequences will execute.

d) In this case, the default clause will execute, and then default statements will execute. However, the default statement is optional. If no case has matched

e) The break statement has used inside the switch block to terminate statement sequences. It is optional. It immediately ends the switch statement.

f) When a break statement executes inside the switch block, the control of execution transfers to the next statement-x after the switch statement (skipping all

g) There is no need to use break statement in the default case, because default is the last case and after it, the switch statement will terminate.
```

```
*Looping statements*
---------------
Q. Why we use the loop?
-------------------------
Example:
-----------
Suppose there is no loop concept, and i will tell you print the number from 1 to 100000. Then you will write the code as below,

console.log(1);
console.log(2);
console.log(3);
.
.
.
.
.
console.log(10);

To overcome the above problem we use the concept called looping.
In javascript, loops are control structure that allow you to repeataly execute a block of code as long as a certain condition
is met.

- In javascript basically there are three types of loop presents,
 a) for loop
 b) while loop
 c) do-while


a) for loop(entry control):
----------------
A for loop  is typically used when you know how many times you want to iterate. It allows you to execute a block of code repeatedly for a specified number of t

Syntax:
---------------
for(initilization; condition checking; incr/decr)
{
 Logic
}


Example:
--------------
function getNumber() {
  for (let i = 1; i <= 5; i++) {
    console.log(i);
  }
}
getNumber();


2. while loop(entry control loop):
---------------------
- The while loop is used when the number of iterations is not known, and it continues executing the block of code as long as the
specified condition is met.

Syntax
---------
while(condition)
{
 logic
}


Example:
-----------
 var i = 1
 while (i <= 10)
 {
   if(i % 2 == 0)
   {
   console.log(i);
   }
   i++;
 }


3. do-while loop(Exit control loop):
---------------------
- Similar to while loop, the do-while loop execute the block of code at least once before checking the condition.

syntax:
--------
do
{
 logics
}
while(condition);

Example:
------------
let i = 1;
do {
  console.log(i);
  i++;

} while (i <= 10);

*Create a simple login based application using loop and conditional statements*
-----------------------------------------------------
*html code*
----------------
<body>
    <script type="text/javascript" src="js/login_attempt.js"></script>
```

```
</body>

*js code:*
--------------
// first i need to add user data then i create the array
const userData =[
    {username:"user1",password:"pass1"},
    {username:"user2",password:"pass2"},
    {username:"user3",password:"pass3"},
]

let loggedIn = false;
let attempts = 3;


while(attempts > 0)
{
    const username =prompt("Enter username");
    const password = prompt("Enter password");

    const user = userData.find((uData) =>uData.username===username && uData.password===password);

    if(user)
    {
        loggedIn = true;
        alert("Login successful");
        break;
    }
    else{
        attempts--;
        alert(`Invalid credentials. You have ${attempts} attempts left.`);
    }
}
if(!loggedIn){
    alert("Maximum number of attempts reached. Please try again later.");

}


*Exception handling*
--------------------------
Q. What is mean by exception?
------------------
Def : exception is an abnormal issue, that are occured due to the mistake of developer. Generally exception occure at runtime.

- When executing javascript code, different errors occurs. Error can be coding error made by the programmer due to wrong     input.
- When exception occur the execution of the program stop, that means it distrub the normal flow of the program.
- TO handle the exception javascript uses three main keyword as below,
 a) try
 b) catch
 c) finally

*Try:*
----------
We can keep our logic within the try block. Suppose in future there will be any error occur then this try block create the object of exception class.

*catch:*
--------
When exception occur in try block, the try block will throw the object of the exception class to the catch block, catch block cathes the exception and rest of

*finally:*
--------


*Error object properties*
-----------------------
It has two parameter internally

1. name ---> sets or return an error name
2. message ---> Sets or return error message in the form string.


*Error Name values:*
------------------------------
Six different values can be returned by the error name property

1.  EvalError : In Latest js it is not present.
2. RangeError : A number "out of range" has occured.
3. ReferenceError : An illegal reference occured.
4. SyntaxError :
5. TypeError:
6. URIError:


*RangeError:*
---------------------
Example:
----------
let num = 1;

try{
    num.toPrecision(500);
}
catch(err){
    //document.getElementById("res").innerHTML = err.name;
    console.log(err.name);
    console.error(err.message);
}

console.log("welcome");



*ReferenceError:*
--------------------------
A ReferenceError is thrown if you use a variable that has not been declared.
```

```
Example:
-------------
let x = 5;
try
{
    x = y + 1;
}
catch (e)
{
    console.log(e.name);
    console.log(e.message);
}
console.log(x); // Outputs: ReferenceError: y is not defined



*SyntaxError*
------------------
A SyntaxError is thrown if you try to evaluvate code with syntax error.

Example:
------------
let x = 5;
try
{
    if (x == 6)
    {
        console.log("x is 6");
    ------------------------------------> curly bracket missing
}
catch (err) {
    console.log(err.name);
}
console.log(x);




*TypeError:*
----------------------
Example:
-------------
let num = 1;
try {
num.toUpperCase();   //-----> You cannot convert a number into uppercase
} catch(err){
    console.log("Error: Cannot convert a number to a string");
    console.log(err.name);
}

*URIError:(Uniform Resource Identifier)*
-----------------
URIError is thrown if you use illegal characters in URI Function.

Example:
----------------
try{
    decodeURI('https://www.google.com%/'); // in this url i am passing the character %
}
catch(err){

        console.log(err.name);
        console.error(err.message);

}


*6. Functions in javascript*
------------------------------------

Q . What is function?
------------------------
Def : A function is a named sequence of statements that performs a specific operation. Functions are used to break down program into smaller, reusable pieces.

*Syntax*
-------------
function function_name(param1, param2......)
{
 statements
 return;
}

- In above syntax a function is defined using the keyword called 'function' followed by function_name.
- In above syntax we also pass the parameters if required.
- If the function return a value by using the return keyword.

*The basic idea of function is to reduce the number of repeated code blocks and executing a code block whenever needed.*

Example:
--------------
function add(a,b)
{

    let sum = a + b;
    return sum;
}

console.log(add(1,2));



*Functions Type*
--------------------------------
*1. classic declaration of function/named function:*
---------------------------------------
```

```
Example:
-------------
function sayHello()
{
    console.log("Hello, World!");
}

//calling function

sayHello();

*2. Functions with parameters*
-----------------------------------------
We can also pass parameters to the function for performaing some calculations, the following is the example,

function add(n1,n2)
{
    let result = n1 + n2;
    return result;
}

//calling function

console.log(add(10,14));


*3. Function expression:*
-------------------------------------
A function expression is way in which we can assign one function result to variable.
Example:
----------------
function doSquare(number)
{
    return number * number;
}

let output = doSquare(10); //function expression
console.log(output);


*4. Anonymous Function:*
----------------------------------------
Anonymous function is function which has no name.

Syntax:
-------------------
function(parameters)
{
 statements
}

Example:
----------------
<body>

    <button  id="mybutton">Click Me..!</button>
    <script src="../js/anonymous.js" type="text/javascript"></script>
</body>

JS Code:
------------------
document.getElementById("mybutton").onclick = function(){

    alert("Welcome to the Javascript");
}


*5. Arrow Function/Fat Function*
---------------------------------------------
Arrow function are a concise way of writting function expression. They use => syntax and have shorter
syntax then any other function type.

const add = (a,b)=>{

    return a+b;
}

console.log(add(1,2));

OR
//anonymous function
const add = function(a,b){

    return a+b;
}

console.log(add(1,2));


OR
//implicit return function
const add =(a,b)=> a+b;
console.log(add(1,2));


6. IIFE(Immediatly invoked function expression):
-------------------------------------------------------------------
The IIFE are functions that are executed immediately upon defination. This function call only one time.

syntax
------------------
(function(){

 statments

})();
```

```
Example:
-------------------
((a,b)=>{

    let res = a+b;
    console.log(`The addition is ${res}`);

})(2,2);


*What is mean by callback function*
-------------------------------------------------------
   Def : A callback function is a function that is passed as an argument to another function and executed after the completion of that function.

- To perform asynchronous operation.

- A callback function is not executed immediately insted it is called back at a certain point, either after     another function complete its task or when an e

Example:
-------------------
function sayHello(name,callback)
{
    console.log("Hello, " + name);
    callback(); // invoking the callback function after the message is printed.

}
//callback function

function showMessage(){

    console.log("This is a callback function.");
}
sayHello("John",showMessage);


*Real time scenario about callback function*
---------------------------------------------------------------
- A real life analogy for a callback function can be understood by using an example of restaurant and       how a customer places an order.

*Scenario*
-----------------------
1. Customer(main function) :
---------------------------------------
The customer enters the restourant, places order for food and then sit down at a table.

2. Chef(callback function):
-------------------------------------
The chef in the kitchen prepare the food.

3. Waiter(callback execution):
----------------------------------------
When the food is ready the waiter brings the food to the customer.

*Example*
------------------
function placeOrder(order, callback)
{
    console.log("Customer: I would like to order " + order);
    console.log("Chef is preparing the order");

    setTimeout(()=>{
        console.log(order + " is ready");
        callback(order); // invoking the callback function after the order is ready

    },8000);
}

function serveFood(order){

    console.log("Waiter : here is your " + order +" enjoy your food");
}

placeOrder("pizaa", serveFood);

-----------------------------------------------------------------------------------------------------------------------


Arrays in javascript
---------------------------
- An array can be used to store the multiple values which has similar data types in single variable.
- In javascript the array can be used to store the different types of data into a single variable.
- In javascript the array is automatically increses and decreases when the element added or deleted from   array.
- The element which are present in array can be accessed by using index.
- We have two ways to create the array in javascript.
 a) By using array literal
 b) By using Array constructor.


*a) By using array literal:*
---------------------------------------------
Creating an array using array literal invlove by using square bracket and we can initilize the array.

Example:
--------------------

let arr = ["html","css","js","java"]
//console.log(arr);

for(let i = 0; i <= arr.length-1; i++){
    console.log(arr[i]);
}


*b) By using Array constructor:*
---------------------------------------------------
```

```
The array constructor refers to a method of creating array by invoking the array constructor function.

Example:
------------
let arr = new Array("html","css","js","java");
//console.log(arr);

for(let i = 0; i <= arr.length-1; i++){
       console.log(arr[i]);
}


*Basic operations on array*
------------------------------------------------------
1. Acceesing the element of array:
---------------------------------------------
We can access the element of array as below,
1. by using index
2. by using for loop
3. by using console.log(arrayname)


2. Accessing the first element of the array:
---------------------------------------------------
We can access the first element from array using index as 0.

Example:
---------
let arr = new Array("html","css","js","java");
console.log(arr[0]);

3. Accessing the last element of the array:
------------------------------------------------------------
We can access the last element from using array.length-1

Example:
-----------------
let lastElement = arr[arr.length-1];
console.log("The last element is = ",lastElement);

4. Modifying array element:
---------------------------------------
We can easily modify the existing element from array.

Example:
-----------------
let arr = new Array("html","css","js","java");
arr[3] = "Bootstrap";
console.log(arr);

5. Adding the element into array:
--------------------------------------------------
Elements can be added into array by using two methods.

a) push : The push method adds the element at the end.

Example:
---------------
let arr = new Array("html","css","js","java");

arr.push("nodejs");

for(let i = 0 ; i<= arr.length-1; i++)
{
    console.log(arr[i]);
}


b) unshift : The unshift adds the element at first location.
Example:
-----------------
let arr = new Array("html","css","js","java");

arr.unshift("nodejs");

// for(let i = 0 ; i<= arr.length-1; i++)
// {
//     console.log(arr[i]);
// }

console.log(arr);


6. Remove the element from array:
------------------------------------------------------
We can remove the element from array by using two methods.

a) pop : the pop method remove the element which are present at last.
b) shift : the shift method remove the element from first index location.

Example:
-------------------

let arr = new Array("html","css","js","java");
let poppedElement = arr.pop();
let shiftedElement = arr.shift();
console.log(arr);
console.log("The popped element is = ",poppedElement);
console.log("The shifted element is = ",shiftedElement);

c) splice: We can also remove the perticular element from array
        The syntax of splice is
        splice(start_index, deletecount)
We can also add new element at specified position by using splice.

Example1:
```

```
--------------------

let arr = new Array("html","css","js","java");
arr.splice(1,0,"pune","mumbai");
console.log(arr);

In above example we can add some cities at perticular location

Example2:
-------------------
let arr = new Array("html","css","js","java");
arr.splice(1,2);
console.log(arr);

In above example we remove two element which are starting from index number 1.


7. Concat:
-------------------------
The concat method can be used for to add more then one into single array.

Example:
-----------------
const arr1 = [1,2,3,4];
const arr2 = [5,6,7,8];

let newArray = arr1.concat(arr2);
console.log(newArray);


8. every:
------------------
The every() method checks if all the array elements pass the given test condition.

Example:
--------------
const personAge = [23,34,55,44,43,12]

function checkAdult(age)
{
    return age >= 18;
}

const checResult = personAge.every(checkAdult)
console.log(checResult);

OR

const number = [2,4,6,8,10,5]
function checkEvenNumber(num){

    return num % 2 == 0;
}

let checkResult = number.every(checkEvenNumber);
console.log(checkResult);


*9. find():*
----------------------
The find() method returns the value of first array element that satisfy the given test condition.

Example:
----------------
const number = [7,4,8,11]
function isEven(number)
{
    return number % 2 == 0;
}

let checkEven = number.find(isEven);
console.log(checkEven);


OR

const number = [100,2,4,6,7];
function isEven(number)
{
    return number % 2 == 0;
}

let checkEven = number.find(isEven)
console.log(checkEven);


let firstOdd = number.find((element)=> element % 2 == 1);
console.log(firstOdd);

OR

const personData = [

    {name:"Ajay", city:"Pune", age:19},
    {name:"Sunil", city:"Nashik", age:15},
    {name:"Sanjay", city:"Mumbai", age:16}

];

let checkAdultData = personData.find((ele)=> ele.age >= 18)
console.log(checkAdultData);



*10. filter():*
-------------------------
```

```
The filter() method returns a new array with all elements that pass the given test condition.

Example
----------------
const personAge = [23,34,55,44,43,12];
let checkResult = personAge.filter(

    (ele)=> ele % 2 == 0
)

console.log(checkResult);


OR
-----
const myarray = [1800, 1500, null,2200, "java","3500"];

let checkPrice = myarray.filter(

    (price)=> price >=2000 && !Number.isNaN(price)
)
console.log(checkPrice);

*11. includes():*
-----------------------------
This method can be used to check perticular element present in array or not. If it is present then return true otherwise false.

Example:
-----------------
let language = ["Javascript","python","java","Ruby","html","css"];
let check = language.includes("jav");
console.log(check);


*12. some():*
--------------------------
This method tests weather any of the array element pass the given test condition.

Example:
---------------
function isEven(num)
{
    return num % 2 == 0;
}

let myArry = [21,5,7,9,6];
console.log(myArry.some(isEven));

*13. reduce():*
-----------------------------
The reduce() method execute a reducer function on each element of the array and return single output value.

Example:
---------------------
let myarray = ["javascript","is","easy"];

function combineWords(accumlator, currentValue)
{
    // console.log(accumlator);
    return accumlator +" "+currentValue;

}

let finalResult = myarray.reduce(combineWords);
console.log(finalResult);

OR

let numbers = [1,2,3,4,5,6,7,8]
let result = numbers.reduce(
    (accumlator,currentValue) => accumlator + currentValue
);

console.log(result);

*14. map():*
---------------------------
The map() method create a new array with results of calling a function for every array element.

Example:
--------------------
let number = [1,2,3,4,5,6]
function doSquare(num)
{
    return num * num;
}

let finalResult = number.map(doSquare);
console.log(finalResult);

*15. isArray():*
-----------------------------
This method check if it is an array or not.

Example
------------------
let number = [1,2,3,4,5,6]
let number1 = "javascript";
console.log(Array.isArray(number1));
console.log(Array.isArray(number));

*16. reverse():*
-------------------------------
This method returns array in reverse order.

Example:
```

```
------------------
let number = [1,2,3,4,5,6]
console.log(number.reverse());

*17. sort():*
-----------------------
The sort() method sort the items of an array in a specific order(asc,desc).

Example:
--------------------
let city = ["pune","mumbai","nashik","Amravati"];

let sortedArray =  city.sort()
console.log(sortedArray);

OR

let city = [1,4,5,2,14,13];

let sortedArray =  city.sort((a,b)=>a-b);
console.log(sortedArray);

function sortNum(a,b)
{
    return b - a
}
city.sort(sortNum);
console.log(city);

-----------------------------------------------------------------------------------------------------------------------------------

*Strings in Javascript*
---------------------------------
Q. What is string?
-----------------------------
Character can be used for storing the single value in single quote.
Example:
-------------
char ch = '1';

- But in javascript there is no concept char data type. If you store the value in a single quote then it        considered as String.

Def : The collection characters is called as String.

OR
Def : The sequence of characters is called String, typically it represent text.

OR
Def : The number of characters comes together and create a meaningful word that is called String.

Example:
------------------
pen, laptop, mobile

- In javascript the string can be declared using the single quote as well as double quote.
- The lenght of the string can be calculated by using length.
- The index of the string starting from 0th index.
- We can combine two or more then two string by using concat.
- We can also get the substring from string.

Q. How we can declare the string?
-----------------------------------------------
1. By using classic declaration
----------------------------------------
let mystr = "Javascript is easy";
console.log(mystr);


2. By using string constructor
-----------------------------------------
let mystr = new String("This is javascript");
console.log(mystr);

3. Template literal:
-----------------------------
let mystr = 'This is javascript';
let newStr = `String created ${mystr}`;
console.log(newStr);

4. Multiline declarations:
-------------------------------------------
let myStr = `Javascript is
             easy
            to
            understand`;
console.log(myStr);



*What are the basic operations performed on String*
---------------------------------------------------------------------------
1. length:
-------------------
You can easily find the length of the string by using length() method.

Example:
----------------
let myStr = "Javascript is easy"
console.log(myStr.length);

2. Combine the string(Concat):
---------------------------------------------
You combine two or more then two string easily.

Example:
----------------
```

```
let mystr1 = "javascript";
let mystr2 = "developer";

console.log(mystr1+" "+mystr2);

3. Breaking long string:
---------------------------------
let myStr = "'The javascript is easy to'\ learn and understands";
console.log(myStr);
document.write(myStr);

4. Finding the substring:
--------------------------------------
You can find the portion of the string by using the substring() method. The substring() requires two argument start number and second end number.

Example:
-----------------
const myStr = "The javascript is easy to learn and understands";
let substr = myStr.substring(4,14);
console.log(substr);

hello javascript (4,10)  oa

5. Convert the string into Uppercase and lowercase:
--------------------------------------------------------------------
We can use two methods
1. toUpperCase()
2. toLowerCase()

Example:
--------------------
const myStr = "The javascript is easy to learn and understands";
console.log(myStr.toUpperCase());
console.log(myStr.toLowerCase());


6. Find the index of the string:
----------------------------------------
Find the index of the string by using indexOf() method. If the string not found then it will return -1.

Example:
----------------
const myStr = "The javascript is easy to learn and understands";
console.log(myStr.indexOf('javascript'));

7. Replace/ReplaceAll:
-------------------------
This method replace the first matching string with new string.
replaceAll() method replace the string which is present at every line.

Example:
-----------------
const myStr = "The javascript is easy to learn and javascript understands";
let newStr = myStr.replace("javascript","python");
console.log(newStr);


8. trim():
-----------------------
- It removes the space from right side and from left side.

Example:
------------------
const myStr = "    The javascript is easy to learn and javascript understands    ";
console.log(myStr.trim());

9. lastIndexOf():
-------------------------
The lastIndexOf() method returns the last index of occurance of a given substring in the string.

Example:
-------------------
const myStr = "The javascript is easy to learn and javascript understands.";
var substr = "javascript12";
var result = myStr.lastIndexOf(substr);
console.log(result);


10. chatAt():
---------------------
The charAt() method returns the character at the specified index in string. The charAt() method olny consider integer number.

Example:
-----------------
const myStr = "The javascript is easy to learn and javascript understands.";
console.log(myStr.charAt(1));

OR

By using float value, but it will considered integer value.

const myStr = "The javascript is easy to learn and javascript understands.";
console.log(myStr.charAt(1.9));

11. startsWith():
------------------------------
The startsWith() method returns true if the string begins with specified value otherwise it will return false.

Example:
----------------
const myStr = "The javascript is easy to learn and javascript understands.";
console.log(myStr.startsWith("The"));



12. endsWith():
```

```
-------------------------------
The endsWith() method returns true if the string ends with specified value otherwise it will return false.

Example:
-------------------
const myStr = "The javascript is easy to learn and javascript understands.pdf";
console.log(myStr.endsWith(".pdf"));

13. match():
------------------------
The match() method returns the result of matching a string against a regular expression.

Example:
-----------------
matchAll():
----------------------
const myStr = "The ajavascript is easy to learn and ajavascript understands.pdf";

let exp = /javascript/g;
let result = myStr.matchAll(exp);
for(r of result)
{
    console.log(r);
}

OR
match() method
------------------------
const myStr = "The ajavascript is easy to learn and ajavascript understands.pdf";

let exp = /javascript/;
let result = myStr.match(exp);
console.log(result);

charCodeAt():
------------------------
Example
------------
const myStr = "The ajavascript is easy to learn and ajavascript understands.pdf";
console.log(myStr.charCodeAt(1));

-----------------------------------------------------------------------------------------------------------------------
*Event handling*
---------------------------
- The change in the state of object is called as event.
- In html, there are various events which represent that some activity is performed by user.
- When javascript code is included with html, javascript reacts over these events and allow the                  execution.
- This process of reacting over these element are called event handling.

Example:
-------------------
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
    <style>
        .custom_btn
        {
            display: block;
            margin:auto;
            margin-top: 200px;
        }
    </style>

</head>
<body>
    <script type="text/javascript">
        function changeBackColor(){
           document.body.style.backgroundColor='#0C090A';
        }
    </script>
    <form>
        <button type="button" id="btn" class="btn btn-primary custom_btn" onclick="changeBackColor()">
            <i class="bi bi-coin"></i>
            Change Color</button>
    </form>

</body>
</html>


OR

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
    <style>
        .custom_btn
        {
            display: block;
            margin:auto;
            margin-top: 200px;
        }
    </style>
```

```
</head>
<body>

    <form>
        <button type="button" id="btn" class="btn btn-primary custom_btn">
            <i class="bi bi-coin"></i>
            Change Color</button>
    </form>
    <script type="text/javascript">
        {
            document.getElementById('btn').addEventListener('mouseover',function(){

                document.body.style.backgroundColor = "#0C090A";
                document.getElementById('btn').style.borderRadius="50%";

            });

            document.getElementById('btn').addEventListener('mouseout',function(){

            document.body.style.backgroundColor = "#fff";
            document.getElementById('btn').style.borderRadius="0";

});
        }
    </script>
</body>
</html>
-------------------------------------------------------------------------------------------------------------------
DOM(document object model/ window)
-----------------------------------------------
how we can get and set the value of attribute?
----------------------------------------------------------
a) getAttribute:
---------------------
we can get the attribute value using following methods.
1. innerText
2. innerHTML
3. getAttribute(attributeName)
4. attributes ---> if we get the name of the attribute then we can use name property as well as we can get the value of attribute then we can use value propert


b) setAttribute:
--------------------
1. innerText
2. innerHTML
3. setAttribute
4. removeAttribute


DOM CSS Styling:
--------------------------------
There are 3 ways to style the element.
1. style
2. className
3. classList

Q. What is the difference between className and classList?
Q. what are the different methods present in classList?

1. add() : it will add new class to document.
2. remove() : it will remove the class which you want.
3. toggle() : Toggle can be used for to change one class into another class and v/s.
4. contains(): contains method can be used to check conditional checking. It always returns true or false value.
5. item(index) : To select perticular class by using the index number.
6. length() : Total number of classes present document.



What is mean by querySelector() and querySelectorAll()?
querySelector -> It will select the first match.
querySelectorAll --> It will select all the match which is present in document.

Timing functions in javascript
---------------------------------------
1. setInterval : It executes every time which time you passed.
2. setTimeout : It executes only one time.
3. clearInterval
4. clearTimeout


DOM Travsal Methods
--------------------------
1. parentElement:
-------------------------
Returns the parent element of the current node. The parent node as an HTMLElement, or null if the parent is not an element or the current node is not in the DO

2. parentNode:
-----------------------
Returns the parent node of the current node (can be an element or another type of node like a document node).The parent node as a Node, or null if the current

3. children:
---------------------
Returns a live HTMLCollection of the child elements (only elements) of the current element.

4. childNodes:
----------------------
Returns a live NodeList of all child nodes, including elements, text, and comment nodes.

5. firstChild:
---------------------
Returns the first child node (can be an element, text, or comment) of the current node.The first child node as a Node, or null if there are no child nodes.

6. lastChild:
---------------------
Returns the last child node (can be an element, text, or comment) of the current node.The last child node as a Node, or null if there are no child nodes.
```

```
7. firstElementChild:
--------------------------------
Returns the first child element (only elements) of the current element. The first child as an HTMLElement, or null if there are no child elements.

8. lastElementChild:
-------------------------------
Returns the last child element (only elements) of the current element.The last child as an HTMLElement, or null if there are no child elements.

9. nextSibling:
------------------------
 Returns the next sibling node (can be an element, text, or comment) of the current node.

10. previousSibling
-------------------------
Returns the previous sibling node (can be an element, text, or comment) of the current node.

11. nextElementSibling:
----------------------------------
Returns the next sibling element (only elements) of the current element.The next sibling as an HTMLElement, or null if there is no next sibling element.

12. previousElementSibling:
------------------------------------
Returns the previous sibling element (only elements) of the current element. The previous sibling as an HTMLElement, or null if there is no previous sibling el

*CreateElement, createTextNode,createComment:*
---------------------------------------------------------------
1. createElement :
----------------------
The createElement method is part of the DOM API in JavaScript. It is used to create a new HTML element dynamically. This element is not yet part of the documen

Syntax:
-----------
document.createElement(tagName);

Parameters:
tagName: A string representing the name of the HTML tag to create (e.g., div, p, span, img).

Return Value:
Returns a reference to the newly created element (of type HTMLElement).

Example:
---------------
let newDiv = document.createElement('div');
newDiv.id = "myDiv";
newDiv.className = "container";
document.body.appendChild(newDiv); // Appends the new div to the body



2. createTextNode :
----------------------------
The createTextNode method is used to create a new text node. Text nodes represent the actual text content that can be added to an element in the DOM.

Syntax:
-----------
document.createTextNode(data)

Parameters:
data: A string containing the text you want to include in the node.

Return Value:
Returns a Text object representing the text node.

Example:
--------------
let textNode = document.createTextNode("Hello, World!");
let newParagraph = document.createElement('p');
newParagraph.appendChild(textNode); // Adds the text node to the paragraph
document.body.appendChild(newParagraph); // Appends the paragraph to the body


*3. appendChild:*
-----------------
The appendChild method is used to add a node to the end of the list of children of a specified parent node. This method appends only one node at a time.

Syntax:
------------
parentNode.appendChild(childNode)

Parameters:
-----------------
parentNode: The parent node to which you want to append the child.
childNode: The node you want to append to the parent node.

Example:
--------------
let parent = document.getElementById('container');
let child = document.createElement('p');
child.textContent = "This is a new paragraph.";
parent.appendChild(child); // Adds the paragraph to the end of 'container'


*4. insertBefore:*
--------------------------
The insertBefore method is used to insert a node before a specified existing child node of a parent node.

Syntax:
-------------
parentNode.insertBefore(newNode, referenceNode)

Parameters:
------------------
newNode: The node you want to insert.
referenceNode: The child node before which the new node will be inserted. If referenceNode is null, newNode is appended at the end.

Example:
```

```
---------------
let parent = document.getElementById('container');
let newChild = document.createElement('p');
newChild.textContent = "Inserted paragraph.";

let referenceChild = parent.firstChild; // Reference to the first child
parent.insertBefore(newChild, referenceChild); // Inserts the new paragraph before the first child


*5. Prepend*
----------------------
The prepend method is used to insert content at the beginning of the specified parent node. It can take nodes or strings as arguments. When a string is passed,

Syntax:
-----------
parentNode.prepend(...nodesOrStrings)

Parameters:
nodesOrStrings: One or more nodes or strings to be inserted as the first child of the parent node.

Example:
---------------
let parent = document.getElementById('container');
let newChild = document.createElement('p');
newChild.textContent = "Prepended paragraph.";
parent.prepend(newChild); // Inserts the paragraph as the first child

*6. removeChild:*
----------------------------
The removeChild method is part of the DOM API in JavaScript. It is used to remove a child node from a specified parent node in the DOM. The removed node is not

Syntax:
-----------
parentNode.removeChild(childNode)

Example:
---------------
let parent = document.getElementById('container');
let child = document.getElementById('childElement');

// Remove the child element from its parent
parent.removeChild(child);

 Remove All Children Dynamically
 --------------------------------------
 let parent = document.getElementById('container');

// Remove all child nodes of the parent element
while (parent.firstChild) {
    parent.removeChild(parent.firstChild);
}

==========================================================================================

Browser Object Model(BOM)
------------------------------------
Q. What is mean by BOM?
----------------------------------
- The Browser Object Model (BOM) is used to interact with the browser.
- The default object of browser is window means you can call all the functions of window by specifying window or     directly. For example:
   window.alert("hello javascript");  is same as alert("hello javascript");

- You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerW

Q. What is mean by Window Object?
--------------------------------------------
The 'window object in JavaScript represents your web browser's window. Everything you create in JavaScript, like variables and functions, is automatically part

Global variables are like items in the window, and global functions are tools you can use from anywhere. The "document" object, which deals with HTML content,

What are the different properties present in BOM?
-----------------------------------------------------------------------
1. innerHeight
2. innerWidth
3. outerHeight
4. outerWidth

*Window open() and window close()*
---------------------------------------------------------
1. window.open(URL, target, spec) / window.Location.href
------------------------------------------------------------------------
a) URL : which website you need to open. It will contains the address of webpage which you need to open

b) target : _blank, _parent, _top, _self

c) spec : we pass some parameters like, by default when i open the window it has location left, top side.
You can change the location of the window by passing attribute like left, top, height and width


2. window.close(window open variable name):
----------------------------------------------------------------
Example:
-------------
let myWindow;
function openWindow() {
    if(!myWindow || myWindow.closed)
    {
        myWindow = window.open(
            "",
            "",
            "height=200,width=300"
        );
        myWindow.document.write("<p>This from test department</p>");

    }
    else
    {
        alert('Window already open');
        myWindow.focus();
    }

}
function closeWindow() {

    if (myWindow) {
        myWindow.close();
        myWindow = null;
    } else {
        alert("No window to close!");
    }
}

function moveWindow()
{
    myWindow.focus();
    myWindow.moveTo(100,100);
}


*moveTo and moveBy:*
-----------------------------------------
*Syntax*
------------------
1. moveTo(x,y);
2. moveBy(x,y);

moveTo : it moves the window from its relative location. If default value is not present that left and top then it will moves to given values which provided in
moveBy : it moves the window from its absolute location. When we use moveBy actually it will sum default values that is left and top into moveBy values.

Example:
------------------
Note: first open the window then perform the following operation.
function moveWindow()
{
    myWindow.focus();
    myWindow.moveTo(100,100);
}

function moveWindowBy(){
    myWindow.focus();
    myWindow.moveBy(100,100);

}

*resizeTo and resizeBy(x, y):*
-------------------------------------------------
Example:
--------------
Note : First open the window then perform the operation

// resizeTo() and resizeBy():

function callrsizeTo(){
    myWindow.resizeTo(200,200);
    myWindow.focus();
}

function callrsizeBy(){
    myWindow.resizeBy(200,200);
    myWindow.focus();
}

*scrollTo and scrollBy:*
---------------------------------------
Syntax:
-------------------
scrollTo(x,y) and scrollBy(x,y)

In above syntax the value may be negative or positive. When the value is positive then it will scroll forward direction when the value is negative then it will

- scrollBy scroll the page step by step.
- scrollTo : It will move direct to given points, and not goes to forward and backward direction.

*Location object*
---------------------------
Location object only works with server.

Methods:
---------------
1. assign("url") : To open the another page
2. reload() : to refresh page
3. replace("url"): It delete all previous history.

=================================================================================
*localStorage And sessionStorage*
------------------------------------------------
*LocalStorage:*
----------------------
- Javascript localStorage is web storage feature that allows you to store key-value pair value in browser.
- The local storage is unsecure.
- Data persists even after the browser is closed.
- The localstorage store the data upto 5mb per domain.

*Syntax:*
---------------
variable_name = window.localStorage;


*What are the different properties present localStorage and sessionStorage?*
---------------------------------------------------------------------------------------------------------
1. setItem(key, value) : stores the key value pair data.
2. getItem((key) : Returns the value against the key.
3. removeItem(key): It will remove specific data from storage.
4. clear() : delete all the key value data from localStorage.
5. length : it will return total number of data present in localStorage.

Examples:
---------------
check on the git
=============================================================================
Q. What is a Promise in JavaScript?
****************************************************

A Promise in JavaScript is an object that represents the eventual completion (or failure) of an
asynchronous operation and its resulting value.
It allows you to handle asynchronous tasks more effectively, replacing older methods like callbacks.

A promise starts in a pending state. That means the process is not complete.
If the operation is successful, the process ends in a fulfilled state. And, if an error occurs,
the process ends in a rejected state.

For example, when you request data from the server by using a promise, it will be in a pending state.
When the data arrives successfully, it will be in a fulfilled state. If an error occurs, then it will be in
a rejected state.

A Promise can be in one of three states:
--------------------------------------------------------
1. Pending: The initial state, neither fulfilled nor rejected.
2. Fulfilled: The operation completed successfully, and the resulting value is available.
3. Rejected: The operation failed, and a reason for failure is available.

You can handle the resolved (fulfilled) or rejected states using:

.then() for success
.catch() for errors
.finally() for cleanup tasks (executed regardless of the outcome)


Basic Syntax of a Promise:
---------------------------------------
let myPromise = new Promise((resolve, reject) => {
    // Asynchronous operation
    let success = true; // Simulating an operation
    if (success) {
        resolve("Operation successful!");
    } else {
        reject("Operation failed.");
    }
});

// Consuming the Promise
myPromise
    .then(result => {
        console.log(result); // Logs: Operation successful!
    })
    .catch(error => {
        console.error(error); // Logs if rejected
    })
    .finally(() => {
        console.log("Operation completed!"); // Always executed
    });

Example:
--------------------
let checkEven = new Promise((resolve, reject) => {
    let number = 4;
    if (number % 2 === 0) resolve("The number is even!");
    else reject("The number is odd!");
});

checkEven
    .then((message) => console.log(message)) // On success
    .catch((error) => console.error(error)); // On failure

Q. What are the different methods of promises?
-------------------------------------------------------------------
1. Promise.all():
 -----------------------
 Promise.all() is a method that takes an array of Promises and returns a single Promise.
 This Promise:
 a) Resolves when all the input Promises resolve, with an array of their results.
 b) Rejects as soon as one of the input Promises rejects, with the reason for rejection.

Example:
--------------------
const promise1 = Promise.resolve(10);
const promise2 = new Promise((resolve) => setTimeout(() => resolve(20), 1000));
const promise3 = new Promise((resolve) => setTimeout(() => resolve(30), 2000));

Promise.all([promise1, promise2, promise3])
    .then(results => {
        console.log(results); // Output: [10, 20, 30]
    })
    .catch(error => {
        console.error(error); // Will not run here since all promises resolve.
    });

In the above example, Promise.all() waits for all promises to resolve before logging the results.
If any one of the promises rejects, the .catch() block will execute.

Rejection Example:
--------------------------------
const promise1 = Promise.resolve(10);
const promise2 = new Promise((resolve, reject) => setTimeout(() => reject("Error in promise2"), 1000));
const promise3 = new Promise((resolve) => setTimeout(() => resolve(30), 2000));

Promise.all([promise1, promise2, promise3])
    .then(results => {
        console.log(results); // Won't execute due to rejection.
    })
    .catch(error => {
        console.error(error); // Output: Error in promise2
    });


2. Promise.race():
----------------------------------------
Promise.race() is a method that takes an array of Promises and returns a single Promise.
This Promise:

a) Resolves or rejects as soon as any one of the input Promises resolves or rejects.
b) The result or error of the "first settled" Promise is returned.

const promise1 = new Promise((resolve) => setTimeout(() => resolve("Promise 1 won!"), 1000));
const promise2 = new Promise((resolve) => setTimeout(() => resolve("Promise 2 won!"), 2000));

Promise.race([promise1, promise2])
    .then(result => {
        console.log(result); // Output: Promise 1 won!
    })
    .catch(error => {
        console.error(error);
    });
In this example, Promise.race() resolves with the result of the first promise (promise1),
because it resolves faster.


Real-Time Example of Promise.race()
-----------------------------------------------------
Scenario: Fetching data with a timeout
----------------------------------------------------
Imagine you're fetching data from a server, but you want to ensure the response comes within
a specified time limit. If the server is too slow, you'll reject the promise and handle the timeout.

Example
-----------------------
const fetchData = new Promise((resolve) => {
    // Simulate fetching data (takes 2 seconds)
    setTimeout(() => resolve("Data fetched successfully!"), 2000);
});

const timeout = new Promise((_, reject) => {
    // Timeout after 1 second
    setTimeout(() => reject("Request timed out!"), 1000);
});

// Use Promise.race to handle whichever promise settles first
Promise.race([fetchData, timeout])
    .then(result => {
        console.log(result); // If fetchData is faster: "Data fetched successfully!"
    })
    .catch(error => {
        console.error(error); // If timeout is faster: "Request timed out!"
    });
================================================================================
async and await
-------------------------
In JavaScript, async and await are used to handle asynchronous operations in a cleaner, more
readable way compared to traditional Promise chaining. They make asynchronous code look and
behave more like synchronous code.

Key Points:
--------------------
1. async:
------------
Declares a function as asynchronous.
An async function always returns a Promise.

2. await:
----------
Pauses the execution of the async function until the Promise is resolved or rejected.
Can only be used inside an async function.

Example: Fetching data from an API
-------------------------------------------------
// An asynchronous function using async/await
async function fetchData() {
  try {
    console.log("Fetching data...");
    const response = await fetch("https://jsonplaceholder.typicode.com/posts/1");
    const data = await response.json(); // Wait for JSON conversion
    console.log("Data fetched successfully:", data);
  } catch (error) {
    console.error("Error fetching data:", error);
  }
}

fetchData();

Advantages of async/await:
---------------------------------------------

1. Simplifies working with asynchronous code.
2. Easier to read and maintain compared to .then() chaining.
3. Error handling is straightforward using try-catch.

Limitations:
-----------------------

1. Requires modern browsers or environments (Node.js 7.6+).
2. Must always be used with Promises or async functions.

Improved Readability and Maintainability
---------------------------------------------------------
```

```
1. async/await makes asynchronous code look and behave like synchronous code, reducing complexity.
2. This improves readability and makes the code easier to follow and maintain.

Example
-----------------
// With async/await

async function fetchData() {
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();
  console.log(data);
}


Sequential Execution of Asynchronous Code
-----------------------------------------------------------------
1. await ensures that asynchronous operations are executed in order, when necessary.
2. This is particularly useful when the result of one operation depends on the previous one.

Example
------------------

async function processData() {
  const user = await getUser(); // Waits for user data
  const orders = await getOrders(user.id); // Waits for orders based on user ID
  console.log(orders);
}

Better Error Handling with try-catch
-------------------------------------------------
Errors can be caught using try-catch blocks, making error handling more intuitive compared to
chaining .catch() with Promises.

Example:
-------------------
async function fetchData() {
  try {
    const response = await fetch('https://api.example.com/data');
    if (!response.ok) throw new Error('Network error');
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('Error:', error);
  }
}
```