JavaScript 5, 6 & 7

Fundamentals

Introduction to JavaScript

- JavaScript is a programming language, which is used to create functionality in the web page. Functionality means, “receiving inputs from the user and providing output to the user”.
- It can perform tasks such as calculations, decision making, repetitive tasks, dynamically displaying the output, reading inputs from the user dynamically, updating content on the web page based on the inputs, interacting with server, validations etc.
- It’s operators and control statements are similar to “C” language.
- JavaScript is client-side (browser-side) language. That means it executes on the browser. It can also be used in server by using NodeJS.
- JavaScript is a case sensitive language.
- JavaScript is “interpreter-based” language. That means the code will be converted into machine language, line-by-line.
- JavaScript was developed by “Netscape Corporation” in 1996.
- JavaScript is the implementation of "EcmaScript". "EcmaScript" is the specification of "JavaScript".
- "EcmaScript" is designed by "Ecma International".

Syntax of JavaScript

```
<script type="text/javascript">
    Javascript code here
</script>

    -- or --

<script>
    Javascript code here
</script>
```

- **Note:** You can write the <script> tag either in <head> tag or <body> tag also; however, writing <script> tag at the end of <body> tag is a best practice.
- The type="text/javascript" attribute specifies that you are using javascript language; It is optional.

console.log()

- The console.log() statement is used to display value in the browser console.
- To see console, first run the program in the browser and press "F12" or right click and choose "Inspect Element" – "Console" option in the browser.

Syntax: console.log(value);

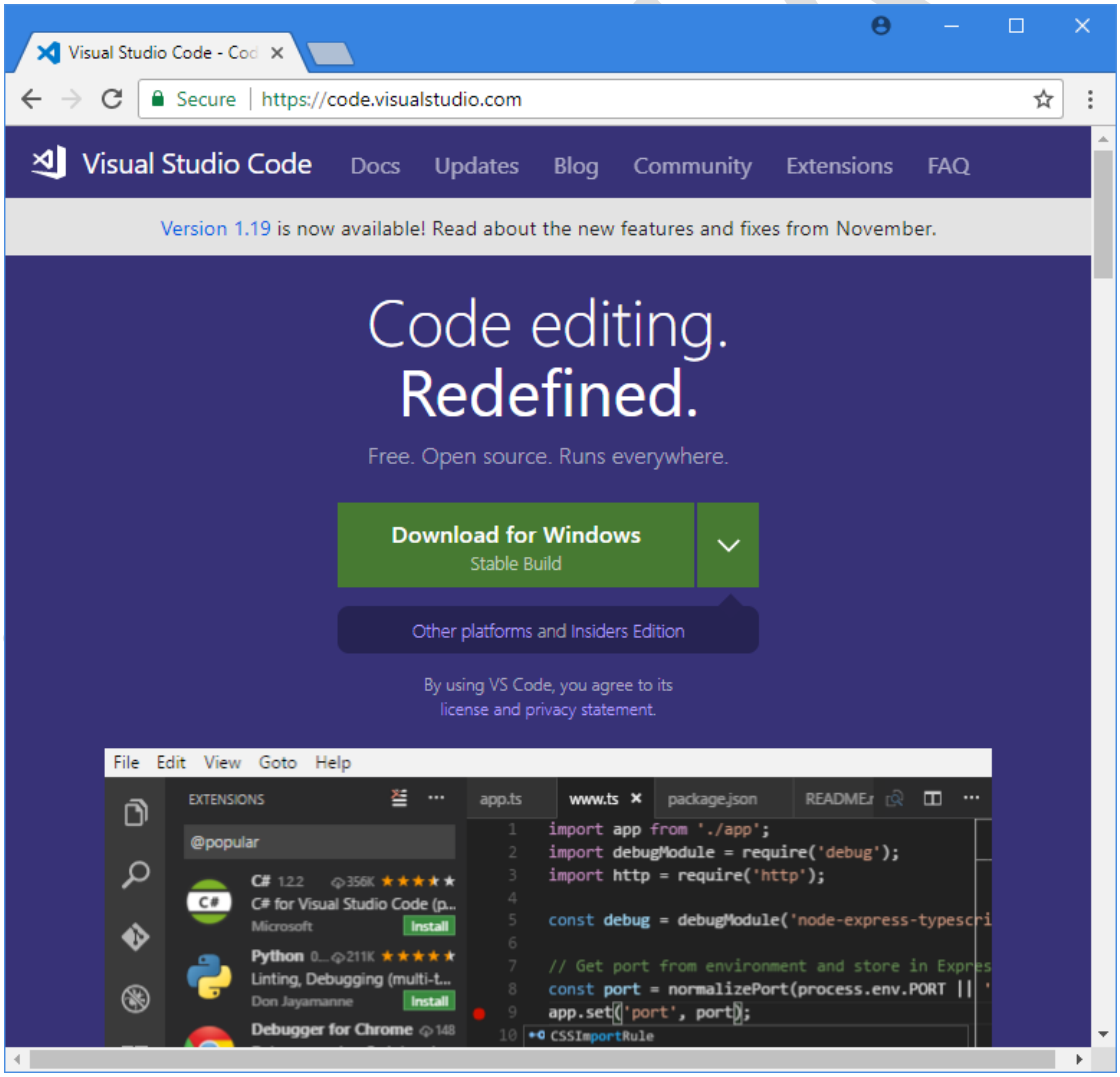
Steps to Prepare First Example in JavaScript

1. Installing Visual Studio Code
2. Creating JavaScript Program
3. Executing JavaScript Program

1. Installing Visual Studio Code

"Visual Studio Code" is the recommended editor for html, css, javascript and many other languages / frameworks.

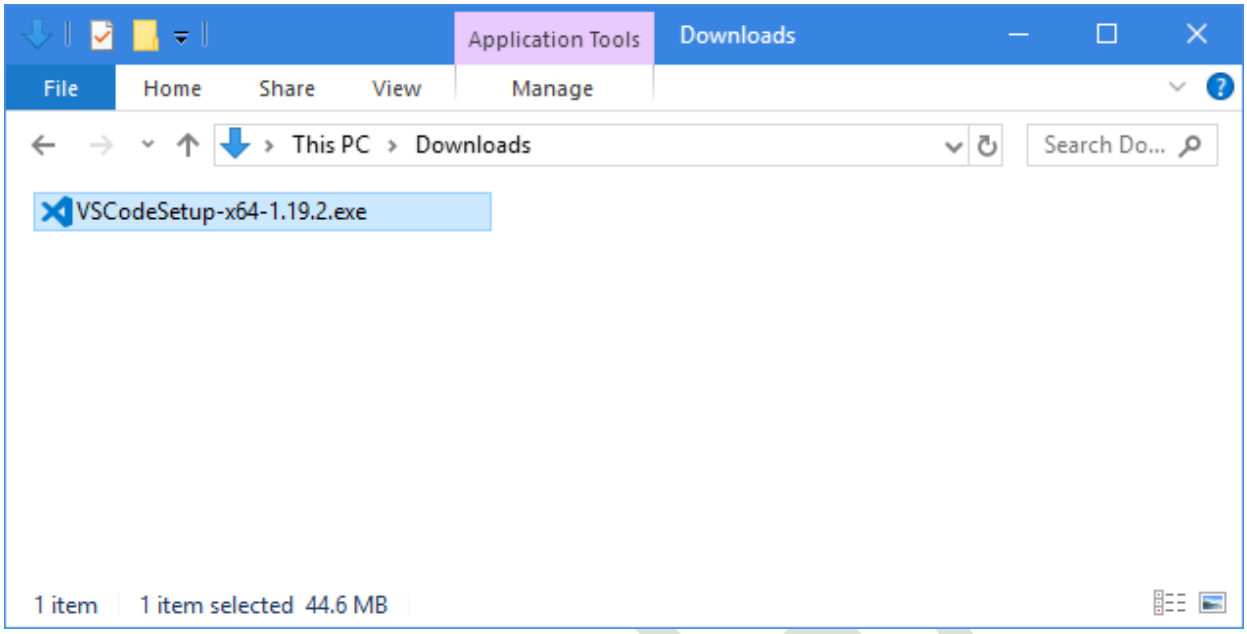
Go to https://code.visualstudio.com



Click on "Download for Windows".

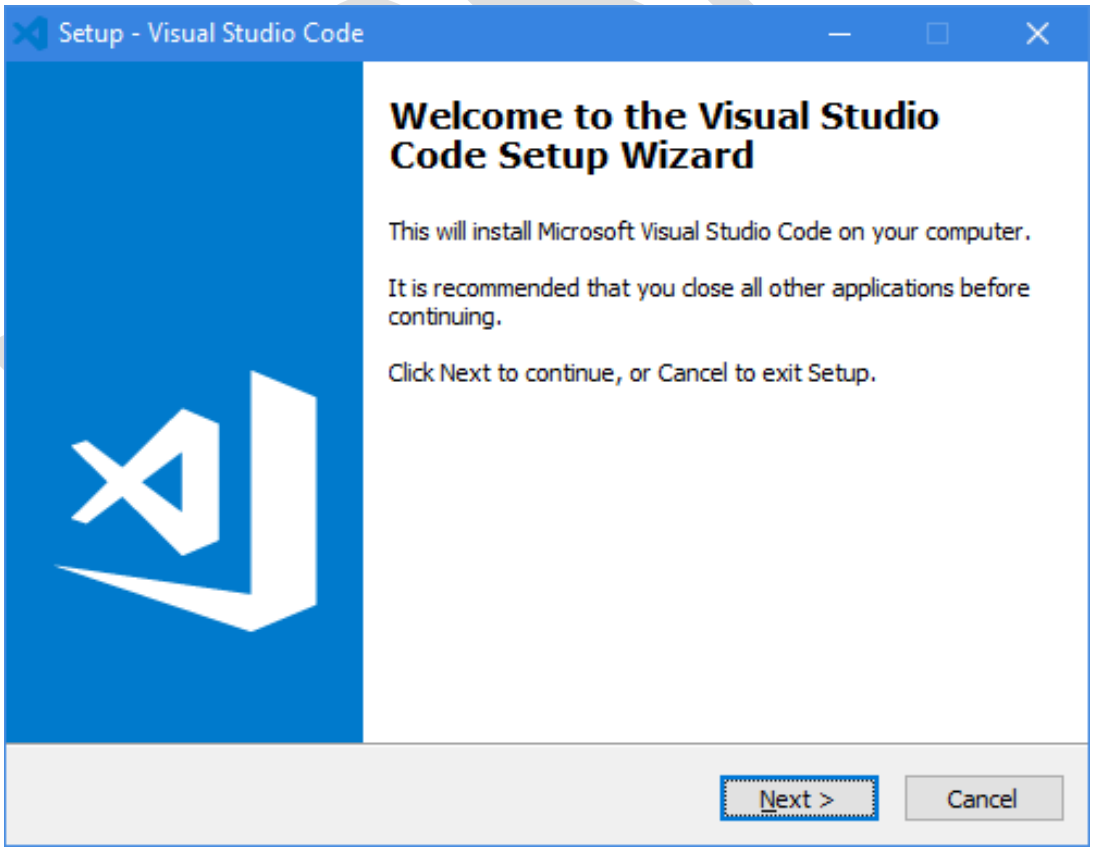
Note: The version number may be different at your practice time.

Go to "Downloads" folder; you can find "VSCodeSetup-x64-1.19.2.exe" file.

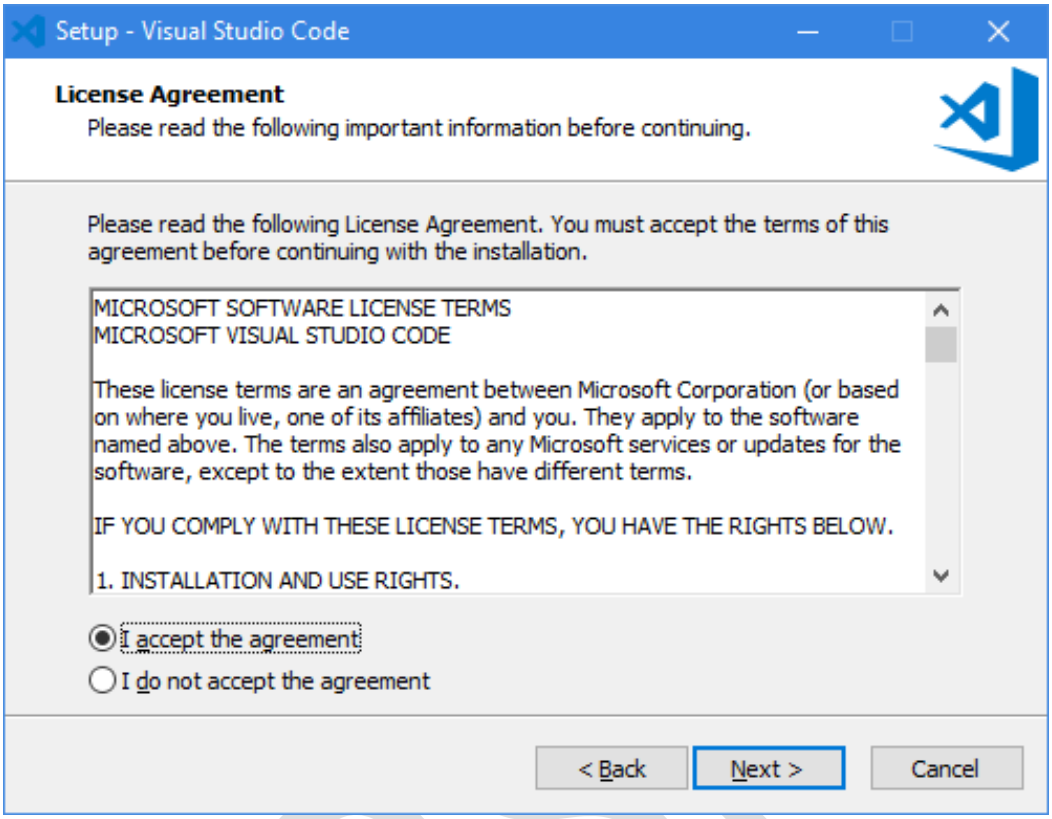


Double click on “VSCodeSetup-x64-1.19.2.exe” file.

Click on “Yes”.

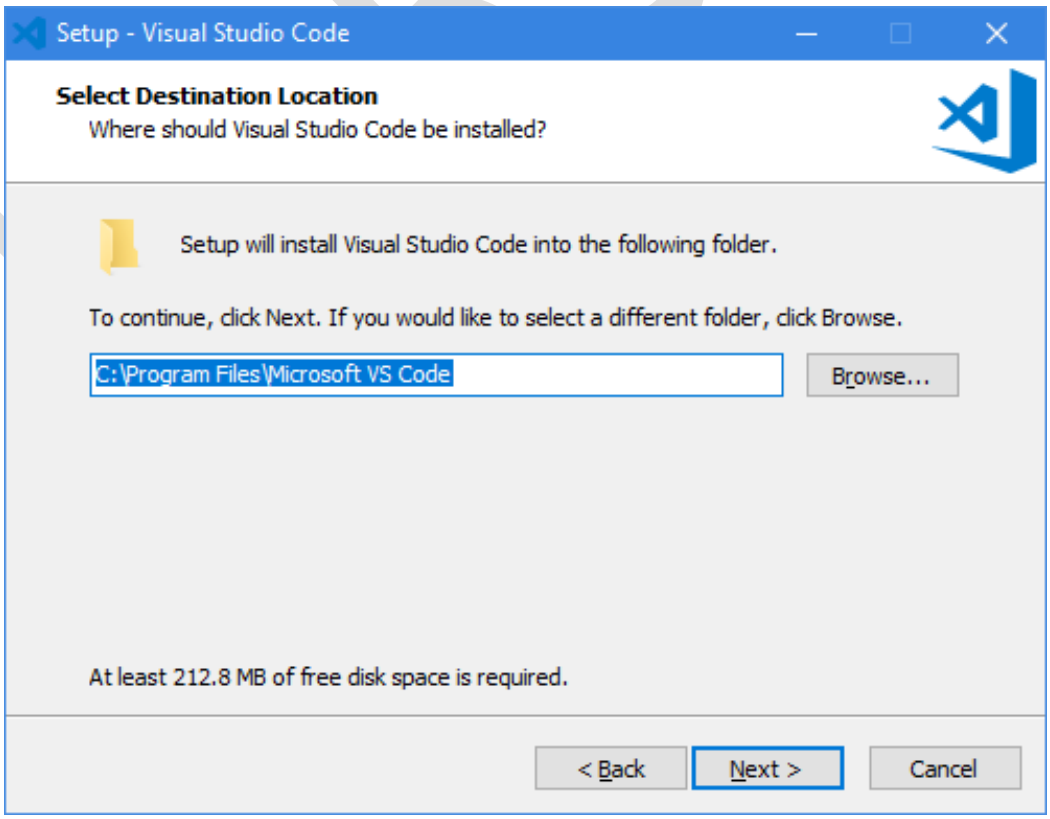


Click on “Next”.

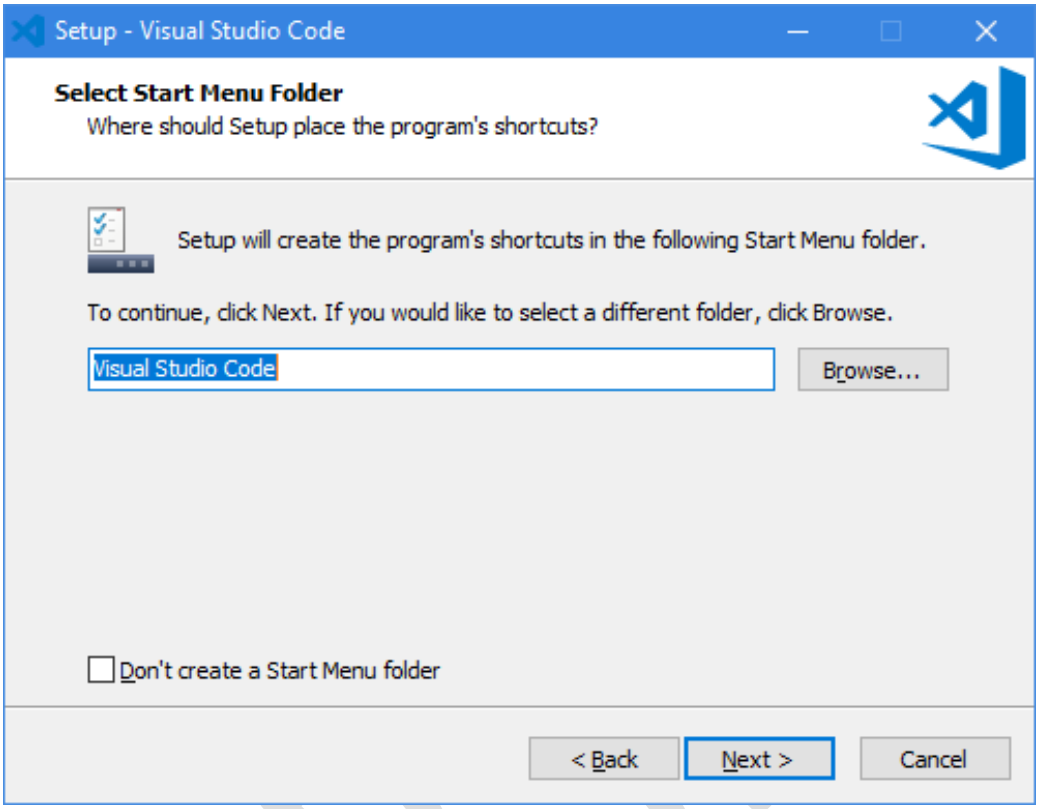


Click on “I accept the agreement”.

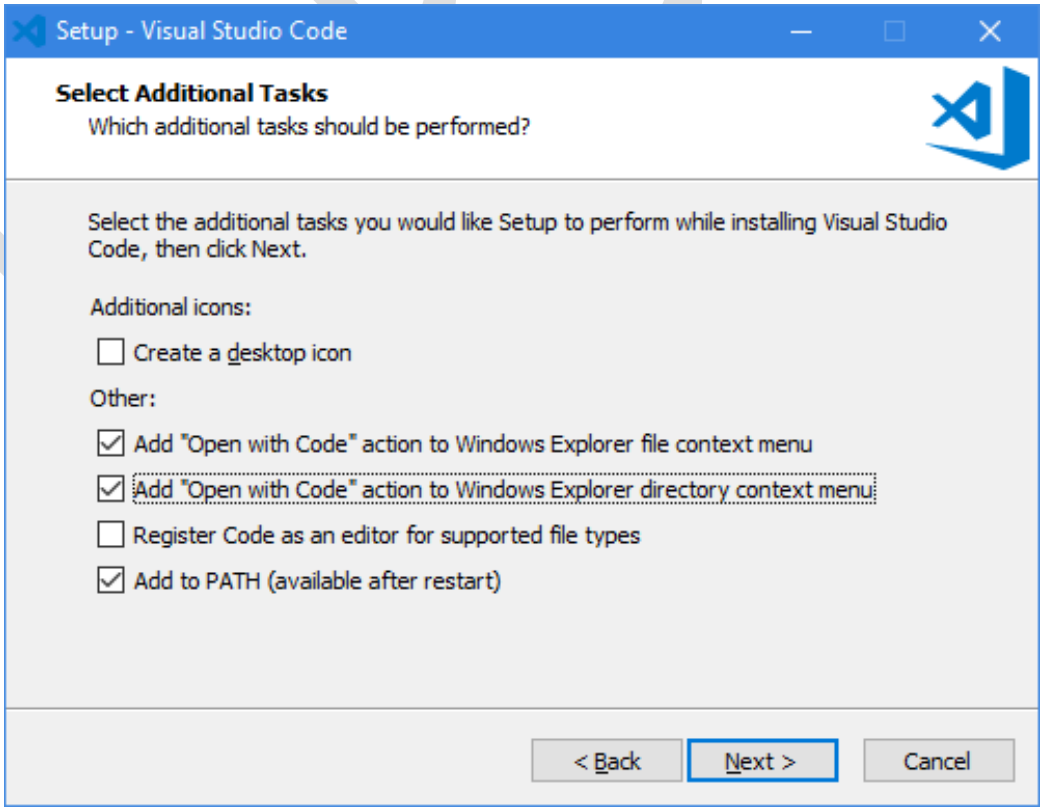
Click on “Next”.



Click on “Next”.



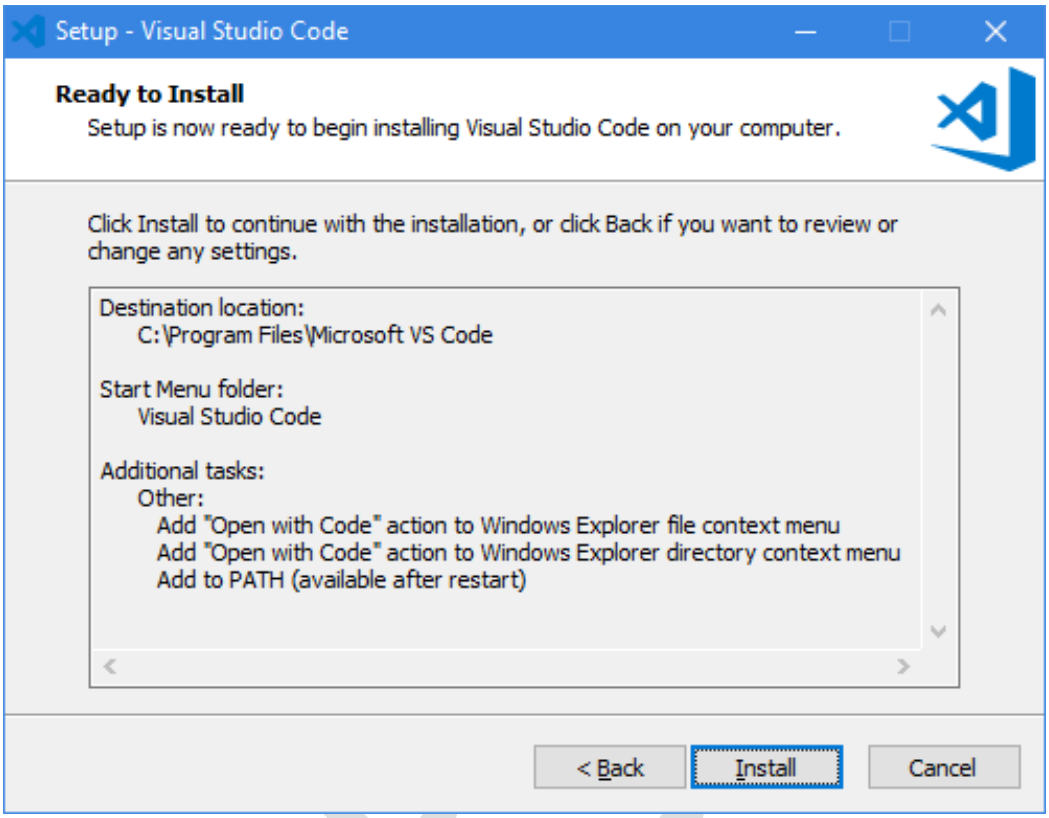
Click on “Next”.



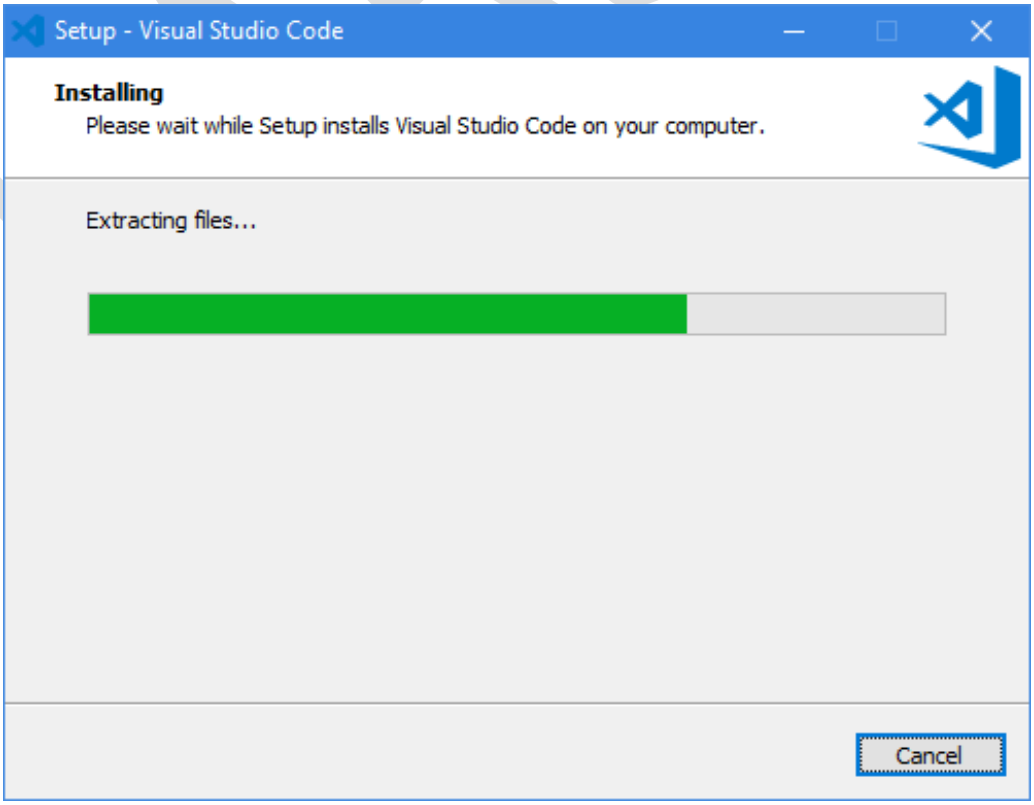
Check the checkbox “Add Open with Code action to Windows Explorer file context menu”.

Check the checkbox “Add Open with Code action to Windows Explorer directory context menu”.

Click on “Next”.



Click on “Install”.



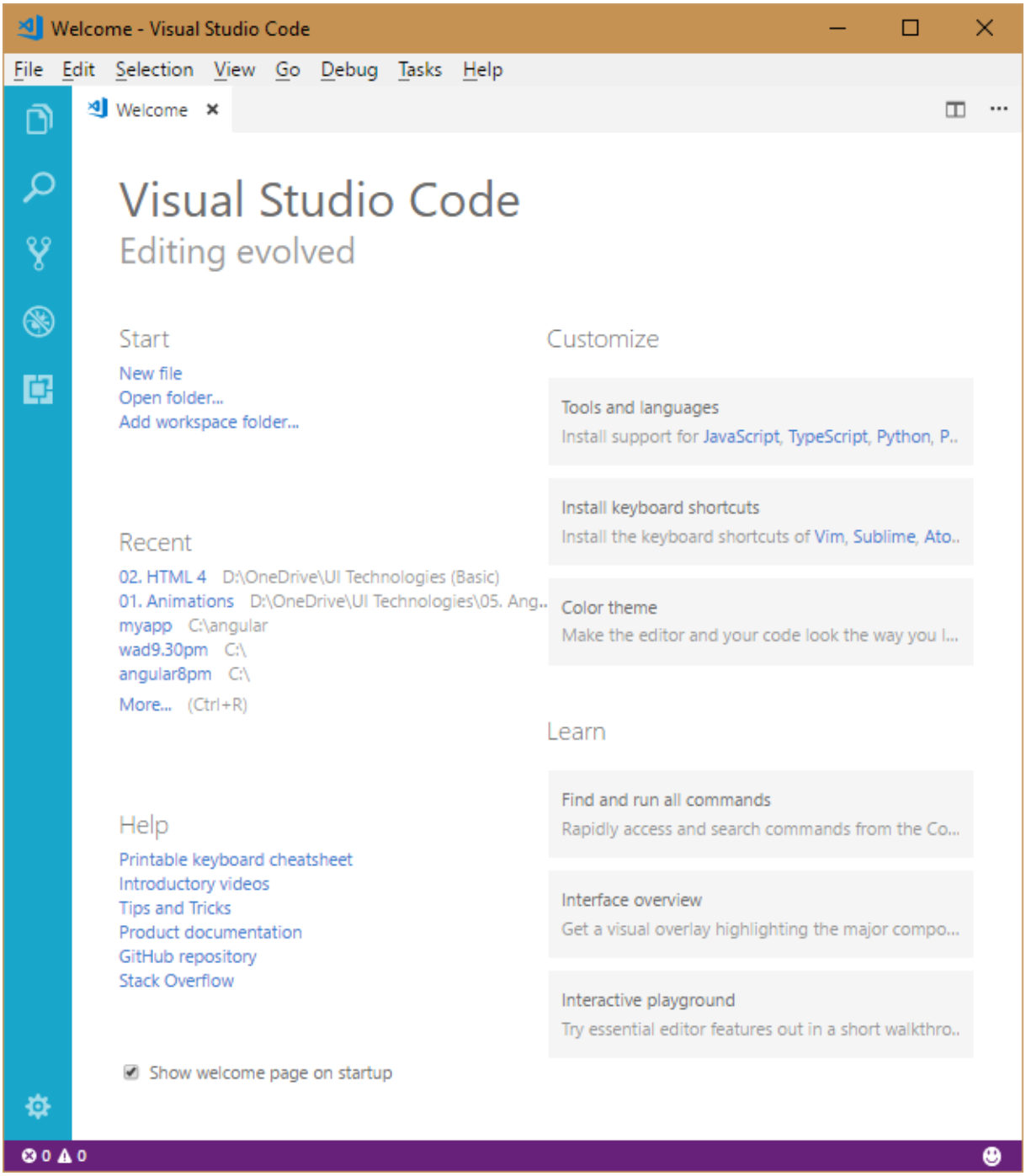
Installation is going on….

Click on “Finish”.

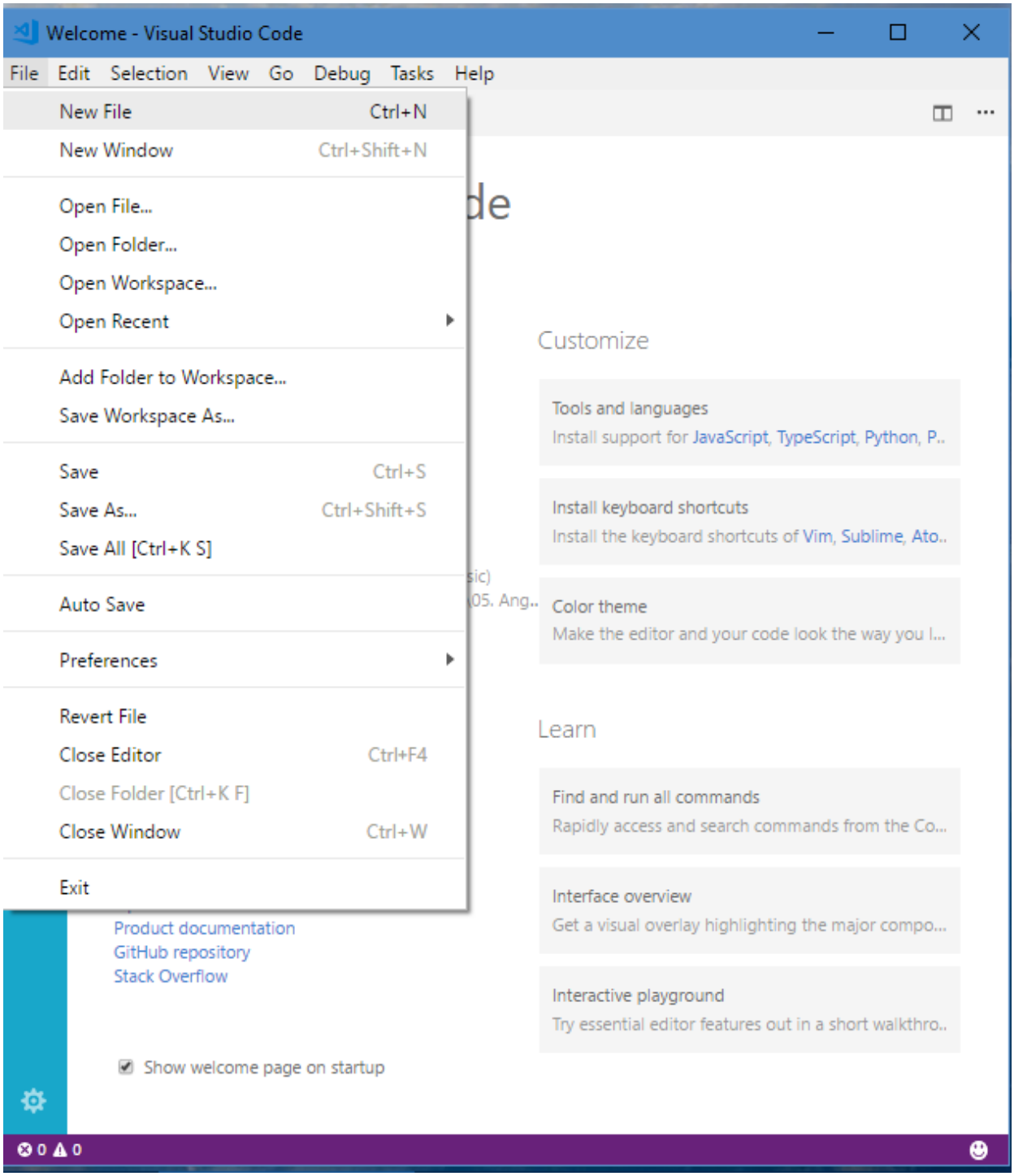
2. Create JavaScript Program

- Open “Visual Studio Code”, by clicking on “Start” – “Visual Studio Code”.

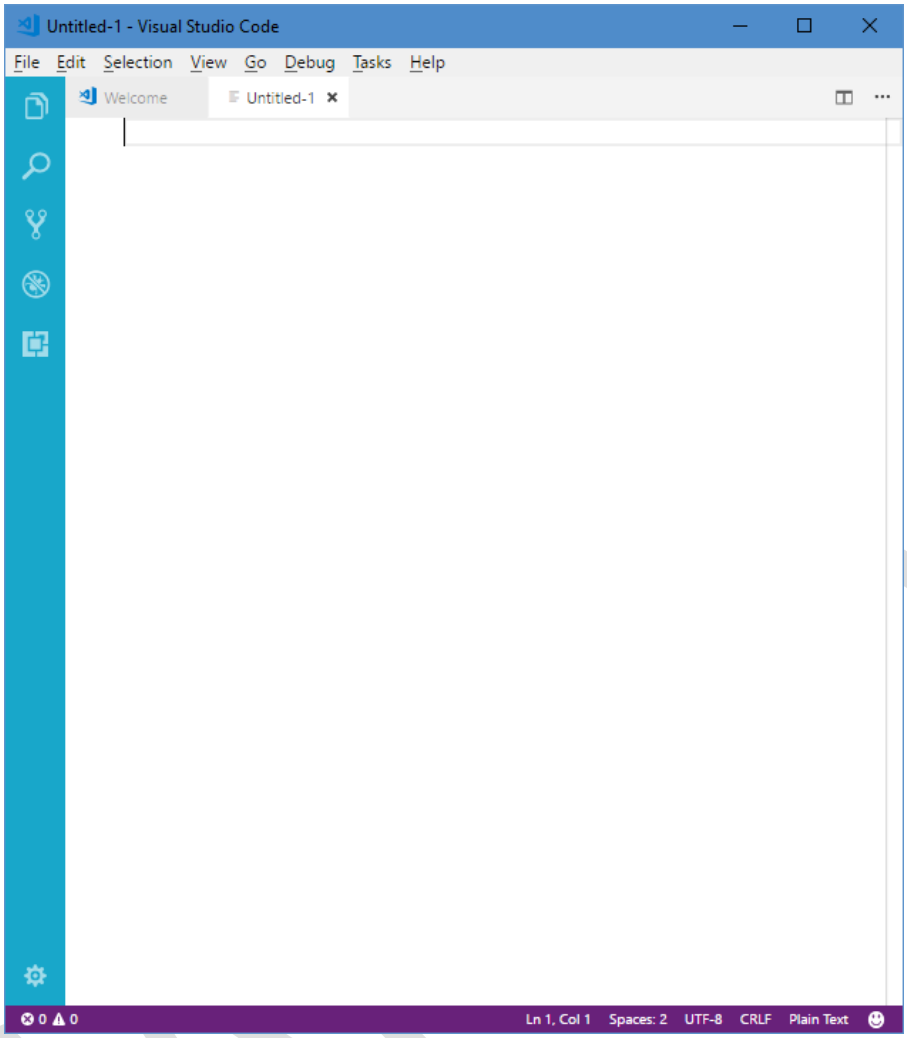

U
UltraSearch
New
V
VideoLAN
New
Visual Studio 2017
New
Visual Studio 2017
Visual Studio Code
Visual Studio Code
Visual Studio Installer
New
Voice Recorder
W
Weather
New
Windows Accessories
Windows Administrative Tools
Windows Defender Security Center
Windows Ease of Access
Windows PowerShell
Windows System
Create
Monday
15
OneDrive
This PC



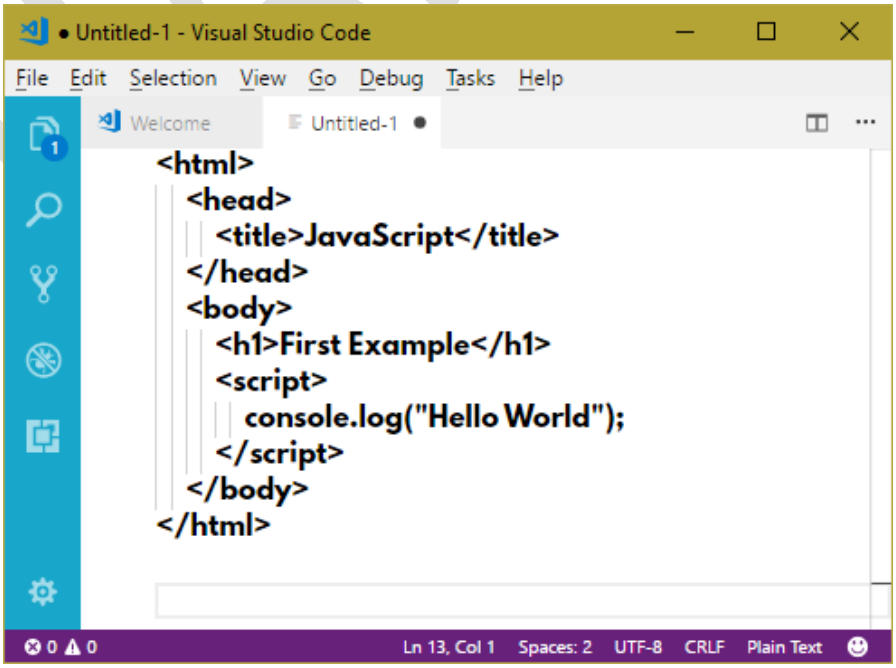
- Visual Studio Code opened.



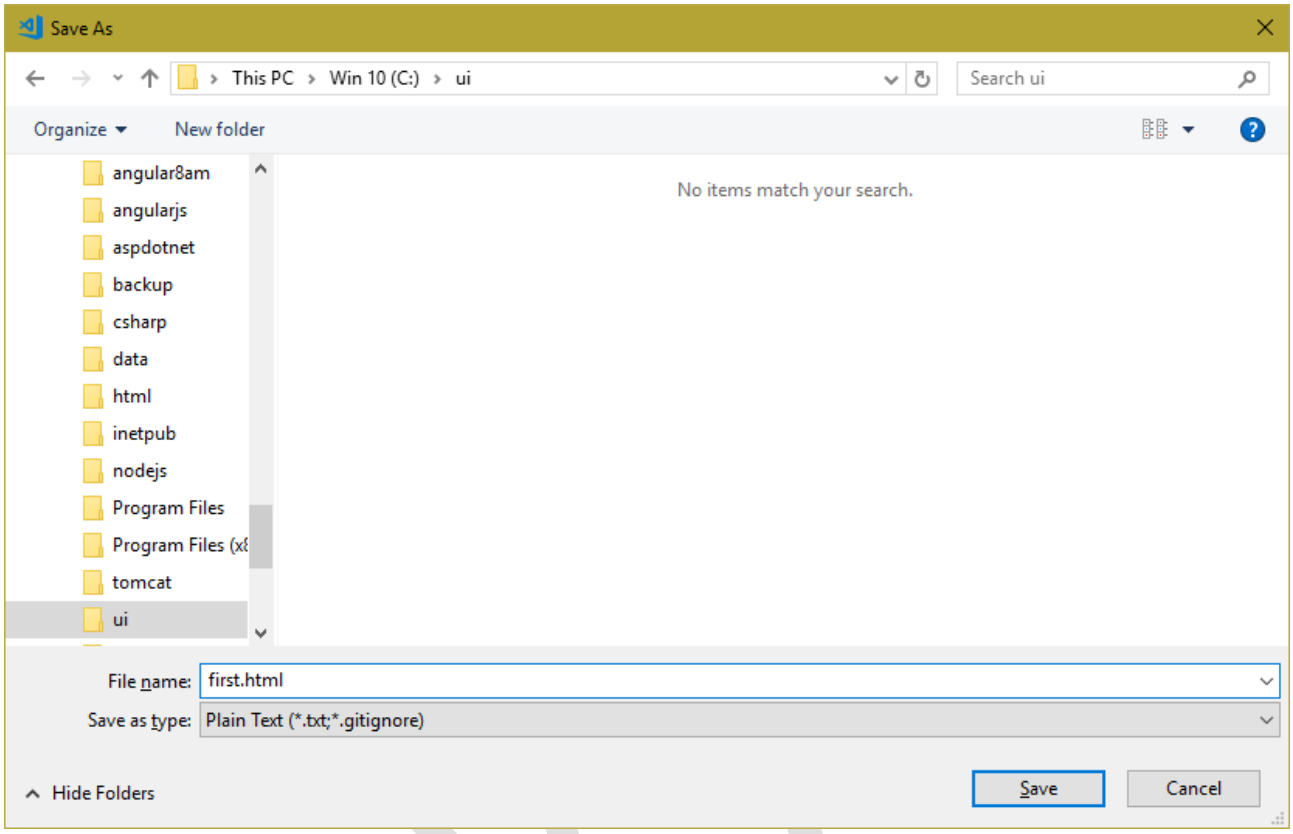
- Go to “File” – “New File”.



Type the program as follows:



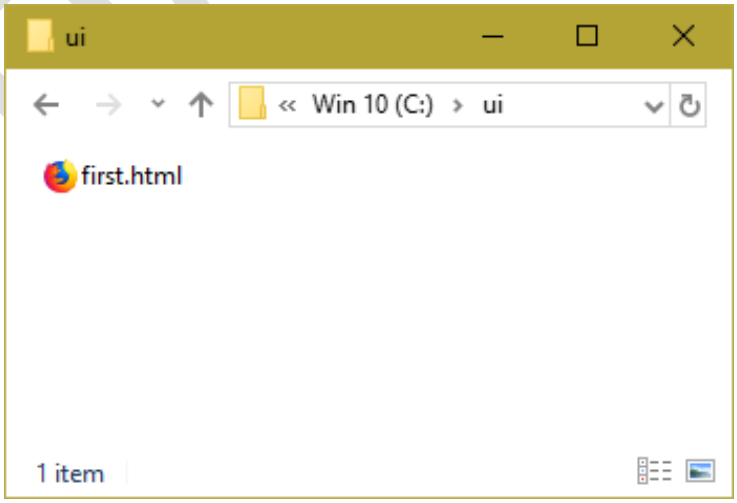
- Go to “File” menu – “Save” (or) Press Ctrl+S.



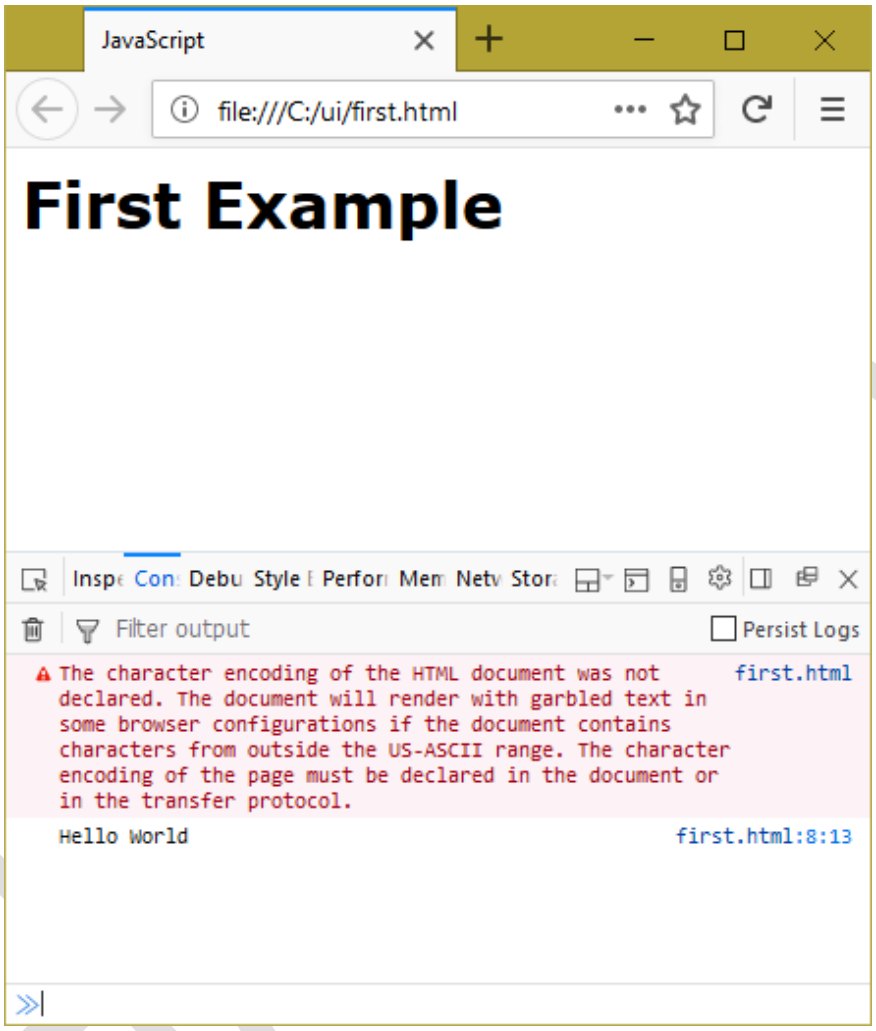
- Go to "c:\" and click on "New folder". Enter the new folder name as "ui".
- Select “c:\ui” folder and enter the filename as “first.html”.
- Click on “Save”.
- Now the typescript file (c:\ui\first.html) is ready.

3. Execute the JavaScript Program

- Go to “Computer” or “This PC” and go to “c:\ui” folder.



- Double click on “first.html” (or) Right click on “first.html” and click on “Open With” – “Mozilla Firefox” / "Open With" – "Google Chrome".
- In the browser, right click in the web page and click on "Inspect Element" (or) press "F12" function key to open console.



Output: Hello World

Variables

- Variable is a “named memory location” in RAM, to store a value temporarily, while executing the program.
- In JavaScript, the variables will be persisted (stored), while the web page is running in the browser.
- The value of variable can be changed any no. of times during the web page execution.
- The data type of the variable can be changed any no. of times during the web page execution, in JavaScript.

Steps for development of variables

- **Declare (create) the variable - optional:** var variablename;
- **Set value into the variable:** variablename = value;
- **Get value from the variable:** variablename

Example on Variables

```
<html>
  <head>
    <title>Variables</title>
  </head>
  <body>
    <h1>Variables</h1>
    <script>
      var a = 10;
      var b = "Hello";
      console.log(a);
      console.log(b);
    </script>
  </body>
</html>
```

Operators

- Operator is a symbol, which represents an operation.
- JavaScript supports the following types of operators.
  1. Arithmetical Operators
  2. Assignment Operators
  3. Increment and Decrement Operators
  4. Relational Operators
  5. Logical Operators
  6. Concatenation Operator

Arithmetical Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder

Example on Arithmetical Operators

```
<html>
  <head>
    <title>Arithmetical operators</title>
```

```
    </head>
    <body>
      <h1>Arithmetical operators</h1>
      <script>
        var a = 10, b = 3;
        var c = a + b; //addition
        var d = a - b; //subtraction
        var e = a * b; //multiplication
        var f = a / b; //division
        var g = a % b; //remainder
        console.log(a); //10
        console.log(b); //3
        console.log(c); //13
        console.log(d); //7
        console.log(e); //30
        console.log(f); //3.3333333333333335
        console.log(g); //1
      </script>
    </body>
  </html>
```

Assignment Operators

=	Assigns to
+=	Add and assigns to
-=	Subtract and assigns to
*=	Multiply and assigns to
/=	Divide and assigns to
%=	Remainder and assigns to

Example on Assignment Operators

```
  <html>
    <head>
      <title>Assignment operators</title>
    </head>
    <body>
      <h1>Assignment operators</h1>
      <script>
        var a;
        a = 100; //assignment operator
        console.log(a); //100
        var b;
        b = a; //assignment operator
        console.log(b); //100
        a += 10;
        console.log(a); //110
        a -= 10;
        console.log(a); //100
        a *= 10;
```

```
            console.log(a); //1000
            a /= 10;
            console.log(a); //100
            a %= 30;
            console.log(a); //10
        </script>
    </body>
</html>
```

Increment and Decrement Operators

++	Increment (+=1)
--	Decrement (-=1)

Example on Increment and Decrement Operators

```
<html>
    <head>
        <title>Increment and decrement operators</title>
    </head>
    <body>
        <h1>Increment and decrement operators</h1>
        <script>
            var a = 10;
            console.log(a); //10
            a++; //increment operator
            console.log(a); //11
            a--; //decrement operator
            console.log(a); //10
        </script>
    </body>
</html>
```

Relational Operators

==	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Example on Relational Operators

```
<html>
    <head>
        <title>Relational operators</title>
    </head>
    <body>
        <h1>Relational operators</h1>
```

```
        <script>
          var x = 100;
          var y = 200;
          var temp1, temp2, temp3, temp4, temp5, temp6;
          temp1 = (x == y);
          console.log(temp1); //false
          temp2 = (x != y);
          console.log(temp2); //true
          temp3 = (x < y);
          console.log(temp3); //true
          temp4 = (x <= y);
          console.log(temp4); //true
          temp5 = (x > y);
          console.log(temp5); //false
          temp6 = (x >= y);
          console.log(temp6); //false
        </script>
      </body>
    </html>
```

Logical Operators

&&	And (both conditions should be true)
||	Or (At least any one condition should be true)
!	Not (given condition will be reverse)

Example on Logical Operators

```
<html>
  <head>
    <title>Logical operators</title>
  </head>
  <body>
    <h1>Logical operators</h1>
    <script>
      var x = 100;
      var y = 200;
      var z = 50;
      var temp1 = ((x < y) && (x > z));
      console.log(temp1); //true
      var temp2 = ((x < y) || (x < z));
      console.log(temp2); //true
      var temp3 = (!(x < y));
      console.log(temp3); //false
    </script>
  </body>
</html>
```

Concatenation Operator

+	Attaches two strings and returns a single string.

Ex: “new” + “delhi” = “newdelhi”

Number + Number = addition

String + String = concatenation

String + Number = concatenation

Number + String = concatenation

Example on Concatenation Operator

```
<html>
  <head>
    <title>Concatenation operator</title>
  </head>
  <body>
    <h1>Concatenation operator</h1>
    <script>
      var s1 = "peers";
      var s2 = "tech";
      var s3;
      s3 = s1 + s2; //string + string
      console.log(s3); //peerstech
    </script>
  </body>
</html>
```

Control Statements

- Control statements are used to control (change) the program execution flow.
- These are used to make the execution flow jump forward / jump backward.
- JavaScript supports two types of control statements:
  1. Conditional Control Statements: Used to jump forward.
  2. Looping Control Statements: Used to jump backward.

1. Conditional Control Statements

- If
- Switch-case

2. Looping Control Statements

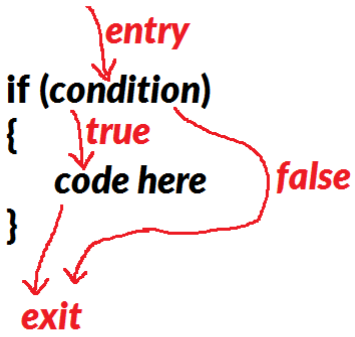
- While
- Do-while
- For

if

- "If" statement is used to check a condition, and execute the code only if the condition is TRUE.

- Types of "if"
    1. If
    2. If-else
    3. Else-if
    4. Nested if

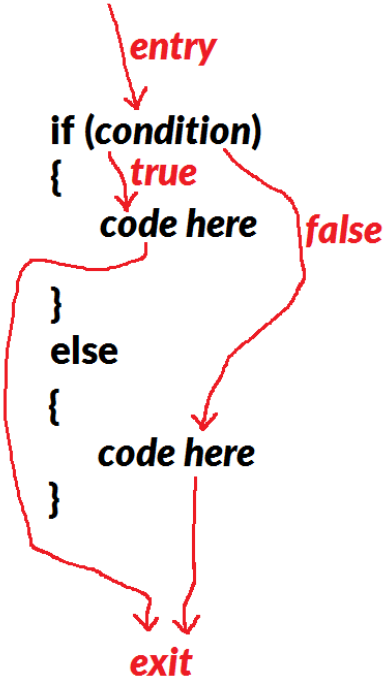
Simple If



Example of "Simple if"

```
<html>
  <head>
    <title>If</title>
  </head>
  <body>
    <h1>If</h1>
    <script>
      var n = 10;
      if (n == 10)
      {
        console.log("n is equal to 10");
      }
      if (n != 10)
      {
        console.log("n is not equal to 10");
      }
    </script>
  </body>
</html>
```

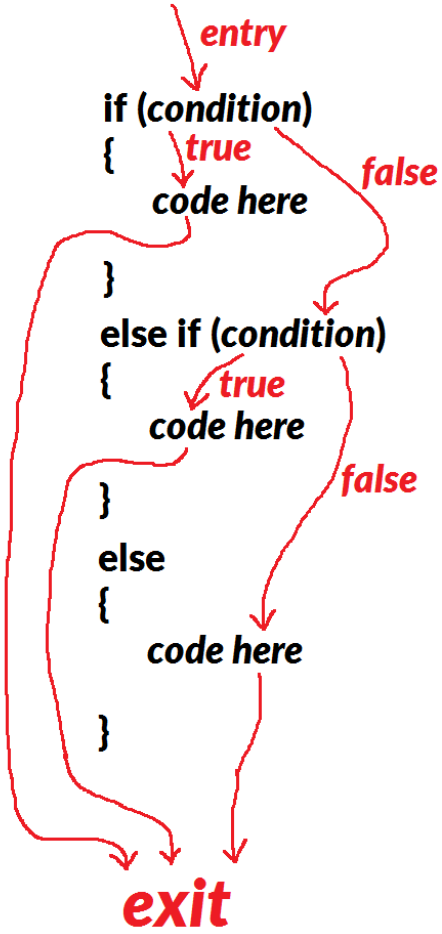
If Else



Example of "if-else"

```
<html>
  <head>
    <title>If-else</title>
  </head>
  <body>
    <h1>If-else</h1>
    <script>
      var n = 10;
      if (n == 10)
      {
        console.log("n is equal to 10");
      }
      else
      {
        console.log("n is not equal to 10");
      }
    </script>
  </body>
</html>
```

Else if

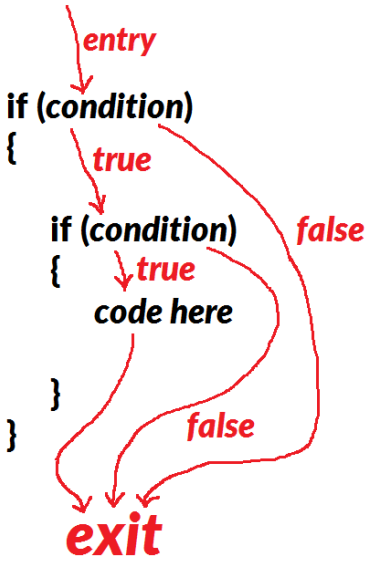


Example of "else-if"

```
<html>
  <head>
    <title>else-if</title>
  </head>
  <body>
    <h1>else-if</h1>
    <script>
      var a = 10, b = 20;
      if (a < b)
      {
        console.log("a is less than b");
      }
      else if (a > b)
      {
        console.log("a is greater than b");
      }
      else
      {
        console.log("a and b are equal");
      }
```

```
    </script>
  </body>
</html>
```

Nested If



Example of "nested if"

```
<html>
  <head>
    <title>Nested if</title>
  </head>
  <body>
    <h1>Nested if</h1>
    <script>
      var a = 10, b = 20;
      if (a != b)
      {
        if (a > b)
        {
          console.log("a is greater than b");
        }
        else
        {
          console.log("a is less than b");
        }
      }
      else
      {
        console.log("a and b are equal");
      }
    </script>
  </body>
</html>
```

Switch-case

- It is used to check a variable's value, whether it matches with any one of the set of cases, and execute the code of the matched case.
- **Syntax:**

```
switch (variable )
{
    case value1: code here ; break;
    case value2: code here ; break;
    ...
    default: code here ; break;
}
```

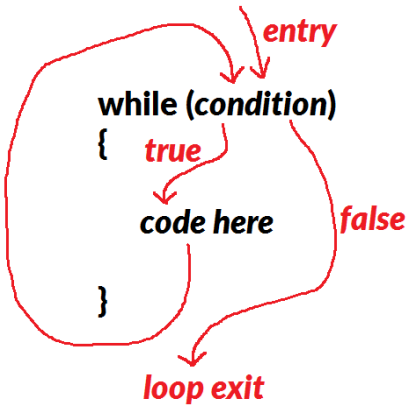
Example on Switch-case

```
<html>
  <head>
    <title>switch-case</title>
  </head>
  <body>
    <h1>switch-case</h1>
    <script>
      var n = 7;
      var monthname;
      switch (n)
      {
        case 1: monthname = "Jan"; break;
        case 2: monthname = "Feb"; break;
        case 3: monthname = "Mar"; break;
        case 4: monthname = "Apr"; break;
        case 5: monthname = "May"; break;
        case 6: monthname = "Jun"; break;
        case 7: monthname = "Jul"; break;
        case 8: monthname = "Aug"; break;
        case 9: monthname = "Sep"; break;
        case 10: monthname = "Oct"; break;
        case 11: monthname = "Nov"; break;
        case 12: monthname = "Dec"; break;
        default: monthname = "Unknown"; break;
      }
      console.log(monthname);
    </script>
  </body>
</html>
```

While

- "While" statement is used to execute the code repeatedly, while the condition is TRUE.

HARSHA



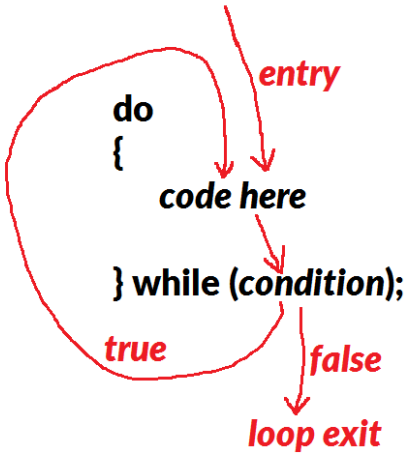
Example on While

```
<html>
  <head>
    <title>While</title>
  </head>
  <body>
    <h1>While</h1>
    <script>
      var i = 1;
      while (i <= 10)
      {
        console.log(i);
        i++;
      }
    </script>
  </body>
</html>
```

Do-While

- "Do-while" loop is mostly same as "while" loop.
- The difference is: "while" loop checks the condition for the first time also; but "do-while" loop doesn't check the condition for the first time.

HARSHA



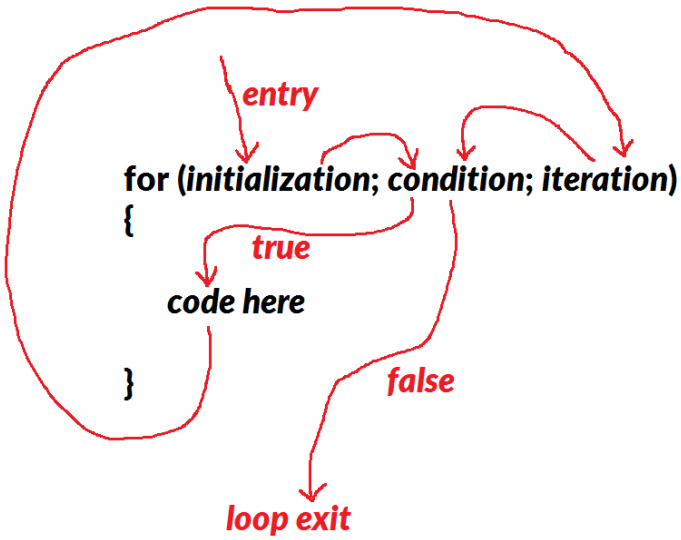
Example on Do-While

```
<html>
  <head>
    <title>do-while</title>
  </head>
  <body>
    <h1>do-while</h1>
    <script>
      var i = 1;
      do
      {
        console.log(i);
        i++;
      } while (i <= 10);
    </script>
  </body>
</html>
```

For

- "For" loop is mostly same as "while" loop.
- The difference is: "While" loop has the initialization, condition and iteration in three different places, so that it will be difficult to understand, if the code increases. But "for" loop has the initialization, condition and iteration in the same line, so that it will be easy to understand.

HARSHA

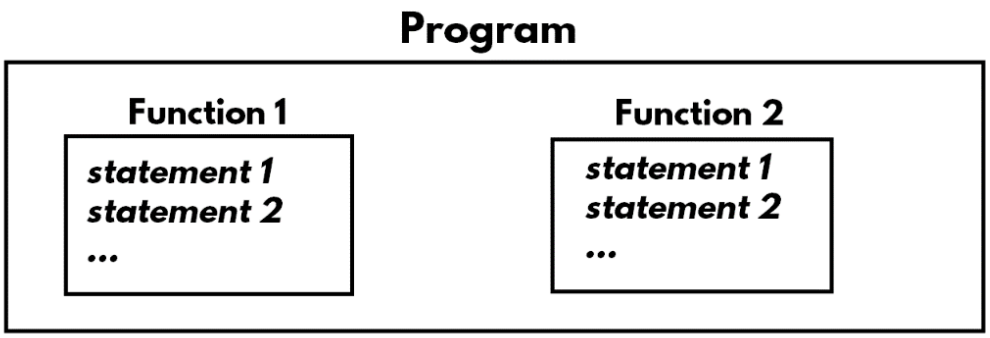


Example on For

```
<html>
  <head>
    <title>for</title>
  </head>
  <body>
    <h1>for</h1>
    <script>
      var i;
      for (i = 1; i <= 10; i++)
      {
        console.log(i);
      }
    </script>
  </body>
</html>
```

Functions

- Function is a re-usable “block” of the program, which is a set of statements with a name.
- The large program can be divided into many parts; each individual part is called as “function”.
- Functions are re-usable. That means functions can be called anywhere and any no. of times within the program. Every time when we call the function, the execution flow jumps to the function definition; executes the function and comes back to the calling portion.



Steps for development of functions

- **Create the function:**

```
function functionname(parameter1, parameter2, ...)
{
    code here
}
```

Parameters: The values that are passed from “calling portion” to the “function definition” are called as “arguments” and “parameters”.

Return: The value that is passed from “function definition” to the “calling portion” is called as “return”.

- **Call the function:**

```
functionName(value1, value2, ...);
```

- **Access the list of arguments**

```
arguments
```

Note: Every function has a property called "arguments", which represents the list of arguments that are passed to the function. arguments = { 0: argument1, 1: argument1, … }

Simple Example on Functions

```
<html>
  <head>
    <title>functions</title>
  </head>
  <body>
    <h1>functions</h1>
    <script>
      function country()
      {
        console.log("India");
```

```
        }
        country();
        country();
        country();
      </script>
    </body>
  </html>
```

Calling Function in Another Function - Example

```
<html>
  <head>
    <title>functions</title>
  </head>
  <body>
    <h1> functions</h1>
    <script>
      function country()
      {
        console.log("India");
      }
      function city()
      {
        console.log("Hyderabad");
        country();
      }
      country();
      city();
    </script>
  </body>
</html>
```

Arguments and Return

```
<html>
  <head>
    <title>functions</title>
  </head>
  <body>
    <h1>functions</h1>
    <script>
      function add(a, b)
      {
        var c; //local variable
        c = a + b;
        return (c);
      }
      var result;
      result = add(10, 20);
      console.log(result); //30
      var x = 100;
      var y = 250;
      var result2 = add(x, y);
```

```
        console.log(result2); //350
      </script>
    </body>
  </html>
```

Arguments - Example

```
<html>
  <head>
    <title>Arguments</title>
  </head>
  <body>
    <h1>Arguments</h1>
    <script>
      function fun1(a, b)
      {
        console.log(arguments);
      }
      fun1(10, 20);
      fun1(10, 20, 30);
    </script>
  </body>
</html>
```

Recursion

- Recursion is a technique of calling a function inside itself.
- Whenever a function calls itself, it is said to be "recursion".
- We should check the condition inside the function and call the same function, only if the condition is TRUE.

Example: Factorial of number = n * n-1 * n-2 * n-3 … * 0
Factorial of 5 = 5 * 4 * 3 * 2 * 1 = 120

Syntax:

```
function functionname()
{
  samefunctionname();
}
```

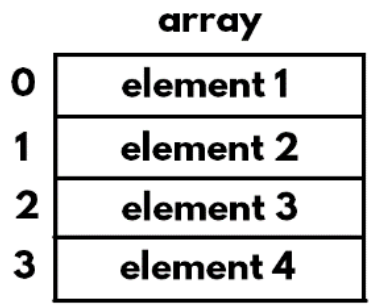
Recursion - Example

```
<html>
  <head>
    <title>Recursion</title>
  </head>
  <body>
    <h1>Recursion</h1>
    <script>
      function factorial(n)
```

```
        {
            if (n == 0)
            {
                return 1;
            }
            else
            {
                return n * factorial(n - 1);
            }
        }
        console.log(factorial(6));
    </script>
  </body>
</html>
```

Arrays

- Array is a collection of multiple values.
- The no. of elements of the array is called as “array size” or “array length”.
- Index (starts from 0) will be maintained for each element automatically.
- For example, you can store list of friends inside an array.
- In JavaScript, there is no rule of maintain "same type" of values in the array. JavaScript array can store the value of ANY data type. For example, you can store numbers, strings, objects in the same array.



Steps for development of arrays

- **Create an array:** var variablename = [value1, value2, …];
- **Set value into the array element:** variablename[index] = value;
- **Get value from the array element:** variablename[index]
- **Get size of the array:** variablename.length
- **Add new element into the array:** variablename.push(newvalue);
- **Remove an existing element:** variablename.splice(index, no. of elements to remove);
- **Insert new element in the middle:** variablename.splice(index, 0, newvalue);

Example on Arrays

```
<html>
  <head>
    <title>arrays</title>
  </head>
  <body>
    <h1>arrays</h1>
    <script>
      var n = [10, 20, 50, 150, 220];
      console.log(n.length);
      console.log(n[0]);
      console.log(n[1]);
      console.log(n[2]);
      console.log(n[3]);
      console.log(n[4]);
    </script>
  </body>
</html>
```

Example on Push

```
<html>
  <head>
    <title>Push</title>
  </head>
  <body>
    <h1>Push</h1>
    <script>
      var n = [10, 20, 50, 150, 220];
      console.log(n);
      n.push(500);
      console.log(n);
    </script>
  </body>
</html>
```

Example on Splice

```
<html>
  <head>
    <title>Splice</title>
  </head>
  <body>
    <h1>Splice</h1>
    <script>
      var n = [10, 20, 50, 150, 220];
      console.log(n);
      n.splice(3, 1);
      console.log(n);
    </script>
  </body>
</html>
```

HARSHA

Example on Insert

```
<html>
  <head>
    <title>Splice</title>
  </head>
  <body>
    <h1>Splice</h1>
    <script>
      var n = [10, 20, 50, 150, 220];
      console.log(n);
      n.splice(3, 0, 800);
      console.log(n);
    </script>
  </body>
</html>
```

Object Oriented Programming in JavaScript

Introduction to Objects

- Object Oriented Programming (OOP) is a programming paradigm (programming style), which is based on the concept of "objects".
- Object represents a physical item / entity.
- Object is a collection of two types of members:
  1. Properties / Fields
  2. Methods
- **Properties / Fields:** Details about the object. Properties are the variables stored inside the object. Properties are used to store data about specific person, product or thing.
- **Methods:** Manipulations on the properties. Methods are the functions stored inside the object. Functions read values from properties and/or write values into properties.

  Example:

  "Car" object
  - Properties
    - carModel: Honda City
    - carColor: Black
    - colorNo: 1234
  - Methods
    - start( )
    - changeGear( )
    - stop( )

- In the above example, the "Car" object has three properties called "carModel", "carColor", "colorNo", which have respective values. The

Types of Object Oriented Programming (OOP) languages

- We have two types of OOP languages:
  1. Class-based Object Oriented Programming Language
  2. Prototype-based Object Oriented Programming Language

Creating Objects

- We can create object in 2 ways:
  1. Object Literals
  2. Constructor Function

Object Literals

- Object literals are represented as curly braces { }, which can include properties and methods.
- The property and value are separated with : symbol.
- **Syntax:** { “property”: value, “method”: function( ) { … } }

Example 1 on Object Literals

```
<html>
  <head>
    <title>Object Literals</title>
  </head>
  <body>
    <h1>Object Literals</h1>
    <script>
      var stu = { studentid":1, studentname: "scott", marks: 80 };
      console.log(stu);
      console.log(stu.studentid);
      console.log(stu.studentname);
      console.log(stu.marks);
    </script>
  </body>
</html>
```

Example 2 on Object Literals

```
<html>
  <head>
    <title>Object Literals - Methods</title>
  </head>
  <body>
    <h1>Object Literals - Methods</h1>
    <script>
```

```
        var stu = {
          studentid: 1, studentname: "scott", marks: 80, "result": function ()
          {
            if (this.marks >= 35)
            {
              return "Pass";
            }
            else
            {
              return "Fail";
            }
          }
        };
        console.log(stu);
        console.log(stu.studentid);
        console.log(stu.studentname);
        console.log(stu.marks);
        console.log(stu.result());
      </script>
    </body>
  </html>
```

Constructor Function

- Constructor function is a function that receives an empty (new) object, initializes properties and methods to the object.
- The "this" keyword inside the constructor function represents the current working object. For example, if it is called for the first time, the "this" keyword represents the first object; if it is called second time, the "this" keyword represents the second object.
- The constructor function can receive one or more parameters and initializes the same values into the respective properties.
- The "reference variable" stores the reference (address) of the object and used to access its members (properties and methods), outside the object literal / constructor function.

Syntax of Constructor Function

```
function functionname(arguments)
{
  this.property = value;
  this.method = function( ) { … };
}
var variablename = new functionname( );
```

Example on Constructor Function

```
<html>
  <head>
```

```
    <title>Constructor Function</title>
  </head>
  <body>
    <h1>Constructor Function</h1>
    <script>
      function Student(a, b, c)
      {
        this.studentid = a;
        this.studentname = b;
        this.marks = c;
        this.result = function ()
        {
          if (this.marks >= 35)
          {
            return "Pass";
          }
          else
          {
            return "Fail";
          }
        };
      }
      var stu = new Student(1, "Scott", 80);
      console.log(stu);
      console.log(stu.studentid);
      console.log(stu.studentname);
      console.log(stu.marks);
      console.log(stu.result());
    </script>
  </body>
</html>
```

Object.Keys

- The “Object.keys” method is used to retrieve the list of properties of an object as an array.
- Sometimes, you will get data from server / browser storage. Then you don't know what properties / methods are present inside the object. Then you have to use Object.keys() are used to properties / methods of the object and also read its values programmatically.

  Syntax: Object.keys(reference variable);

  Example: Object.keys(stu);
- This is useful if you don’t know what properties are exist in the object.

Example on Object.Keys

```
<html>
  <head>
    <title>Keys</title>
  </head>
  <body>
    <h1>Keys</h1>
```

```
<script>
  function Student(a, b, c)
  {
    this.studentid = a;
    this.studentname = b;
    this.marks = c;
    this.result = function ()
    {
      if (this.marks >= 35)
      {
        return "Pass";
      }
      else
      {
        return "Fail";
      }
    };
  }
  var stu = new Student(1, "Scott", 80);
  console.log(stu);
  var keys = Object.keys(stu);
  console.log(keys);
  for (var i = 0; i < keys.length; i++)
  {
    console.log(stu[keys[i]]);
  }
</script>
</body>
</html>
```

JSON

- "JSON" stands for "JavaScript Object Notation".
- JSON is similar to "Object Literal", but having the following differences.
  - In "Object Literal", double quotes are optional for the properties; In "JSON", double quotes are must for properties.
  - In "Object Literal", methods are allowed; In "JSON", methods are not allowed.
- JSON is mainly used as data exchange format; it can be transferred from browser to server; and vice versa; and also it is can be stored in the local storage and session storage.

Syntax:

{ "property" : value, "property" : value, … }

Stringify

- The “JSON.stringify” is used to convert “Object Literal” to “JSON” format. JSON is a text format which follows “JavaScript Object Literal” syntax. JSON is mainly used for store or exchange data between browser and server.

Syntax: JSON.stringify(reference variable)

Example: JSON.stringify(stu)

Example on JSON.stringify

```
<html>
  <head>
    <title>Stringify</title>
  </head>
  <body>
    <h1>Stringify</h1>
    <script>
      function Student(a, b, c)
      {
        this.studentid = a;
        this.studentname = b;
        this.marks = c;
        this.result = function ()
        {
          if (this.marks >= 35)
          {
            return "Pass";
          }
          else
          {
            return "Fail";
          }
        };
      }
      var stu = new Student(1, "Scott", 80);
      var str = JSON.stringify(stu);;
      console.log(str);
    </script>
  </body>
</html>
```

Parse

- The “JSON.parse” is used to convert “JSON” to “Object Literal” format.

  Syntax: JSON.parse(json data)

  Example: JSON.parse(‘ { “a”:10, “b”: 20 } ’)

Example on JSON.parse

```
<html>
  <head>
    <title>Parse</title>
  </head>
  <body>
    <h1>Parse</h1>
    <script>
      var s = '{ "studentid":1, "studentname": "scott", "marks": 80 }';
```

```
        console.log(s);
        var stu = JSON.parse(s);
        console.log(stu);
        console.log(stu.studentid);
        console.log(stu.studentname);
        console.log(stu.marks);
      </script>
    </body>
  </html>
```

Object Array Literal

- "Object Array Literal" is a collection of "object literals", stored as an array.
- It is used to represent group of records, for example list of students.
- Each object inside the object array represents one single record; for example "one student".

Syntax:

```
[
  { property : value, property : value, … },
  { property : value, property : value, … },
  { property : value, property : value, … },
  …
]
```

Example on Object Array Literal

```
<html>
  <head>
    <title>Object Array Literal</title>
  </head>
  <body>
    <h1>Object Array Literal</h1>
    <script>
      var employees =
        [
          { "empid": 101, "empname": "Scott", "salary": 4000 },
          { "empid": 102, "empname": "Smith", "salary": 5690 },
          { "empid": 103, "empname": "Allen", "salary": 6723 },
          { "empid": 104, "empname": "John", "salary": 8729 }
        ];
      console.log(employees);

      for (var i = 0; i < employees.length; i++)
      {
        console.log(employees[i].empid);
        console.log(employees[i].empname);
        console.log(employees[i].salary);
      }
    </script>
```

```
  </body>
</html>
```

Object Array

- "Object Array Literal" is a collection of "objects created using constructor function", stored as an array.

Syntax:

```
[
  new constructorfunction(argument1, argument2, …),
  new constructorfunction(argument1, argument2, …),
  new constructorfunction(argument1, argument2, …),
  …
]
```

Example on Object Array

```
<html>
  <head>
    <title>Object Array</title>
  </head>
  <body>
    <h1>Object Array</h1>
    <script>
      function Employee(a, b, c)
      {
        this.empid = a;
        this.empname = b;
        this.salary = c;
      }

      var employees =
        [
          new Employee(101, "Scott", 4000),
          new Employee(102, "Smith", 5690),
          new Employee(103, "Allen", 9500),
          new Employee(104, "John", 7400)
        ];
      console.log(employees);
      for (var i = 0; i < employees.length; i++)
      {
        console.log(employees[i]);
      }
    </script>
  </body>
</html>
```

HARSHA

Prototype

- The "prototype" generally represents model of the object, which contains list of properties and methods of the object.
- Every Constructor Function has a property called "prototype".
- Any properties or methods added to "prototype", will be automatically added to every object that is created based on the same constructor function.

Example on Prototype

```
<html>
  <head>
    <title>Prototype</title>
  </head>
  <body>
    <h1>Prototype</h1>
    <script>
      function Student(a, b)
      {
        this.studentid = a;
        this.studentname = b;
      }
      Student.prototype.marks = 70;
      Student.prototype.result = function ()
      {
        if (this.marks >= 35)
          return "Pass";
        else
          return "Fail";
      };
      var s = new Student(101, "scott");
      console.log(s);
      console.log(s.studentid);
      console.log(s.studentname);
      console.log(s.marks);
      console.log(s.result());
    </script>
  </body>
</html>
```

Inheritance

- The process of creating an object based on another object is called as “inheritance”.
- So all the properties and methods of the parent object is inherited into the child object.

Syntax:

```
function parentconstructorfunction(arguments)
{
    …
```

```
}

function childconstructorfunction(arguments)
{
   parentconstructorfunction.call(this, arguments);
   …
}
```

Example on Inheritance

```
<html>
  <head>
    <title>Inheritance</title>
  </head>
  <body>
    <h1>Inheritance</h1>
    <script>
      function Person(a, b)
      {
        this.name = a;
        this.email = b;
      }
      function Student(a, b, c)
      {
        Person.call(this, a, b);
        this.marks = c;
      }
      var stu = new Student("scott", "scott@gmail.com", 70);
      console.log(stu);
    </script>
  </body>
</html>
```

Data Types

- "Data type" specifies type of data that you want to store in the variable or property.
- JavaScript supports the following data types:
  1. number : Any numbers
  2. string : Collection of characters
  3. Boolean : true, false
  4. undefined : undefined
  5. object : { }, new
  6. function : function() { }

Example on Data Types

```
<html>
  <head>
    <title>Data Types</title>
  </head>
  <body>
    <h1>Data Types</h1>
    <script>
      var a = 10;
      var b = 20.3248;
      var c = "hello";
      var d = 'hello'
      var e = true;
      var f = false;
      var g;
      var h = {};
      var i = new fun1();
      var j = function () { };
      console.log(a);
      console.log(b);
      console.log(c);
      console.log(d);
      console.log(e);
      console.log(f);
      console.log(g);
      console.log(h);
      console.log(i);
      console.log(j);

      function fun1()
      {
      }
    </script>
  </body>
</html>
```

String

- "String" is a collection of characters. The characters include with the following:
  1. Uppercase alphabets : A-Z
  2. Lowercase alphabets : a-z
  3. Digits : 0-9
  4. Symbols : $ # @ & * etc.
  5. Spaces
- JavaScript string literals should be in either single quotes or double quotes.

  Ex: 'hello123'

  "hello123"

HARSHA

typeof

- The "typeof" keyword is used to get the data type of given value.

  Syntax: typeof value

  Example: typeof 10

Example on typeof

```
<html>
  <head>
    <title>typeof</title>
  </head>
  <body>
    <h1>typeof</h1>
    <script>
      var a = 10;
      var b = 20.3248;
      var c = "hello";
      var d = 'hello'
      var e = true;
      var f = false;
      var g;
      var h = {};
      var i = new fun1();
      var j = function () { };
      console.log(typeof a);
      console.log(typeof b);
      console.log(typeof c);
      console.log(typeof d);
      console.log(typeof e);
      console.log(typeof f);
      console.log(typeof g);
      console.log(typeof h);
      console.log(typeof i);
      console.log(typeof j);

      function fun1()
      {
      }
    </script>
  </body>
</html>
```

undefined vs null

- "undefined" represents "empty value", which is by default to assigned to every uninitialized variables. The developer is not supposed to assign "undefined" manually.
- "null" represents "empty value", which can be assigned by the developer.
- The datatype of "undefined" is "undefined".
- The datatype of "null" is "object".

HARSHA

Syntax of undefined: undefined

Syntax of null: null

Example on undefined vs null

```
<html>
  <head>
    <title>undefined vs null</title>
  </head>
  <body>
    <h1>undefined vs null</h1>
    <script>
      var a;
      var b = null;
      console.log(a);
      console.log(b);
      console.log(typeof a);
      console.log(typeof b)
    </script>
  </body>
</html>
```

== and ===

- == operator checks only value; === operator checks value and data type also.
- == operator internally first converts the right side value in the data type of left side and checks the value. === operator will not perform any automatic conversion and directly checks the value.
- Use == operator to check only value. Use === operator check value and data type

Syntax of == value1 == value2

Syntax of === value1 === value2

Example on == vs ===

```
<html>
  <head>
    <title>== and ===</title>
  </head>
  <body>
    <h1>== and ===</h1>
    <script>
      var a = 10;
      var b = '10';
      console.log(a == b); //true
      console.log(a === b); //false
    </script>
  </body>
</html>
```

String Function

- The "String()" is a pre-defined function, which is used to convert a number into string.

Syntax: String(number value)

Example on String Function

```
<html>
  <head>
    <title>String</title>
  </head>
  <body>
    <h1>String</h1>
    <script>
      var a = 10;
      var b = String(a);
      console.log(a);
      console.log(b);
      console.log(typeof a);
      console.log(typeof b);
    </script>
  </body>
</html>
```

ToString Function

- The "toString()" is a pre-defined function, which is used to convert a number into string.
- The difference between String() and toString() function is:
- The String() function is a global function; The toString() function is available inside number data type.

Syntax: numbervalue.toString()

Example on ToString Function

```
<html>
  <head>
    <title>toString</title>
  </head>
  <body>
    <h1>toString</h1>
    <script>
      var a = 10;
      var b = a.toString();
      console.log(a);
      console.log(b);
      console.log(typeof a);
      console.log(typeof b);
    </script>
  </body>
</html>
```

HARSHA

Number Function

- The "Number()" is a pre-defined function, which is used to convert string to number.
- It returns "NaN", if the string is alphanumerical / alphabetic. "Nan" stands for "Not A Number", which indicates the value is not a number. The datatype of "NaN" is "number".
- It returns "0", if the string is empty / space.

Syntax: Number(string value)

Example on Number Function

```
<html>
  <head>
    <title>Number</title>
  </head>
  <body>
    <h1>Number</h1>
    <script>
      var a = "10";
      var b = Number(a);
      var c = "10ab";
      var d = Number(c);
      var e = "ab10";
      var f = Number(e);
      var g = "ab";
      var h = Number(g);
      var i = "";
      var j = Number(i);
      var k = " ";
      var l = Number(k);
      console.log(a); //"10"
      console.log(b); //10
      console.log(c); //"10ab"
      console.log(d); //Nan
      console.log(e); //"ab10"
      console.log(f); //Nan
      console.log(g); //"ab"
      console.log(h); //Nan
      console.log(i); //[empty]
      console.log(j); //0
      console.log(k); //[space]
      console.log(l); //10
      console.log(typeof a);
      console.log(typeof b);
      console.log(typeof c);
      console.log(typeof d);
      console.log(typeof e);
      console.log(typeof f);
      console.log(typeof g);
      console.log(typeof h);
      console.log(typeof i);
      console.log(typeof j);
```

HARSHA

```
        console.log(typeof k);
        console.log(typeof l);
      </script>
    </body>
  </html>
```

ParseInt Function

- The "parseInt()" is a pre-defined function, which is used to convert string to number.
- parseInt() doesn't supports decimal places.
- It returns number, if the string is alphanumerical that starts with number.
- It returns "NaN", if the string is alphanumerical that starts with alphabet.
- It returns "NaN", if the string is alphabetic.
- It returns "Nan", if the string is empty / space.

Syntax: parseInt(string value)

Example on parseInt Function

```
<html>
  <head>
    <title>ParseInt</title>
  </head>
  <body>
    <h1>ParseInt</h1>
    <script>
      var a = "10.72";
      var b = parseInt(a);
      var c = "10ab";
      var d = parseInt(c);
      var e = "ab10";
      var f = parseInt(e);
      var g = "ab";
      var h = parseInt(g);
      var i = "";
      var j = parseInt(i);
      var k = " ";
      var l = parseInt(k);
      console.log(a); //"10.72"
      console.log(b); //10
      console.log(c); //"10ab"
      console.log(d); //10
      console.log(e); //"ab10"
      console.log(f); //Nan
      console.log(g); //"ab"
      console.log(h); //10
      console.log(i); //[empty]
      console.log(j); //Nan
      console.log(k); //[space]
      console.log(l); //Nan
      console.log(typeof a);
```

```
        console.log(typeof b);
        console.log(typeof c);
        console.log(typeof d);
        console.log(typeof e);
        console.log(typeof f);
        console.log(typeof g);
        console.log(typeof h);
        console.log(typeof i);
        console.log(typeof j);
        console.log(typeof k);
        console.log(typeof l);
    </script>
  </body>
</html>
```

ParseFloat Function

- The "parseFloat()" is a pre-defined function, which is used to convert string to number.
- It is same as parseInt(); but it accepts decimal places.
- It returns number, if the string is alphanumerical that starts with number.
- It returns "NaN", if the string is alphanumerical that starts with alphabet.
- It returns "NaN", if the string is alphabetic.
- It returns "Nan", if the string is empty / space.

Syntax: parseFloat(string value)

Example on parseFloat Function

```
<html>
  <head>
    <title>ParseFloat</title>
  </head>
  <body>
    <h1>ParseFloat</h1>
    <script>
      var a = "10.72";
      var b = parseFloat(a);
      var c = "10ab";
      var d = parseFloat(c);
      var e = "ab10";
      var f = parseFloat(e);
      var g = "ab";
      var h = parseFloat(g);
      var i = "";
      var j = parseFloat(i);
      var k = " ";
      var l = parseFloat(k);
      console.log(a); //"10.72"
      console.log(b); //10.72
      console.log(c); //"10ab"
      console.log(d); //10
```

```
            console.log(e); //"ab10"
            console.log(f); //Nan
            console.log(g); //"ab"
            console.log(h); //10
            console.log(i); //[empty]
            console.log(j); //Nan
            console.log(k); //[space]
            console.log(l); //Nan
            console.log(typeof a);
            console.log(typeof b);
            console.log(typeof c);
            console.log(typeof d);
            console.log(typeof e);
            console.log(typeof f);
            console.log(typeof g);
            console.log(typeof h);
            console.log(typeof i);
            console.log(typeof j);
            console.log(typeof k);
            console.log(typeof l);
        </script>
    </body>
</html>
```

+ Unary Operator

- It is same as "Number()" function, but it supports only numbers.

Syntax: +

Example on + Unary Operator

```
<html>
    <head>
        <title>+</title>
    </head>
    <body>
        <h1>+</h1>
        <script>
            var a = "10.92";
            var b = +a;
            console.log(a);
            console.log(b);
            console.log(typeof a);
            console.log(typeof b);
        </script>
    </body>
</html>
```

toFixed() function

- It converts the number into string and adds the specified no. of decimal places.
- It also rounds the number.

Syntax: numbervalue.toFixed(no. of decimals)

Example on toFixed()

```
<html>
  <head>
    <title>toFixed</title>
  </head>
  <body>
    <h1>toFixed</h1>
    <script>
      var a = 10.86231;
      var b = a.toFixed(0);
      var c = a.toFixed(1);
      var d = a.toFixed(2);
      var e = a.toFixed(3);
      var f = a.toFixed(7);
      console.log(a);
      console.log(b);
      console.log(c);
      console.log(d);
      console.log(e);
      console.log(f);
      console.log(typeof a);
      console.log(typeof b);
      console.log(typeof c);
      console.log(typeof d);
      console.log(typeof e);
      console.log(typeof f);
    </script>
  </body>
</html>
```

String Functions

- String functions are used to perform manipulations on strings

Ex: Converting into uppercase.

toUpperCase( )

- This function converts the string value into uppercase and returns it.

Syntax: string.toUpperCase( )

Example: “hello”.toUpperCase( ) → “HELLO”

Example toUpperCase()

```
<html>
  <head>
    <title>toUpperCase</title>
  </head>
  <body>
```

HARSHA

```
    <h1>toUpperCase</h1>
    <script>
      var s1 = "Hyderabad";
      var s2 = s1.toUpperCase();
      console.log(s2);
    </script>
  </body>
</html>
```

toLowerCase( )

- This function converts the string value into lowercase and returns it.

Syntax: string.toLowerCase( )

Example: "HELLO".toLowerCase( ) → "hello"

Example on toLowerCase()

```
<html>
  <head>
    <title>toLowerCase</title>
  </head>
  <body>
    <h1>toLowerCase</h1>
    <script>
      var s1 = "Hyderabad";
      var s2 = s1.toLowerCase();
      console.log(s2);
    </script>
  </body>
</html>
```

length

- This property returns the no. of characters in the string.

Syntax: string.length

Example: "hello".length → 5

Example on length

```
<html>
  <head>
    <title>length</title>
  </head>
  <body>
    <h1>length</h1>
    <script>
      var s1 = "Hyderabad";
      var n = s1.length;
      console.log(n);
    </script>
  </body>
```

```
</html>
```

charAt( )

- This function returns the single character present at the specified index.
- Index starts from 0 (zero).

Syntax: string.charAt( )

Example: "hello".charAt(1) → e

Example on charAt()

```
<html>
  <head>
    <title>charAt</title>
  </head>
  <body>
    <h1>charAt</h1>
    <script>
      var s1 = "Hyderabad";
      var ch = s1.charAt(5);
      console.log(ch);
    </script>
  </body>
</html>
```

charCodeAt( )

- This function returns the ASCII value of the single character present at the specified index.

Syntax: string.charCodeAt( )

Example: "hello".charCodeAt(1) → 101

Example on charCodeAt()

```
<html>
  <head>
    <title>charCodeAt</title>
  </head>
  <body>
    <h1>charCodeAt</h1>
    <script>
      var s1 = "Hyderabad";
      var n = s1.charCodeAt(5);
      console.log(n);
    </script>
  </body>
</html>
```

substr( )

- This function returns a part of the string, starting from specified index, upto the specified length of the string.

Syntax: string.substr(index, length)

Example: “hyderabad”.substr(2, 4) → dera

Example on substr()

```
<html>
  <head>
    <title>Substr</title>
  </head>
  <body>
    <h1>Substr</h1>
    <script>
      var s1 = "Hyderabad";
      var n1 = 2;
      var n2 = 4;
      var s2 = s1.substr(n1, n2);
      console.log(s2); //dera
    </script>
  </body>
</html>
```

indexOf( )

- This function searches for the given sub string in the string, and returns the index of the first character in the given string if it is found; it returns -1, if the character is not found.
- In case if you specify the start index, searching starts from the specified startindex.

Syntax: string.indexOf(“character”, startindex)

Example: “hyderabad”.indexOf(“a”) → 5

Example: “hyderabad”.indexOf(“era”) → 3

Example: “hyderabad”.indexOf(“z”) → -1

- This function always considers the first occurrence only.

Example on indexOf()

```
<html>
  <head>
    <title>indexOf</title>
  </head>
  <body>
    <h1>indexOf</h1>
    <script>
      var s1 = "Hyderabad";
      var n = s1.indexOf("r");
      console.log(n);
    </script>
  </body>
</html>
```

Example on indexOf() – second occurrence

```
<html>
  <head>
    <title>indexOf - second occurrence</title>
  </head>
  <body>
    <h1>indexOf - second occurrence</h1>
    <script>
      var s1 = "Hyderabad";
      var n = s1.indexOf("a");
      var n2 = s1.indexOf("a", n + 1);
      console.log(n2);
    </script>
  </body>
</html>
```

Example indexOf() with -1

```
<html>
  <head>
    <title>indexOf with -1</title>
  </head>
  <body>
    <h1>indexOf - with -1</h1>
    <script>
      var s1 = "Hyderabad";
      var n = s1.indexOf("q");
      console.log(n);
    </script>
  </body>
</html>
```

replace( )

- It replaces a “word” with “another word”.

Syntax: string.replace(“old word”, “new word”)

Example: “MS Office”.replace(“Office”, “Windows”) → MS Windows

Example on replace()

```
<html>
  <head>
    <title>Replace</title>
  </head>
  <body>
    <h1>Replace</h1>
    <script>
      var s1 = "Hyderabad is one of the cities in India";
      var s2 = "Hyderabad";
      var s3 = "Chennai";
      var s4 = s1.replace(s2, s3);
      console.log(s4); //Chennai is one of the cities in India
```

HARSHA

```
    </script>
  </body>
</html>
```

split( )

- This function converts a string into an array of many small strings and returns the array, based on the separator character.

Syntax: string.split("separator character")

Example: "how are you".split("") → ["how", "are", "you"]

Example on split()

```
<html>
  <head>
    <title>Split</title>
  </head>
  <body>
    <h1>Split</h1>
    <script>
      var s1 = "How are you";
      var a = s1.split(' ');
      for (var i = 0; i < a.length; i++)
      {
        console.log(a[i]);
      }
    </script>
  </body>
</html>
```

trim( )

- This function removes the unnecessary spaces at left side and right side of the string.

Syntax: string.trim( )

Example: " abc def ".trim( ) → "abc def"

Example on trim()

```
<html>
  <head>
    <title>Trim</title>
  </head>
  <body>
    <h1>Trim</h1>
    <script>
      var s1 = "      Hyderabad is one of the cities in India.    ";
      var s2 = s1.trim();
      console.log(s1);
      console.log(s2);
      var n1 = s1.length;
      var n2 = s2.length;
```

```
        console.log("Before trim: " + n1); //55
        console.log("After trim: " + n2); //40
      </script>
    </body>
  </html>
```

concat( )

- This function attaches two strings and make then as a sinle string.

Syntax: string.concat(“another string”)

Example: “united”.concat(“states”) → “unitedstates”

Example on concat()

```
<html>
  <head>
    <title>Concat</title>
  </head>
  <body>
    <h1>Concat</h1>
    <script>
      var s1 = "hydera";
      var s2 = "bad";
      var s3 = s1.concat(s2);
      console.log(s3); //hyderabad
    </script>
  </body>
</html>
```

Date Functions

- Date functions are used to manipulate date and time.

new Date( )

- This is used to get the current system date and time.

Syntax: new Date( )

Example: new Date( ) → Thu Aug 10 2017 11:04:51 GMT+0530 (India Standard Time)

Example on new Date()

```
<html>
  <head>
    <title>Date</title>
  </head>
  <body>
    <h1>Date</h1>
    <script>
      var d = new Date();
      console.log(d);
    </script>
```

```
    </body>
</html>
```

toLocaleDateString( )

- This function returns the date in the following format: M/d/yyyy

Syntax: new Date( ).toLocaleDateString( )

Example: new Date( ).toLocaleDateString( ) → 8/10/2017

Example on toLocaleDateString()

```
<html>
    <head>
        <title>toLocaleDateString</title>
    </head>
    <body>
        <h1>toLocaleDateString</h1>
        <script>
            var d = new Date();
            var s = d.toLocaleDateString();
            console.log(s);
        </script>
    </body>
</html>
```

toLocaleTimeString( )

- This function returns the time in the following format: hh:mi:ss am/pm

Syntax: new Date( ).toLocaleTimeString( )

Example: new Date( ).toLocaleTimeString( ) → 11:04:51 AM

Example on toLocaleTimeString()

```
<html>
    <head>
        <title>toLocaleTimeString</title>
    </head>
    <body>
        <h1>toLocaleTimeString</h1>
        <script>
            var d = new Date();
            var s = d.toLocaleTimeString();
            console.log(s);
        </script>
    </body>
</html>
```

getTime( )

- This function returns the no. of milli seconds since “1/1/1970 12:00:00 AM”.

Syntax: new Date( ).getTime( )

Example: new Date( ).getTime( ) → 1502343291481

Example on getTime()

```
<html>
    <head>
        <title>getTime</title>
    </head>
    <body>
        <h1>getTime</h1>
        <script>
            var d = new Date();
            var n = d.getTime(); //milliseconds since 01.01.1970 12:00:00 AM
            console.log(n);
        </script>
    </body>
</html>
```

getDay( )

- This function returns the no. of the day of the week.

  0 = Sunday

  1 = Monday

  2 = Tuesday

  3 = Wednesday

  4 = Thursday

  5 = Friday

  6 = Saturday

Syntax: new Date( ).getDay( )

Example: new Date( ).getDay( ) → 4

Example on getDay()

```
<html>
    <head>
        <title>getDay</title>
    </head>
    <body>
        <h1>getDay</h1>
        <script>
            var d = new Date();
            n = d.getDay();
            console.log(n);
        </script>
    </body>
</html>
```

HARSHA

getDate( )

- This function returns only the date.

Syntax: new Date( ).getDate( )

Example: new Date( ).getDate( ) → 10

Example on getDate()

```
<html>
    <head>
        <title>getDate</title>
    </head>
    <body>
        <script>
            var d = new Date();
            var s = d.getDate();
            console.log(s);
        </script>
    </body>
</html>
```

getMonth( )

- This function returns only the month (0 to 11).

Syntax: new Date( ).getMonth( )

Example: new Date( ).getMonth( ) → 7

Example on getMonth()

```
<html>
    <head>
        <title>getMonth</title>
    </head>
    <body>
        <script>
            var d = new Date();
            var s = d.getMonth();
            console.log(s);
        </script>
    </body>
</html>
```

getFullYear( )

- This function returns only the year.

Syntax: new Date( ).getFullYear( )

Example: new Date( ).getFullYear( ) → 2017

Example on getFullYear()

```
<html>
```

HARSHA

```
<head>
    <title>getFullYear</title>
</head>
<body>
    <h1>getFullYear</h1>
    <script>
        var d = new Date();
        var s = d.getFullYear();
        console.log(s);
    </script>
</body>
</html>
```

getHours( )

- This function returns only the hours (in 24 hours format).

Syntax: new Date( ).getHours( )

Example: new Date( ).getHours( ) → 11

Example on getHours()

```
<html>
    <head>
        <title>getHours</title>
    </head>
    <body>
        <script>
            var d = new Date();
            var s = d.getHours();
            console.log(s);
        </script>
    </body>
</html>
```

getMinutes( )

- This function returns only the minutes.

Syntax: new Date( ).getMinutes( )

Example: new Date( ).getMinutes( ) → 4

Example on getMinutes()

```
<html>
    <head>
        <title>getMinutes</title>
    </head>
    <body>
        <script>
            var d = new Date();
            var s = d.getMinutes();
```

```
            console.log(s);
        </script>
    </body>
</html>
```

getSeconds( )

- This function returns only the seconds.

Syntax: new Date( ).getSeconds( )

Example: new Date( ).getSeconds( ) → 51

Example on getSeconds()

```
<html>
    <head>
        <title>getSeconds</title>
    </head>
    <body>
        <script>
            var d = new Date();
            var s = d.getSeconds();
            console.log(s);
        </script>
    </body>
</html>
```

getMilliseconds( )

- This function returns only the milli seconds.

Syntax: new Date( ).getMiliseconds( )

Example: new Date( ).getMilliseconds( ) → 481

Example on getMilliseconds()

```
<html>
    <head>
        <title>getMilliSeconds</title>
    </head>
    <body>
        <script>
            var d = new Date();
            var s = d.getMilliseconds();
            console.log(s);
        </script>
    </body>
</html>
```

Creating a Custom Date:

- We can create custom (user-defined) date using the following steps:
- Create a date object and variable: var variable = new Date( );

- **Set year:** variable.setFullYear(year);
- **Set month:** variable.setMonth(month);

  Note: month is 0 to 11
- **Set date:** variable.setDate(date);

Example on Custom Date

```
<html>
    <head>
        <title>Custom date</title>
    </head>
    <body>
        <h1>Custom Date</h1>
        <script>
            var d = new Date();
            d.setFullYear(2018);
            d.setMonth(11); //0 to 11
            d.setDate(31);
            console.log(d.toLocaleDateString());
        </script>
    </body>
</html>
```

Advanced

NoScript

- It is used to display a message to the user, when the user has disabled JavaScript in the browser.

Syntax:

<noscript>message</noscript>

Examples on NoScript

```
<html>
  <head>
    <title>noscript</title>
  </head>
  <body>
    <script>
      console.log("<h1>Message from javascript</h1>");
    </script>

    <noscript>
      <h1>Please enable javascript</h1>
    </noscript>
  </body>
</html>
```

Clousers

- “Clousers” are used to create private variables that are accessible to only a set of methods.
- Create a function; Create private variables in the same function with "var" keyword; Return an object from this function; If you call the function, the private variables of this function are not accessible outside the function.

Syntax:

```
var functionname = function( )
{
   var variablename = value;
   return
  {
    method: function( )
    {
        code
    },
    method: function( )
    {
        code
    }
  };
};
var x = new functionname( );
//Private variables are not accessible using "x".
```

Examples on Clousers

```
<html>
  <head>
    <title>Clousers</title>
  </head>
  <body>
    <h1>Clousers</h1>
    <script>
      var Sample = function ()
      {
        var x = 0;
        return {
          increment: function ()
          {
            x++;
          },
          decrement: function ()
```

```
            {
               x--;
            },
            getValue: function ()
            {
               return x;
            }
         };
      };
      var s = new Sample();
      console.log(s.getValue()); //Output: 0
      s.increment();
      s.increment();
      console.log(s.getValue()); //Output: 2
      s.decrement();
      console.log(s.getValue()); //Output: 1
   </script>
  </body>
</html>
```

Hoisting

- JavaScript automatically lifts-up the variable declarations (that are created using "var" keyword) to top of the program or top of the function. This is called as “Hoisting”.

Note: Variable declarations cum initializations are not lifted-up.

Example on Hoisting

```
<html>
  <head>
    <title>Hoisting</title>
  </head>
  <body>
    <h1>Hoisting</h1>
    <script>
      x = 5;
      console.log(x);
      var x; //lifted-up
    </script>
  </body>
</html>
```

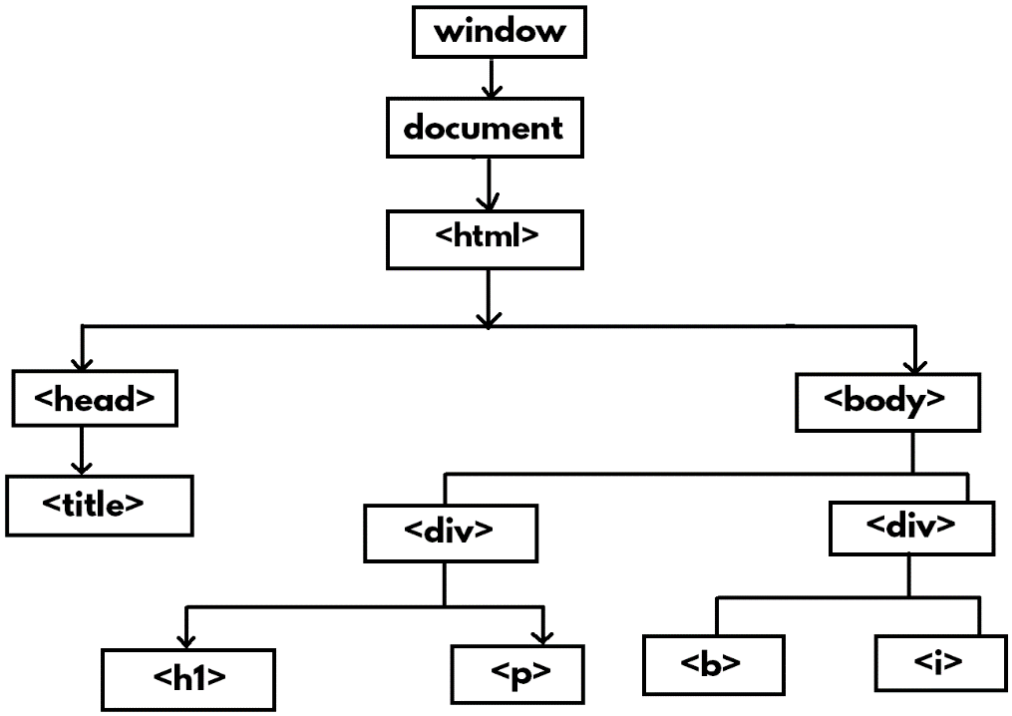
DOM

- DOM (Document Object Model) is the tree structure of html elements (tags) that are present within the web page.
- When the web page has been opened in the browser, DOM will be automatically created by the browser.
- The changes made to DOM are called as “DOM manipulations”. DOM manipulations are performed using JavaScript.

- The entire browser window is called as "window". The webpage running on the browser is called as "document". It has only one main element called <html>. It has two children <head> and <body>. There are many children for both <head> and <body>.

Example:

```
<html>
    <head>
        <title>DOM (Document Object Model)</title>
    </head>
    <body>
        <div>
            <h1>Heading</h1>
            <p>Paragraph</p>
        </div>
        <div>
            <b>bold</b>
            <i>italic</i>
        </div>
    </body>
</html>
```



Objects inside DOM

I. Window
II. Document
III. Element

I) Window

- "window" object represents the entire browser window.
- It has the following properties and methods:
  1. location.href
  2. navigator.userAgent
  3. screen
  4. screen
  5. alert( )
  6. confirm( )
  7. print( )
  8. setTimeout( )
  9. setInterval( )
  10. scrollTo
  11. open( )

1. location.href

- This property represents url of the current web page running in the browser window.
- **Syntax:** window.location.href

2. navigator.userAgent

- This property represents the name of current browser.
- **Syntax:** window.navigator.userAgent

3. navigator.screenX

- This property represents X value of current browser position on the screen.
- **Syntax:** window.navigator.screenX

4. navigator.screenY

- This property represents Y value of current browser position on the screen.
- **Syntax:** window.navigator.screenY

Example on Window Properties

```
<html>
  <head>
    <title>window</title>
  </head>
  <body>
```

```
    <h1>window</h1>
    <script>
      console.log(window.location.href);
      console.log(window.navigator.userAgent);
      console.log(window.screenX);
      console.log(window.screenY);
    </script>
  </body>
</html>
```

5. alert()

- This method displays an information dialogbox (message dialogbox) to the user.
- It contains only OK button.

Syntax: window.alert("message")

Example: window.alert("Hello");

Example on alert()

```
<html>
  <head>
    <title>alert</title>
  </head>
  <body>
    <h1>alert</h1>
    <script>
      window.alert("Hello");
    </script>
  </body>
</html>
```

6. confirm()

- This method displays an confirmation dialogbox to the user.
- It contains OK and Cancel buttons.
- It returns "true", if the user clicks on "OK" button; It returns "false", if the user clicks on "Cancel" button.

Syntax: window.confirm("message")

Example on confirm()

```
<html>
  <head>
    <title>confirm</title>
  </head>
  <body>
    <h1>confirm</h1>
    <script>
      var result = window.confirm("Are you sure to delete");
      console.log(result);
```

```
        </script>
    </body>
</html>
```

7. print()

- This method displays print dialogbox, which is used to print the current webpage through selected printer.
- In Google Chrome, it shows "Print Preview" also.

Syntax: window.print()

Example on print()

```
<html>
    <head>
        <title>Print</title>
    </head>
    <body>
        <h1>Print</h1>
        <p>Paragraph</p>
        <script>
        window.print();
        </script>
    </body>
</html>
```

8. setTimeout()

- This method calls the specified function, after completion of specified no. of milli seconds.
- **Note:** 1000 milli seconds = 1 second

Syntax: window.setTimeout(function, milli seconds)

Example on setTimeout()

```
<html>
    <head>
        <title>setTimeout</title>
    </head>
    <body>
        <h1>setTimeout</h1>
        <script>
            window.setTimeout(fun1, 5000);
            function fun1()
            {
                console.log("Hello");
            }
        </script>
    </body>
</html>
```

HARSHA

9. setInterval()

- This method calls the specified function repeatedly, for every completion of specified no. of milli seconds.
- **Note:** 1000 milli seconds = 1 second

  Syntax: window.setInterval(function, milli seconds)

Example on setInterval()

```
<html>
    <head>
        <title>setInterval</title>
    </head>
    <body>
        <h1>setInterval</h1>
        <script>
            window.setInterval(fun1, 1000);
            function fun1()
            {
                console.log("Hello");
            }
        </script>
    </body>
</html>
```

10. scrollTo()

- This method scrolls the web page horizontally / vertically to the specified X and Y co-ordinates.
- The X and Y co-ordinates are calculated in the form of pixels.

  Syntax: window.scrollTo(x, y)

Example on scrollTo()

```
<html>
    <head>
        <title>scrollTo</title>
    </head>
   <body>
      <h1>scrollTo</h1>
      <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
      <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
      <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
      <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
```

```
    <p>Our Services are very diverse, so sometimes additional terms or product requirements (including age requirements) may apply. Additional terms will be available with the relevant Services, and those additional terms become part of your agreement with us if you use those Services.</p>
    <p>Our Services are very diverse, so sometimes additional terms or product requirements (including age requirements) may apply. Additional terms will be available with the relevant Services, and those additional terms become part of your agreement with us if you use those Services.</p>
    <p>Our Services are very diverse, so sometimes additional terms or product requirements (including age requirements) may apply. Additional terms will be available with the relevant Services, and those additional terms become part of your agreement with us if you use those Services.</p>
    <script>
      window.scrollTo(0, 500);
    </script>
  </body>
</html>
```

11. open()

- This method opens a browser child window / popup window.
- It is mainly useful for opening ads.

Syntax: window.open("url", "logical name", "width=pixels, height=pixels");

Example on open()

open.html

```
<html>
  <head>
      <title>Open</title>
  </head>
  <body>
    <h1>Open</h1>
    <p>Click "Options" - "Allow Popups"</p>
    <script>
      window.open("mypage.html", "popup1", "width=300, height=300");
    </script>
  </body>
</html>
```

mypage.html

```
<html>
  <head>
      <title>Popup</title>
  </head>
  <body>
      Hello, World
  </body>
</html>
```

HARSHA

II) Document

- The "document" object represents the current working web page.
- It has properties and methods to manipulate web page.

1. title

- This property represents title of the web page.
- **Syntax:** document.title

2. head

- This property represents <head> tag of the web page.

Syntax: document.head

3. body

- This property represents <body> of the web page.

Syntax: document.body

4. images

- This property represents all images of the web page, as an array.

Syntax: document.images

5. links

- This property represents all hyperlinks (<a> tags) of the web page, as an array.

Syntax: document.links

6. URL

- This property represents url of the web page.

Syntax: document.URL

Example on Document Properties

```
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <h1>document</h1>
    <img src="img1.jpg" width="100px">
    <img src="img2.jpg" width="100px">
    <br>
    <a href="http://www.google.com">Google</a>
    <a href="http://www.facebook.com">Facebook</a>
    <script>
      console.log(document.title);
```

```
            console.log(document.head);
            console.log(document.body);
            console.log(document.images);
            console.log(document.links);
            console.log(document.URL);
        </script>
    </body>
</html>
```

document.write()

- This method displays the given message in the web page.

Syntax: document.write("message")

Example on document.write()

```
<html>
    <head>
        <title>document</title>
    </head>
    <body>
        <script>
            document.write("Hello World");
        </script>
    </body>
</html>
```

document.getElementById()

- This method retrieves the single element that has specified ID.

Syntax: document.getElementById("id")

Example on document.getElementById()

```
<html>
    <head>
        <title>Getting element by ID</title>
    </head>
    <body>
        <p id="abc">Hello</p>
        <script>
            console.log(document.getElementById("abc"));
        </script>
    </body>
</html>
```

document.getElementsByName()

- This method retrieves the array of elements that have specified name.

Syntax: document.getElementsByName("name")

HARSHA

Example on document.getElementsByName()

```
<html>
  <head>
    <title>getElementsByName</title>
  </head>
  <body>
    <h1>getElementsByName</h1>
    <p name="abc">Hello 1</p>
    <p name="abc">Hello 2</p>
    <p>Hai</p>
    <p name="abc">Hello 3</p>
    <script>
      console.log(document.getElementsByName("abc"));
    </script>
  </body>
</html>
```

document.getElementsByTagName()

- This method retrieves the array of elements that have specified tag name.

Syntax: document.getElementsByTagName("tag name")

Example on document.getElementsByTagName()

```
<html>
  <head>
    <title>getElementsByTagName</title>
  </head>
  <body>
    <h1>getElementsByTagName</h1>
    <p>Hello 1</p>
    <p>Hello 2</p>
    <p>Hello 3</p>
    <p>Hello 4</p>
    <script>
      console.log(document.getElementsByTagName("p"));
    </script>
  </body>
</html>
```

document.getElementsByClassName()

- This method retrieves the array of elements that have specified class name.

Syntax: document.getElementsByClassName("class name")

Example on document.getElementsByClassName()

```
<html>
  <head>
    <title>getElementsByClassName</title>
  </head>
  <body>
```

```
    <h1>getElementsByClassName</h1>
    <p class="abc">Hello 1</p>
    <p class="abc">Hello 2</p>
    <p>Hello</p>
    <p class="abc">Hello 2</p>
    <script>
      console.log(document.getElementsByClassName("abc"));
    </script>
  </body>
</html>
```

document.querySelectorAll()

- This method retrieves the array of elements that are matching with specified selector.
- You can use any CSS selectors:
  1. Tag Selector : tag
  2. ID Selector : #id
  3. Class Selector : .classname
  4. Grouping Selector : tag1,tag2,…
  5. Child Selector : parent child

Syntax: document.querySelectorAll("selector")

Example on document.querySelectorAll()

```
<html>
  <head>
    <title>querySelectorAll</title>
  </head>
  <body>
    <p class="abc">Hello 1</p>
    <p class="abc">Hello 2</p>
    <p class="abc">Hello 3</p>
    <script>
      console.log(document.querySelectorAll(".abc"));
    </script>
  </body>
</html>
```

document.querySelector()

- This method retrieves the first element that matches with specified selector.

Syntax: document.querySelector("selector")

Example on document.querySelector()

```
<html>
  <head>
    <title>querySelector</title>
```

```
    </head>
    <body>
        <p id="abc">Hello</p>
        <script>
        console.log(document.querySelector("#abc"));
        </script>
    </body>
</html>
```

III) Element

- The "element" object represents a single tag.
- It has properties and methods to manipulate the element.

tagName

- This property represents name of the tag.
- **Syntax:** document.getElementById("id").tagName

Example on tagName

```
<html>
  <head>
    <title>TagName</title>
  </head>
  <body>
    <div id="div1">Hello</div>
    <script>
      console.log(document.getElementById("div1").tagName);
    </script>
  </body>
</html>
```

id

- This property represents id of the tag.

Syntax: document.getElementById("id").id

Example on id

```
<html>
  <head>
    <title>ID</title>
  </head>
  <body>
    <div id="div1">Hello</div>
    <script>
      console.log(document.getElementById("div1").id);
    </script>
  </body>
</html>
```

innerHTML

- This property represents content of the tag.

Syntax: document.getElementById("id").innerHTML

Example on innerHTML

```
<html>
  <head>
    <title>innerHTML</title>
  </head>
  <body>
    <div id="div1">Hello</div>
    <script>
      console.log(document.getElementById("div1").innerHTML);
      document.getElementById("div1").innerHTML = "Hai";
    </script>
  </body>
</html>
```

innerText

- This property represents content of the tag, without tags.

Syntax: document.getElementById("id").innerText

Example on innerText

```
<html>
  <head>
    <title>innerText</title>
  </head>
  <body>
    <div id="div1">Hello <b>World</b></div>
    <script>
      console.log(document.getElementById("div1").innerHTML);
      console.log(document.getElementById("div1").innerText);
    </script>
  </body>
</html>
```

style

- This property represents css style of the tag.

Syntax: document.getElementById("id").style.property

Example on style

```
<html>
  <head>
    <title>Style</title>
  </head>
  <body>
```

```
    <div id="div1">div 1</div>
    <script>
      document.getElementById("div1").style.fontFamily = "Comic Sans MS";
      document.getElementById("div1").style.fontSize = "50px";
      document.getElementById("div1").style.fontWeight = "bold";
      document.getElementById("div1").style.fontStyle = "italic";
      document.getElementById("div1").style.width = "500px";
      document.getElementById("div1").style.height = "200px";
      document.getElementById("div1").style.backgroundColor = "#006633";
      document.getElementById("div1").style.color = "#ccffff";
      document.getElementById("div1").style.border = "5px double red";
      document.getElementById("div1").style.paddding = "20px";
      console.log(document.getElementById("div1").style.fontFamily);
    </script>
  </body>
</html>
```

parentElement

- This property represents parent element of the tag.
- **Syntax:** document.getElementById("id").parentElement

Example on parentElement

```
<html>
  <head>
    <title>ParentElement</title>
  </head>
  <body>
    <div>
      <p id="p1">Hello</p>
    </div>
    <script>
      console.log(document.getElementById("p1").parentElement);
    </script>
  </body>
</html>
```

children

- This property represents child elements of the tag.

Syntax: document.getElementById("id").children

Example on children

```
<html>
  <head>
    <title>Children</title>
  </head>
  <body>
    <div id="div1">
      <p id="p1">Para 1</p>
```

```
            <p id="p2">Para 2</p>
            <p id="p3">Para 3</p>
        </div>
        <script>
            console.log(document.getElementById("div1").children);
        </script>
    </body>
</html>
```

scrollTop

- This property moves vertical scrollbar, based on the specified no. of pixels.

Syntax: document.getElementById("id").scrollTop

Example on scrollTop

```
<html>
    <head>
        <title>scrollTop</title>
        <style>
            #div1
            {
                background-color: skyblue;
                width: 400px;
                height: 400px;
                overflow: auto;
            }
        </style>
    </head>
    <body>
        <h1>scrollTop</h1>
        <div id="div1">
            <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
            <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
            <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
            <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
            <p>Our Services are very diverse, so sometimes additional terms or product requirements
(including age requirements) may apply. Additional terms will be available with the relevant Services,
and those additional terms become part of your agreement with us if you use those Services.</p>
        </div>
        <script>
            document.getElementById("div1").scrollTop = 300;
        </script>
    </body>
```

```
</html>
```

setAttribute()

- This method sets an attribute to the tag.

Syntax: document.getElementById("id").setAttribute("attribute name", "value")

Example on setAttribute()

```
<html>
  <head>
    <title>setAttribute</title>
  </head>
  <body>
    <img src="img1.jpg" id="myimage" width="300px">
    <script>
      document.getElementById("myimage").setAttribute("src", "img2.jpg");
    </script>
  </body>
</html>
```

Note: Place "img1.jpg" and "img2.jpg" in the current folder.

getAttribute()

- This method gets the value of specified attribute of the tag.
- **Syntax:** document.getElementById("id").getAttribute("attribute name")

Example on getAttribute()

```
<html>
  <head>
    <title>getAttribute</title>
  </head>
  <body>
    <img src="img1.jpg" id="myimage" width="300px">
    <script>
      var a = document.getElementById("myimage").getAttribute("src");
      console.log(a);
    </script>
  </body>
</html>
```

Note: Place "img1.jpg" in the current folder.

removeAttribute()

- This method removes the specified attribute of the tag.

Syntax: document.getElementById("id").removeAttribute("attribute name")

Example on removeAttribute()

```
<html>
  <head>
    <title>removeAttribute</title>
  </head>
  <body>
    <img src="img1.jpg" id="myimage" width="300px" title="title here">
    <script>
      document.getElementById("myimage").removeAttribute("title");
    </script>
  </body>
</html>
```

Note: Place "img1.jpg" in the current folder.

attributes

- This property retrieves all the attributes of the tag, along with values.

Syntax: document.getElementById("id").attributes

Example on attributes

```
<html>
  <head>
    <title>Attributes</title>
  </head>
  <body>
    <img src="img1.jpg" id="myimage" width="200px" title="hello">
    <script>
      var a = document.getElementById("myimage").attributes;
      console.log(a);
      console.log(a[0]);
      console.log(a[1]);
      console.log(a[2]);
      console.log(a[0].name);
      console.log(a[1].name);
      console.log(a[2].name);
      console.log(a[0].value);
      console.log(a[1].value);
      console.log(a[2].value);
    </script>
  </body>
</html>
```

Note: Place "img1.jpg" in the current folder.

hasAttribute()

- This method checks whether the element has specified attribute or not; returns "true" if it contains; returns "false" if it doesn't contain.

Syntax: document.getElementById("id").hasAttribute("attribute name");

Example on hasAttribute()

```
<html>
  <head>
    <title>hasAttribute</title>
  </head>
  <body>
    <img src="img1.jpg" width="200px" id="myimage">
    <script>
      console.log(document.getElementById("myimage").hasAttribute("src"));
    </script>
  </body>
</html>
```

Note: Place "img1.jpg" in the current folder.

focus()

- This method places the cursor inside the element.

Syntax: document.getElementById("id").focus( )

Example on focus()

```
<html>
  <head>
    <title>Focus</title>
  </head>
  <body>
    Firstname: <input type="text" id="txt1"><br>
    Lastname: <input type="text" id="txt2"><br>
    <script>
      document.getElementById("txt1").focus();
    </script>
  </body>
</html>
```

click()

- This method clicks the specified element (equal to manual mouse click).
- **Syntax:** document.getElementById("id").click( )

Example on click()

```
<html>
  <head>
    <title>Click</title>
  </head>
  <body>
    <form action="http://localhost:8080">
      Firstname: <input type="text" name="firstname" value="abc"><br>
      Lastname: <input type="text" name="lastname" value="xyz"><br>
      <input type="submit" value="Submit" id="button1">
    </form>
```

```
        <script>
          document.getElementById("button1").click();
        </script>
      </body>
    </html>
```

remove()

- This method removes the current element.

Syntax: document.getElementById("id").remove( )

Example on remove()

```
<html>
    <head>
        <title>Remove</title>
    </head>
    <body>
        <p>para 1</p>
        <p id="p2">para 2</p>
        <p>para 3</p>
        <script>
        document.getElementById("p2").remove();
        </script>
    </body>
</html>
```

document.createElement()

- This method creates a new element for the specified tag.

Syntax: document.createElement("tag name")

appendChild()

- This method adds new child element to the current element.

Syntax: document.appendChild(newelement)

Example on createElemet( ) and appendChild()

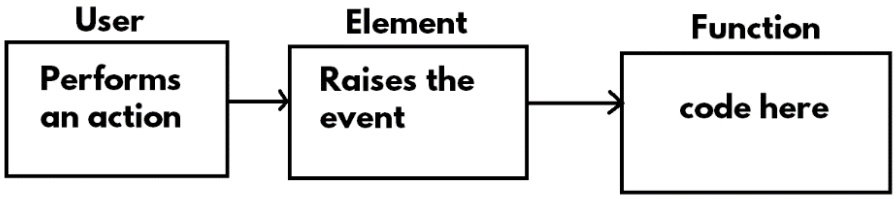
```
<html>
    <head>
        <title>AppendChild</title>
    </head>
    <body>
        <div id="div1">
            <p>para 1</p>
            <p>para 2</p>
            <p>para 3</p>
        </div>
        <script>
            var mypara = document.createElement("p");
            mypara.innerHTML = "para " + n;
```

```
            mypara.setAttribute("id", "p" + n);
            document.getElementById("div1").appendChild(mypara);
        </script>
    </body>
</html>
```

addEventListener()

Introduction to Events

- "Event" is a keyboard / mouse action, that is performed by the user, at run time.
- "Event Handling" is a concept of attaching the event with a function.
- Whenever the user performs an action, automatically the element raises the event; then we can call a function.



- **Syntax:** addEventListener("event name", functionname)
- **Example:** addEventListener("click", fun1)

List of Events

1. click
2. dblclick
3. mouseover
4. mouseout
5. mousemove
6. keyup
7. keypress
8. focus
9. blur
10. change
11. contextmenu
12. cut
13. copy
14. paste

"Click" event

- The "click" event executes when the user clicks on the element.

Syntax:

addEventListener("click", functionname);

function functionname( )

{

}

Example on "Click" event

```
<html>
  <head>
    <title>click</title>
    <style>
      #div1
      {
        background-color: skyblue;
        width: 200px;
        height: 200px;
      }
    </style>
  </head>
  <body>
    <h1>click</h1>
    <div id="div1">click me</div>
    <script>
      document.getElementById("div1").addEventListener("click", fun1);
      function fun1()
      {
        document.getElementById("div1").innerHTML = "thanx";
      }
    </script>
  </body>
</html>
```

"Dblclick" event

- The "dblclick" event executes when the user double clicks on the element.

Syntax:

addEventListener("dblclick", functionname);

function functionname( )

{

}

Example on "Dblclick" event

```
<html>
```

```
<head>
  <title>dblclick</title>
  <style>
    #div1
    {
      background-color: skyblue;
      width: 200px;
      height: 200px;
    }
  </style>
</head>
<body>
  <h1>dblclick</h1>
  <div id="div1">double click me</div>
  <script>
    document.getElementById("div1").addEventListener("dblclick", fun1);
    function fun1()
    {
      document.getElementById("div1").innerHTML = "thanx";
    }
  </script>
</body>
</html>
```

"Mouseover" and "Mouseout" events

"Mouseover" event

- The "mouseover" event executes when the user moves the mouse pointer from outside to inside the element.

  Syntax:

  addEventListener("mouseover", functionname);

  function functionname( )

  {

  }

"Mouseout" event

- The "mouseout" event executes when the user moves the mouse pointer from inside to outside the element.
- **Syntax:**

  addEventListener("mouseout", functionname);

  function functionname( )

  {

  }

Example on “Mouseover” and “Mouseout” events

```
<html>
  <head>
    <title>mouseover, mouseout</title>
    <style>
      #div1
      {
        width: 400px;
        height: 200px;
        background-color: lightgreen;
      }
    </style>
  </head>
  <body>
    <h1>mouseover, mouseout</h1>
    <div id="div1">
      mouseover me
    </div>
    <script>
      document.getElementById("div1").addEventListener("mouseover", fun1);
      function fun1()
      {
        document.getElementById("div1").innerHTML = "thanx";
      }

      document.getElementById("div1").addEventListener("mouseout", fun2);
      function fun2()
      {
        document.getElementById("div1").innerHTML = "mouseover me";
      }
    </script>
  </body>
</html>
```

“Mousemove” event

- The “mousemove” event executes when the user moves the mouse pointer across the element.

Syntax:

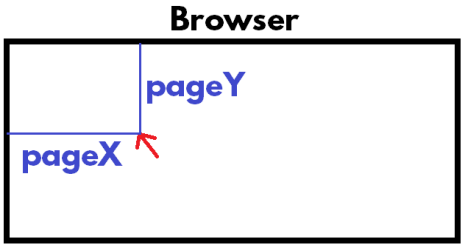
```
addEventListener(“mousemove”, functionname);
function functionname( )
{
}
```

event

Represents the information, given by the browser. Whenever the user performs an action, the browser collects some information, and it passes the same information to the function automatically. This is called “event”.

event.pageX: Represents “X” co-ordinate of mouse pointer position.

event.pageY: Represents "Y" co-ordinate of mouse pointer position.



Example on "Mousemove" event

```
<html>
  <head>
    <title>mousemove</title>
    <style>
      #div1
      {
        width: 400px;
        height: 200px;
        background-color: lightgreen;
      }
    </style>
  </head>
  <body>
    <h1>mousemove</h1>
    <div id="div1">
      mouseover me
    </div>

    <script>
      document.getElementById("div1").addEventListener("mousemove", fun1);
      function fun1(event)
      {
        //event = browser given information
        var x = event.pageX;
        var y = event.pageY;
        console.log(x + ", " + y);
      }
    </script>
  </body>
</html>
```

"Keyup" event

- The "keyup" event executes when the user presses any key on the keyboard, while the cursor is inside the element.

Syntax:

addEventListener("keyup", functionname);

function functionname( )

{

}

Example on "Keyup" event

```
<html>
  <head>
    <title>Keyup</title>
  </head>
  <body>
    <h1>Keyup</h1>
    <input type="text" id="txt1">
    <script>
      document.getElementById("txt1").addEventListener("keyup", fun1);
      function fun1()
      {
        console.log(document.getElementById("txt1").value);
      }
    </script>
  </body>
</html>
```

"Keypress" event

- The "keypress" event executes when the user presses any key on the keyboard, while the cursor is inside the element.
- "Keypress" event is very similar to "keyup" event.
- When you press any key on the keyboard, the following process happens:
    1. "Keypress" event executes.
    2. The character will be added in the textbox.
    3. "Keyup" event executes.
- "Keypress" event executes before accepting the currently pressed character into the element. "Keyup" event executes after accepting the currently pressed character into the element.
- In "keypress" event, we can accept / reject the currently pressed character, because it executes "before accepting" the character.
- In "keyup" event, we can't reject the currently pressed character, because it executes "after accepting" the character.

Syntax:

addEventListener("keypress", functionname);

function functionname( )

{

}

- **event.which:** Represents the ASCII value of currently pressed character. That means when the user presses a character, the browser automatically identifies its ASCII value and passes the same to the function, as “event.which”.
  - ASCII = American Standard Code for Information Interchange
  - As per ASCII, every character has a number.
    - 65 to 90 : A-Z
    - 97 to 122 : a-z
    - 48 to 57 : 0-9
    - 32 : Space
    - 8 : Backspace
    - 9 : TAB
    - 13 : Enter

- **event.preventDefault( ):** It stops the default functionality. That means it rejects the currently pressed character. If this method is not used, by default it accepts the currently pressed character.

Example on “Keypress” event

```
<html>
  <head>
    <title>Keypress</title>
  </head>
  <body>
    <h1>Keypress</h1>
    <input type="text" id="txt1">
    <script>
      document.getElementById("txt1").addEventListener("keypress", fun1);
      function fun1()
      {
        console.log(document.getElementById("txt1").value);
      }
    </script>
  </body>
</html>
```

Example on “Keypress” event – Alphabets only

```
<html>
  <head>
    <title>Alphabets only</title>
  </head>
  <body>
    <h1>Alphabets only</h1>
    <input type="text" id="txt1">
```

```
    <script>
      document.getElementById("txt1").addEventListener("keypress", fun1);
      function fun1(event)
      {
        var ch = event.which;
        console.log(ch);
        if (!((ch >= 65 && ch <= 90) || (ch >= 97 && ch <= 122) || (ch == 32) || (ch == 8) || (ch == 0)))
        {
          event.preventDefault();
        }
      }
    </script>
  </body>
</html>
```

Example on "Keypress" event – Numbers only

```
<html>
  <head>
    <title>Numbers only</title>
  </head>
  <body>
    <h1>Numbers only</h1>
    <input type="text" id="txt1">
    <script>
      document.getElementById("txt1").addEventListener("keypress", fun1);
      function fun1(event)
      {
        var ch = event.which;
        console.log(ch);
        if (!((ch >= 48 && ch <= 57) || (ch == 8) || (ch == 0)))
        {
          event.preventDefault();
        }
      }
    </script>
  </body>
</html>
```

"Focus" and "Blur" events

"Focus" event

- The "focus" event executes when the cursor enters into the element.

Syntax:

```
addEventListener("focus", functionname);
function functionname( )
{
}
```

"Blur" event

- The "blur" event executes when the cursor goes out of the element.
- **Syntax:**

```
addEventListener("blur", functionname);
function functionname( )
{
}
```

Example on "Focus" and "Blur" events

```
<html>
  <head>
    <title>focus and blur</title>
  </head>
  <body>
    <h1>focus and blur</h1>
    Email:
    <input type="text" id="txt1">
    <span id="span1" style="color:gray; display:none">Use gmail only</span>
    <script>
      document.getElementById("txt1").addEventListener("focus", fun1);
      function fun1()
      {
        document.getElementById("span1").style.display = "inline";
      }

      document.getElementById("txt1").addEventListener("blur", fun2);
      function fun2()
      {
        document.getElementById("span1").style.display = "none";
      }
    </script>
  </body>
</html>
```

"Change" event

- The "change" event executes when the value of the element has been changed.
- That means it executes in following cases:
  1. When the user modifies the value of textbox and presses TAB key.
  2. When the user checks / unchecks the checkbox.
  3. When the user selects the radio button.
  4. When the user selects an item in the dropdownlist.

Syntax:

```
addEventListener("change", functionname);
function functionname( )
```

```
{
}
```

Example on “Change” event with TextBox

```
<html>
  <head>
    <title>Change - TextBox</title>
  </head>
  <body>
    <h1>Change - TextBox</h1>
    Source Text:
    <input type="text" id="txt1">
    <br>
    Destination Text:
    <input type="text" id="txt2">
    <script>
      document.getElementById("txt1").addEventListener("change", fun1);
      function fun1()
      {
        document.getElementById("txt2").value = document.getElementById("txt1").value;
      }
    </script>
  </body>
</html>
```

Example on “Change” event with CheckBox

```
<html>
  <head>
    <title>Change - CheckBox</title>
  </head>
  <body>
    <h1>Change - CheckBox</h1>
    <input type="checkbox" id="chk1">
    <label for="chk1">I accept license agreement</label><br>
    <input type="submit" id="btn1" disabled="disabled">

    <script>
      document.getElementById("chk1").addEventListener("change", fun1);
      function fun1()
      {
        var b = document.getElementById("chk1").checked;
        if (b == true)
        {
          document.getElementById("btn1").disabled = "";
        }
        else
        {
          document.getElementById("btn1").disabled = "disabled";
        }
      }
```

```
    </script>
  </body>
</html>
```

Example on “Change” event with RadioButton

```
<html>
  <head>
    <title>Change - RadioButton</title>
    <style>
      #div1
      {
        color: darkblue;
      }
    </style>
  </head>
  <body>
    <h1>Change - RadioButton</h1>
    <form>
      <input type="radio" id="rb1" name="group1" checked="checked">
      <label for="rb1">Small</label>
      <input type="radio" id="rb2" name="group1">
      <label for="rb2">Medium</label>
      <input type="radio" id="rb3" name="group1">
      <label for="rb3">Large</label>
      <br>
      <div id="div1" style="font-size:20px;">
        Hello, World
      </div>
    </form>

    <script>
      document.getElementById("rb1").addEventListener("change", fun1);
      document.getElementById("rb2").addEventListener("change", fun1);
      document.getElementById("rb3").addEventListener("change", fun1);
      function fun1()
      {
        if (document.getElementById("rb1").checked == true)
          document.getElementById("div1").style.fontSize = "20px"; //small
        else if (document.getElementById("rb2").checked == true)
          document.getElementById("div1").style.fontSize = "35px"; //medium
        else if (document.getElementById("rb3").checked == true)
          document.getElementById("div1").style.fontSize = "50px"; //large
      }
    </script>
  </body>
</html>
```

Example on “Change” event with DropDownList

```
<html>
  <head>
    <title>Change - DropDownList</title>
```

```
</head>
<body>
  <h1>Change - DropDownList</h1>
  <select id="drp1">
    <option>Choose Country</option>
    <option>India</option>
    <option>UK</option>
    <option>US</option>
  </select>
  <br><br>
  You selected: <span id="span1"></span>
  <script>
    document.getElementById("drp1").addEventListener("change", fun1);
    function fun1()
    {
      document.getElementById("span1").innerHTML = document.getElementById("drp1").value;
    }
  </script>
</body>
</html>
```

"Contextmenu" event

- The "contextmenu" event executes when the user right clicks on an element.

Syntax:

addEventListener("contextmenu", functionname);

function functionname( )

{

}

event.preventDefault(): It disables the right click menu (context menu).

Example on "Contextmenu" event

```
<html>
  <head>
    <title>Disabling right click</title>
  </head>
  <body>
    <h1>Right click disabled</h1>
    <script>
      window.addEventListener("contextmenu", fun1);
      function fun1(event)
      {
        event.preventDefault();
        alert("right click not allowed");
      }
    </script>
  </body>
</html>
```

"Cut", "Copy", "Paste" events

"Cut" event

- The "cut" event executes when the user selects "cut" option with keyboard / mouse.

Syntax:

```
addEventListener("cut", functionname);
function functionname( )
{
}
```

- **event.preventDefault( ):** It disables the cut operation.

"Copy" event

- The "copy" event executes when the user selects "copy" option with keyboard / mouse.

Syntax:

```
addEventListener("copy", functionname);
function functionname( )
{
}
```

- **event.preventDefault( ):** It disables the copy operation.

"Paste" event

- The "paste" event executes when the user selects "paste" option with keyboard / mouse.

Syntax:

```
addEventListener("paste", functionname);
function functionname( )
{
}
```

event.preventDefault( ): It disables the paste operation.

Example on "Cut", "Copy", "Paste" events

```
<html>
  <head>
    <title>Cut, copy, paste</title>
  </head>
  <body>
    <h1>Cut, copy, paste disabled</h1>
```

HARSHA

```
    <input type="text">
    <input type="text">
    <input type="text">
    <script>
      window.addEventListener("cut", fun1);
      window.addEventListener("copy", fun1);
      window.addEventListener("paste", fun1);
      function fun1(event)
      {
        event.preventDefault();
        alert("cut copy paste not allowed");
      }
    </script>
  </body>
</html>
```

"this" keyword

- The "this" keyword represents the "current element", which has raised the event.

Example on "this" keyword

```
<html>
  <head>
    <title>this</title>
  </head>
  <body>
    <h1>this</h1>
    <input type="button" id="button1" value="click me">
    <input type="button" id="button2" value="click me">
    <input type="button" id="button3" value="click me">
    <input type="button" id="button4" value="click me">
    <input type="button" id="button5" value="click me">
    <script>
      document.getElementById("button1").addEventListener("click", fun1);
      document.getElementById("button2").addEventListener("click", fun1);
      document.getElementById("button3").addEventListener("click", fun1);
      document.getElementById("button4").addEventListener("click", fun1);
      document.getElementById("button5").addEventListener("click", fun1);
      function fun1()
      {
        console.log(this);
        this.setAttribute("value", "thanx");
        this.style.backgroundColor = "skyblue";
      }
    </script>
  </body>
</html>
```

Login - Example

```
<html>
  <head>
```

```
    <title>Login</title>
  </head>
  <body>
    <h1>Login</h1>
    <form>
      Username: <input type="text" id="txt1"><br>
      Password: <input type="password" id="txt2"><br>
      <input type="submit" id="button1" value="Login"><br>
      <span id="span1"></span>
    </form>
    <script>
      document.getElementById("button1").addEventListener("click", fun1);
      function fun1(event)
      {
        event.preventDefault();
        var username = document.getElementById("txt1").value;
        var password = document.getElementById("txt2").value;
        if (username == "admin" && password == "manager")
        {
          document.getElementById("span1").innerHTML = "Successful login";
        }
        else
        {
          document.getElementById("span1").innerHTML = "Invalid login";
        }
      }
    </script>
  </body>
</html>
```

Add, Subtract, Multiply, Divide Example

```
<html>
  <head>
    <title>Add Subtract Multiply Divide</title>
    <style>
      #txt3
      {
        background-color: lightgray;
      }

      #button1, #button2, #button3, #button4
      {
        background-color: #ffcc99;
        border: 1px ridge red;
      }
    </style>
  </head>
  <body>
    <h1>Add Subtract Multiply Divide</h1>
    <form>
      Enter value for a:
      <input type="text" id="txt1"><br>
```

```
        Enter value for b:
        <input type="text" id="txt2"><br>
        <input type="button" id="button1" value="Add">
        <input type="button" id="button2" value="Subtract">
        <input type="button" id="button3" value="Multiply">
        <input type="button" id="button4" value="Divide"><br>
        Result:
        <input type="text" id="txt3" readonly="readonly">
    </form>

    <script>
        document.getElementById("button1").addEventListener("click", fun1);
        function fun1()
        {
            var a = parseInt(document.getElementById("txt1").value);
            var b = parseInt(document.getElementById("txt2").value);
            var c = a + b;
            document.getElementById("txt3").value = c;
        }

        document.getElementById("button2").addEventListener("click", fun2);
        function fun2()
        {
            var a = parseInt(document.getElementById("txt1").value);
            var b = parseInt(document.getElementById("txt2").value);
            var c = a - b;
            document.getElementById("txt3").value = c;
        }

        document.getElementById("button3").addEventListener("click", fun3);
        function fun3()
        {
            var a = parseInt(document.getElementById("txt1").value);
            var b = parseInt(document.getElementById("txt2").value);
            var c = a * b;
            document.getElementById("txt3").value = c;
        }

        document.getElementById("button4").addEventListener("click", fun4);
        function fun4()
        {
            var a = parseInt(document.getElementById("txt1").value);
            var b = parseInt(document.getElementById("txt2").value);
            var c = a / b;
            document.getElementById("txt3").value = c;
        }
    </script>
  </body>
</html>
```

Validations

- Validation is a process of checking the form input values, whether those are correct or not.
- If all the form input values are correct, then we will allow the form to be submitted to the server.
- If any one of the form input values are incorrect, then we will stop submitting the form and display appropriate error message to the user.
- Validations are done using JavaScript.
- **Common Examples of Validations:**
  1. The value can't be blank.
  2. The value should be in the proper format. Ex: phone number, email etc.
  3. The value should be within the given range (minimum and maximum).

     Ex: Amount should be in between 1000 and 10000 etc.

Syntax for Validations

```
document.getElementById("form id").addEventListener("submit", functionname);
function functionname(event)
{
  if (condition)
  {
    //show error message
      event.preventDefault( );
  }
  else
  {
  //hide error message
  }
}
```

Example on Validations

```
<html>
  <head>
    <title>Validations</title>
    <style>
      .error
      {
        color: red;
      }
    </style>
  </head>
  <body>
    <h1>Validations</h1>
    <form id="f1">
```

```
        Username:
        <input type="text" id="txtusername"> *
        <span id="spanusername" class="error"></span><br>
        Password:
        <input type="password" id="txtpassword"> *
        <span id="spanpassword" class="error"></span><br>
        <input type="submit" value="Login">
    </form>

    <script>
        document.getElementById("f1").addEventListener("submit", fun1);
        function fun1(event)
        {
            //event = browser given information
            var s1 = document.getElementById("txtusername").value;
            if (s1 == "")
            {
                event.preventDefault();
                document.getElementById("spanusername").innerHTML = "Username can't be blank";
            }
            else
            {
                document.getElementById("spanusername").innerHTML = "";
            }

            var s2 = document.getElementById("txtpassword").value;
            if (s2 == "")
            {
                event.preventDefault();
                document.getElementById("spanpassword").innerHTML = "Password can't be blank";
            }
            else
            {
                document.getElementById("spanpassword").innerHTML = "";
            }
        }
    </script>
</body>
</html>
```

Regular Expressions

- Regular expression represents “pattern” of the value.
- Regular expressions are useful in validations, to check whether it is matching with the specified pattern or not.

List of Important Regular Expressions:

Sl. No	Description	Regular Expression
1	Digits only	^[0-9]*$
2	Alphabets only	^[a-zA-Z]*$
3	Indian Mobile Number	^[789]\d{9}$

4	Email	\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
5	Usernames: Alphabets, Digits and Hyphens only	([A-Za-z0-9-]+)
6	Passwords: 6 to 15 characters; at least one upper case letter, one lower case letter and one digit	((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,15})

Syntax to check Regular Expressions in JavaScript:

```
/regular expression/.test(value)
```

Example on Regular Expressions

```
<html>
  <head>
    <title>Regular Expressions</title>
    <style>
      .error
      {
        color: red;
      }
    </style>
  </head>
  <body>
    <h1>Regular Expressions</h1>
    <form id="f1" action="http://localhost/serverpage.aspx">
      Email:
      <input type="text" id="txtemail">
      <span id="spanemail" class="error"></span><br>
      Mobile:
      <input type="text" id="txtmobile">
      <span id="spanmobile" class="error"></span><br>
      <input type="submit" value="Login">
    </form>

    <script>
      document.getElementById("f1").addEventListener("submit", fun1);
      function fun1(event)
      {
        //event = browser given information
        var s1 = document.getElementById("txtemail").value;
        var regexpemail = /\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*/;
        if (regexpemail.test(s1) == false)
        {
          event.preventDefault();
          document.getElementById("spanemail").innerHTML = "Invalid email";
        }
        else
        {
          document.getElementById("spanemail").innerHTML = "";
        }
```

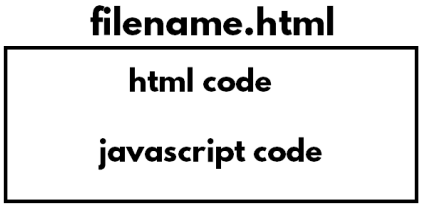
```
            var s2 = document.getElementById("txtmobile").value;
            var regexpmobile = /^[789]\d{9}$/;
            if (regexpmobile.test(s2) == false)
            {
               event.preventDefault();
               document.getElementById("spanmobile").innerHTML = "Invalid mobile";
            }
            else
            {
               document.getElementById("spanmobile").innerHTML = "";
            }
         }
      </script>
   </body>
</html>
```

Types of JavaScript

- JavaScript program can be created in two ways:
  1. Internal JavaScript / Embedded JavaScript
  2. External JavaScript

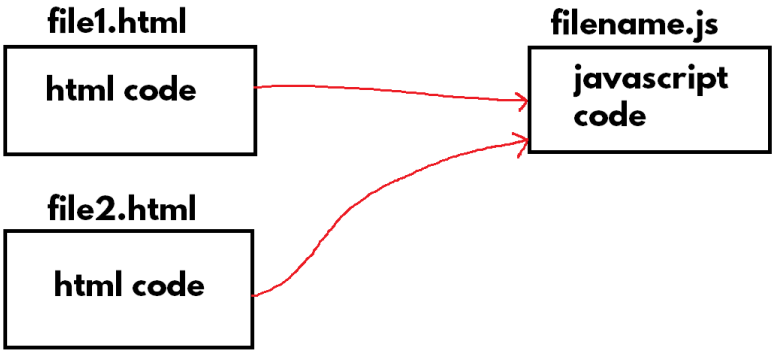
1. Internal JavaScript

- Both "html code" and "javascript code" will be written in the same ".html" file.
- **Advantage:** Easy to understand.
- **Disadvantage:** The javascript code that is written in ".html" file can't be used in another html file.
- All the previous programs are the examples of "Internal JavaScript".



2. External JavaScript

- The javascript code will be written in ".js" file and will be called in one or more ".html" files.
- **Advantage:** We can call the same ".js" file from many html files (re-usability).
- In realtime, external JavaScript is recommended.



Example on External JavaScript

JavaScript.js

```
document.write("<h1>Message from javascript</h1>");
```

Page1.html

```
<html>
 <head>
    <title>External JavaScript - Page 1</title>
 </head>
 <body>
    <h1>External JavaScript - Page 1</h1>
    <script src="JavaScript.js"></script>
 </body>
</html>
```

Page2.html

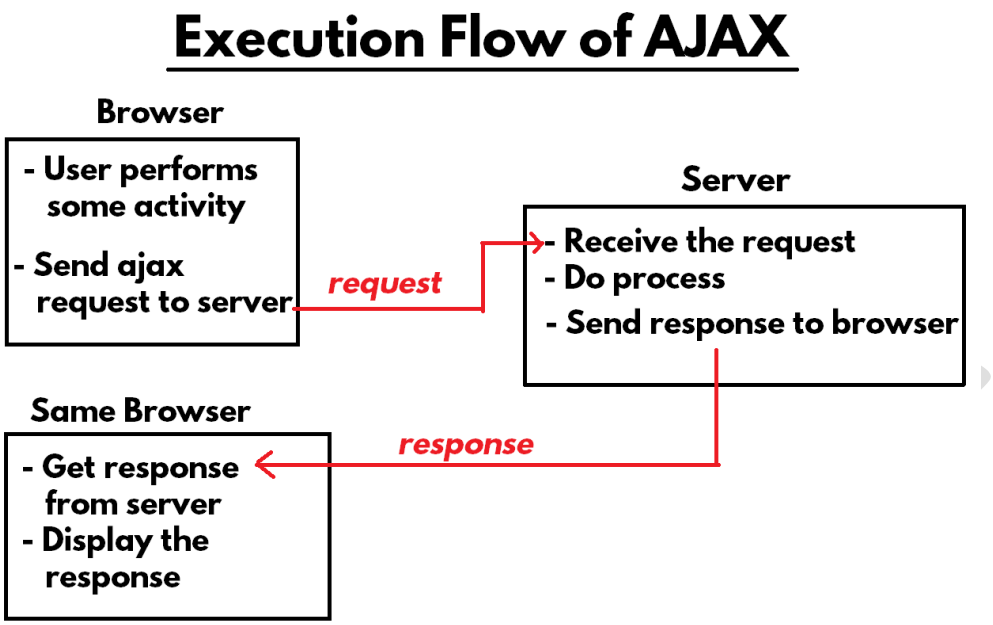
```
<html>
 <head>
    <title>External JavaScript - Page 2</title>
 </head>
 <body>
    <h1>External JavaScript - Page 2</h1>
    <script src="JavaScript.js"></script>
 </body>
</html>
```

AJAX

Introduction to AJAX

- AJAX is not a language, but it is a “concept”, which is used to “send a background request from browser to server” and also “get background response from server to browser”, without refreshing the web page in the browser.

- AJAX allows us to interact with the server and get some data from server, without refreshing the full web page.
- Ex: Google search, IRCTC search trains.



Advantages of AJAX

- Executes faster.
- Less burden on browser (client) and server.
- User experience is better.

Types of AJAX request

- In AJAX, the browser can send the following four types of requests:
  1. GET : Used to retrieve / search data from server.
  2. POST : Used to insert data to server.
  3. PUT : Used to update data on server.
  4. DELETE : Used to delete data from server.

The "XMLHttpRequest" class

- We can send ajax request to server and get ajax response from server by using a pre-defined class called "XMLHttpRequest".
- This class is supported by all the modern browsers (except IE 6, 7 and 8).

Members of XMLHttpRequest

- XMLHttpRequest class
  1. "readyState" property
  2. "onreadystatechange" property
  3. "responseText" property
  4. "open( )" method
  5. "send( )" method

open( ) method:

- This method is used to specify the request details such as server url (address), type of the request (GET / POST).

▪ **Server url**: The address (url) of server program, to which you want to send request.

▪ **Type:** Represents type of the request.

Options: GET | POST | PUT | DELETE

GET : Retrieving data from server

POST : Inserting data

PUT : Updating data

DELETE : Deleting data

Syntax: open("type of request", "url")

Example: open("get", "http://localhost:7070")

send( ) method

- This method sends an asynchronous request (background request) to server.

Syntax: send( )

Example: send( )

readyState

- It represents a number, that represents current status of the request.

Syntax: readyState

Example: readyState

0 : Request not initialized.

1 : Request opened (open( ) method is called).

2 : Request sent to server (send( ) method called).

3 : Server started processing the request.

4 : Response received from server.

onreadystatechange

- This property stores a callback function, which executes automatically, when the "readyState" property gets changed.

Syntax: onreadystatechange = functionname;

Example: onreadystatechange = fun1;

responseText

- This property stores the actual response sent from the server to the browser.
- It represents the data in string format.

Syntax: responseText

Example: responseText

AJAX – NodeJS – Simple - Example

Creating the application

- Create "c:\ajax" folder.
- Create "index.html" and "index.html" files in the "c:\ajax" folder.

c:\ajax

- message.txt
- index.html

c:\ajax\message.txt

```
Hello World
```

c:\ajax\index.html

```
<html>
  <head>
    <title>JavaScript AJAX - Simple</title>
  </head>
  <body>
    <h1>JavaScript AJAX - Simple</h1>
    <input type="button" id="button1" value="GET data from server">
    <div id="div1"></div>
    <script>
      var xhr;
```

```
        document.getElementById("button1").addEventListener("click", fun1);
        function fun1()
        {
           xhr = new XMLHttpRequest();
           xhr.open("GET", "/message.txt");
           xhr.onreadystatechange = fun2;
           xhr.send();
        }

        function fun2()
        {
           if (xhr.readyState == 4)
           {
              document.getElementById("div1").innerHTML += xhr.responseText + "<br>";
           }
        }
     </script>
  </body>
</html>
```

Execution

- Download and install “nodejs” from “https://nodejs.org”
- Open “Command Prompt” and run the following commands:

  cd c:\ajax

  npm install http-server -c-0

  http-server
- Open browser and enter the url: http://localhost:8080/index.html

AJAX – NodeJS – Json Object - Example

Creating the application

- Create “c:\ajax” folder.
- Create “employee.json” and “index.html” files in the “c:\ajax” folder.

c:\ajax

- employee.json
- index.html

c:\ajax\employee.json

```
{ "empid": 1, "empname": "Scott", "salary": 4000 }
```

c:\ajax\index.html

```
<html>
  <head>
    <title>JavaScript AJAX - Json Object</title>
  </head>
  <body>
    <h1>JavaScript AJAX - Json Object</h1>
    <input type="button" id="button1" value="Get data from server"><br>
    Emp ID: <input type="text" id="txt1"><br>
    Emp Name: <input type="text" id="txt2"><br>
    Salary: <input type="text" id="txt3"><br>

    <script>
      var xhr;
      document.getElementById("button1").addEventListener("click", fun1);
      function fun1()
      {
        xhr = new XMLHttpRequest();
        xhr.open("GET", "/employee.json");
        xhr.onreadystatechange = fun2;
        xhr.send();
      }

      function fun2()
      {
        if (xhr.readyState == 4)
        {
          var emp = JSON.parse(xhr.responseText);
          document.getElementById("txt1").value = emp.empid;
          document.getElementById("txt2").value = emp.empname;
          document.getElementById("txt3").value = emp.salary;
        }
      }
    </script>
  </body>
</html>
```

Execution

- Download and install “nodejs” from “https://nodejs.org”
- Open “Command Prompt” and run the following commands:

  cd c:\ajax

  npm install http-server -c-0

  http-server

- Open browser and enter the url: http://localhost:8080/index.html

HARSHA

AJAX – NodeJS – Json Array - Example

Creating the application

- Create “c:\ajax” folder.
- Create “employees.json” and “index.html” files in the “c:\ajax” folder.

c:\ajax

- employees.json
- index.html

c:\ajax\employees.json

```
[
    { "empid": 1, "empname": "Scott", "salary": 4000 },
    { "empid": 2, "empname": "Allen", "salary": 7500 },
    { "empid": 3, "empname": "Jones", "salary": 9200 },
    { "empid": 4, "empname": "James", "salary": 8400 },
    { "empid": 5, "empname": "Smith", "salary": 5600 }
]
```

c:\ajax\index.html

```
<html>
  <head>
    <title>JavaScript AJAX - Json Array</title>
  </head>
  <body>
    <h1>JavaScript AJAX - Json Array</h1>
    <input type="button" id="button1" value="Get data from server">
    <table id="table1" border="1"></table>

    <script>
      var xhr;
      document.getElementById("button1").addEventListener("click", fun1);
      function fun1()
      {
        xhr = new XMLHttpRequest();
        xhr.open("GET", "/employees.json");
        xhr.onreadystatechange = fun2;
        xhr.send();
      }

      function fun2()
      {
        if (xhr.readyState == 4)
        {
          var emps = JSON.parse(xhr.responseText);
```

```
            document.getElementById("table1").innerHTML = "<tr> <th>Emp ID</th> <th>Emp
Name</th> <th>Salary</th> </tr>";
            for (var i = 0; i < emps.length; i++)
            {
              document.getElementById("table1").innerHTML += "<tr> <td>" + emps[i].empid + "</td>
<td>" + emps[i].empname + "</td> <td>" + emps[i].salary + "</td> </tr>";
            }
          }
        }
      </script>
    </body>
</html>
```

Execution

- Download and install “nodejs” from “https://nodejs.org”
- Open “Command Prompt” and run the following commands:

  cd c:\ajax

  npm install http-server -c-0

  http-server
- Open browser and enter the url: http://localhost:8080/index.html

HARSHA

JavaScript - New Features

Introduction to JavaScript 6

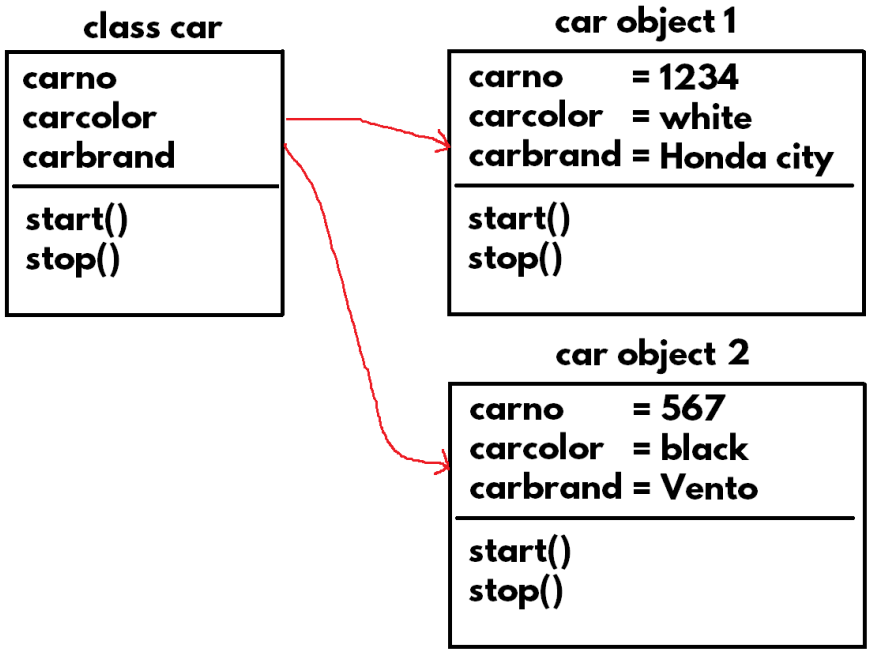
- "JavaScript 6" version in released in 2015, which is also called as "EcmaScript 6 or ES 6".
- JavaScript 6 is the superset of JavaScript 5, which contains the following new features:
    1. Classes
    2. Constructors
    3. Inheritance
    4. Method Overriding
    5. Set and Get Methods
    6. Default Arguments
    7. Arrow Functions
    8. Let
    9. Const
    10. Rest
    11. Destructuring
    12. Multiline Strings
    13. String Interpolation
    14. Reading Elements from Array

Class-based OOP

Classes

- "Object" is a "physical item", which is a collection of properties (details) and methods (manipulations).
- "Class" is a "model" of objects, which defines the list of properties and methods of the objects.

HARSHA



Steps for working with Classes and Objects

Create a class

```
class classname
{
}
```

Create an object

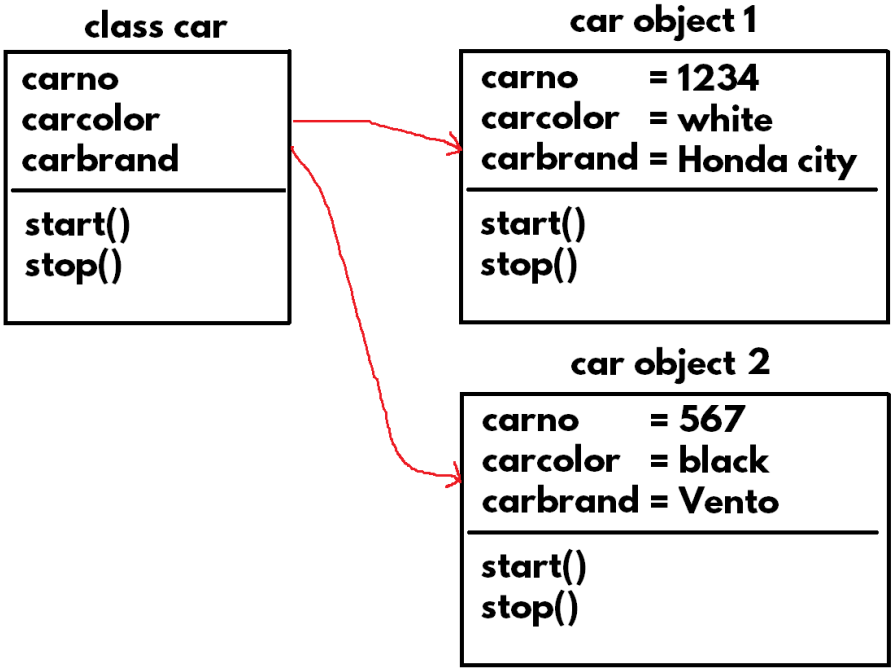
```
new classname();
```

Example on Classes and Objects

```
<html>
  <head>
    <title>Classes</title>
  </head>
  <body>
    <h1>Classes</h1>
    <script>
      class Student
      {
      }
      var s = new Student();
      console.log(s);
    </script>
  </body>
</html>
```

Classes

- "Object" is a "physical item", which is a collection of properties (details) and methods (manipulations).
- "Class" is a "model" of objects, which defines the list of properties and methods of the objects.



Steps for working with Classes and Objects

Create a class

```
class classname
{
}
```

Create an object

```
new classname();
```

Example on Classes and Objects

```
<html>
  <head>
    <title>Classes</title>
  </head>
  <body>
    <h1>Classes</h1>
    <script>
      class Student
      {
      }
      var s = new Student();
```

```
        console.log(s);
      </script>
    </body>
  </html>
```

Constructors

- Constructor is a special method in the class, that executes every time when we created an object for the class.
- Constructor's name should be "constructor".
- Constructor can receive arguments; but can't return any value.
- It is not possible to call the constructor without creating an object.
- Constructors are mainly used to create an initialize properties in the class.
- Parameterized Constructor is a constructor that receives one or more arguments (parameters) and initializes the same to the respective properties.

Syntax of Constructor

```
class  classname
{
   constructor(arguments)
    {


    }
}
```

Example on Constructors

```
<html>
  <head>
    <title>Constructors</title>
  </head>
  <body>
    <h1>Constructors</h1>
    <script>
      class Student
      {
        constructor()
        {
          this.studentid = 101;
          this.studentname = "Scott";
          this.marks = 70;
        }
      }
      var s = new Student();
      console.log(s.studentid);
```

```
        console.log(s.studentname);
        console.log(s.marks);
      </script>
    </body>
  </html>
```

Example on Parameterized Constructors

```
<html>
  <head>
    <title>Parameterized Constructor</title>
  </head>
  <body>
    <h1>Parameterized Constructor</h1>
    <script>
      class Student
      {
        constructor(a, b, c)
        {
          this.studentid = a;
          this.studentname = b;
          this.marks = c;
        }
      }
      var s = new Student(201, "Smith", 80);
      console.log(s.studentid);
      console.log(s.studentname);
      console.log(s.marks);
      console.log(s.result());
    </script>
  </body>
</html>
```

Methods

- Method is a function that manipulates data of the object.

Syntax of Method

```
class  classname
{
   methodname(arguments)
    {


    }
}
```

Example on Methods

```
<html>
  <head>
    <title>Methods</title>
  </head>
  <body>
    <h1>Methods</h1>
    <script>
      class Student
      {
        constructor(a, b, c)
        {
          this.studentid = a;
          this.studentname = b;
          this.marks = c;
        }
        result()
        {
          if (this.marks >= 35)
          {
            return "Pass";
          }
          else
          {
            return "Fail";
          }
        }
      }
      var s = new Student(101, "Scott", 70);
      console.log(s.studentid);
      console.log(s.studentname);
      console.log(s.marks);
      console.log(s.result());
    </script>
  </body>
</html>
```

Inheritance

- Inheritance is a concept of extending the parent class, by creating the child class.
- When you create an object of child class, all the members of parent class will be automatically added to the child class's object.

Syntax of Inheritance

```
class  parentclassname
{
   constructor(arguments)
   {
```

HARSHA

```
    }
}

class childclassname extends parentclassname
{
   constructor(arguments)
   {
      super(arguments);
   }
}
```

Example on Inheritance

```
<html>
  <head>
    <title>Inheritance</title>
  </head>
  <body>
    <h1>Inheritance</h1>
    <script>
      class Person
      {
        constructor(name, age)
        {
          this.name = name;
          this.age = age;
        }
      }
      class Student extends Person
      {
        constructor(name, age, studentid, marks)
        {
          super(name, age);
          this.studentid = studentid;
          this.marks = marks;
        }
      }
      var s = new Student("scott", 20, 101, 78);
      console.log(s.name);
      console.log(s.age);
      console.log(s.studentid);
      console.log(s.marks);
    </script>
  </body>
</html>
```

HARSHA

Method Overriding

- Method Overriding is a concept of extending the "parent class method", by creating similar method in the "child class", with same signature (name and arguments).

Syntax of Method Overriding

```
class parentclassname
{
   method(arguments)
    {
    }
}

class childclassname extends parentclassname
{
   method(arguments)
    {
        super.method(arguments);
    }
}
```

Example on Method Overriding

```
<html>
  <head>
    <title>Method Overriding</title>
  </head>
  <body>
    <h1>Method Overriding</h1>
    <script>
      class Person
      {
        constructor(name, age)
        {
          this.name = name;
          this.age = age;
        }
        display()
        {
          return "name is " + this.name + ", age is " + this.age;
        }
      }
      class Student extends Person
      {
        constructor(name, age, studentid, marks)
        {
          super(name, age);
```

HARSHA

```
            this.studentid = studentid;
            this.marks = marks;
        }
        display()
        {
            return super.display() + ", studentid is " + this.studentid + ", marks is " + this.marks;
        }
    }

    class Employee extends Person
    {
        constructor(name, age, employeeid, salary)
        {
            super(name, age);
            this.employeeid = employeeid;
            this.salary = salary;
        }
        display()
        {
            return super.display() + ", employeeid is " + this.employeeid + ", salary is " + this.salary;
        }
    }
    var s = new Student("scott", 20, 101, 78);
    console.log(s.display());
    var e = new Employee("smith", 25, 201, 9500);
    console.log(e.display());
</script>
</body>
</html>
```

Misc. Concepts

Set and Get Methods

- The "set" method executes automatically when we assign a value into the accessor. It performs validation and assigns the value into the property, if it is valid.
- The "get" method executes automatically when we retrieve the value from the accessor. It returns the current value of the property.

Steps for working with Set Method and Get Method

Create Set Method

```
set accessorname(argumentname)
{
    this.property = argumentname;
}
```

Create Get Method

```
get accessorname(argumentname)
{
    return this.property;
}
```

Call Set Method

```
accessorname = actualvalue;
```

Create Get Method

```
accessorname
```

Example on Set and Get Methods

```
<html>
  <head>
    <title>Set and Get Methods</title>
  </head>
  <body>
    <h1>Set and Get Methods</h1>
    <script>
      class Person
      {
        constructor()
        {
          this.name = null;
          this.age = 0;
        }
        set Name(value)
        {
          if (value.length <= 20)
          {
            this.name = value;
          }
        }
        get Name()
        {
          return this.name;
        }
        set Age(value)
        {
          if (value >= 18 && value <= 70)
          {
            this.age = value;
          }
        }
        get Age()
        {
```

HARSHA

```
            return this.age;
        }
    }
    var p = new Person();
    p.Name = "abcdefghijklmnopqrstuvwxyz";
    p.Age = 800;
    console.log(p.Name);
    console.log(p.Age);
    p.Name = "Scott";
    p.Age = 30;
    console.log(p.Name);
    console.log(p.Age);
  </script>
 </body>
</html>
```

Default Arguments

- Default Arguments are used to provide a default value for the method of a parameter.
- When we call the method and don't supply a value for the parameter, the specified default value will be assigned to the parameter automatically.

Steps for working with Default Arguments

```
method(argument = defaultvalue)
{
}
```

Example on Default Arguments

```
<html>
 <head>
  <title>Default Arguments</title>
 </head>
 <body>
  <h1>Default Arguments</h1>
  <script>
   class sample
   {
     method1(x = 100)
     {
       console.log(x);
     }
   }
   var s = new sample();
   s.method1(200);
   s.method1();
  </script>
 </body>
</html>
```

Arrow Functions

- "Arrow Functions" are the functions created using "=>" (arrow) operator.
- Arrow Function's "this" keyword reflects the current object; it will not be changed by the caller.

Steps for working with Arrow Functions

```
this.method = (arguments) =>
{
}
```

Example on Arrow Functions

```
<html>
  <head>
    <title>Arrow Functions</title>
  </head>
  <body>
    <h1>Arrow Functions</h1>
    <input type="button" value="Click me" id="button1">
    <script>
      class sample
      {
        constructor()
        {
          this.x = 100;
          this.method1 = () =>
          {
            console.log(this);
          }
        }
      }
      var s = new sample();
      document.getElementById("button1").addEventListener("click", s.method1);
    </script>
  </body>
</html>
```

Let

- The "let" keyword can be used as alternative for "var" keyword, to create variables.
- The "var" keyword always creates "local variables" or "block level variables"; but "let" keyword creates "block level variables".

Steps for working with Let

```
let  variablename = value;
```

Example on Let

```
<html>
  <head>
    <title>Let</title>
  </head>
  <body>
    <h1>Let</h1>
    <script>
      for (var i = 1; i <= 10; i++)
      {
      }
      console.log(i);

      for (let j = 1; j <= 10; j++)
      {
      }
      console.log(j);
    </script>
  </body>
</html>
```

Const

- The "const" keyword is used to create constants.
- We can't change the value of the constant.
- If you try to change the value of constant, it shows error.

Steps for working with Const

```
const  constantname = value;
```

Example on Const

```
<html>
  <head>
    <title>Const</title>
  </head>
  <body>
    <h1>Const</h1>
    <script>
      const n = 10;
      console.log(n);
      n = 20;
      console.log(n);
    </script>
  </body>
</html>
```

Rest (...) Operator

- The rest (. . .) operator (should three dots) represents all remaining values.

- It can be used with methods / arrays.

Steps for working with Rest Operator with Methods

```
method(…argumentname)
{
}
```

Steps for working with Rest Operator with Arrays

```
var  arrayvariable = [ …array1, …array2 ];
```

Example on Rest Operator With Methods

```
<html>
  <head>
    <title>Rest - Methods</title>
  </head>
  <body>
    <h1>Rest - Methods</h1>
    <script>
      class sample
      {
        method1(arg1, arg2, ...rest)
        {
          console.log(rest);
        }
      }
      var s = new sample();
      s.method1(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    </script>
  </body>
</html>
```

Example on Rest Operator With Arrays

```
<html>
  <head>
    <title>Rest - Arrays</title>
  </head>
  <body>
    <h1>Rest - Arrays</h1>
    <script>
      var south = ["Hyderabad", "Chennai", "Bangalore"]
      var north = ["Pune", "Mumbai", "New Delhi"];
      var cities = [...south, ...north];
      console.log(cities);
    </script>
  </body>
</html>
```

HARSHA

Destructuring

- Destructuring is used to retrieve each value of an array into respective variables.

Steps for working with Destructuring

var [variable1, variable2, …] = arrayname;

Example 1 on Destructuring

```
<html>
  <head>
    <title>Destructuring</title>
  </head>
  <body>
    <h1>Destructuring</h1>
    <script>
      var cities = ["Hyderabad", "Chennai", "Bangalore"];
      var [city1, city2, city3] = cities;
      console.log(city1);
      console.log(city2);
      console.log(city3);
    </script>
  </body>
</html>
```

Example 2 on Destructuring

```
<html>
  <head>
    <title>Destructuring - Skip</title>
  </head>
  <body>
    <h1>Destructuring - Skip</h1>
    <script>
      var cities = ["Hyderabad", "Chennai", "Bangalore", "Pune", "Mumbai"];
      var [city1, city2, , city4] = cities;
      console.log(city1);
      console.log(city2);
      console.log(city4);
    </script>
  </body>
</html>
```

Example 3 on Destructuring

```
<html>
  <head>
    <title>Destructuring - Rest</title>
  </head>
  <body>
    <h1>Destructuring - Rest</h1>
    <script>
```

HARSHA

```
        var cities = ["Hyderabad", "Chennai", "Bangalore", "Pune", "Mumbai"];
        var [city1, city2, ...rest] = cities;
        console.log(city1);
        console.log(city2);
        console.log(rest);
      </script>
    </body>
  </html>
```

Multiline Strings

- This concept is used to create a string with multiple lines of text.
- The multiline string should be enclosed within backticks (` `).

Steps for working with Multiline Strings

```
`
    line 1
    line 2
    line 3
    ...
`
```

Example on Multiline Strings

```
<html>
  <head>
    <title>Multiline Strings</title>
  </head>
  <body>
    <h1>Multiline Strings</h1>
    <script>
var s = `hello
how
are
you`;
        console.log(s);
    </script>
  </body>
</html>
```

String Interpolation

- "String Interpolation" replaces the expressions in the strin with actual values of the respective variables.
- These strings must be enclosed within backticks (` `), instead of single quotes (' ') or double quotes (" ").

HARSHA

Steps for working with String Interpolation

```
` text here ${variable} text here`
```

Example on String Interpolation

```
<html>
  <head>
    <title>String Interpolation</title>
  </head>
  <body>
    <h1>String Interpolation</h1>
    <script>
      var name = "Scott";
      var result = `Hello, my name is ${name}`;
      console.log(result);
    </script>
  </body>
</html>
```

Reading Elements from Array

- The most common task of the developer is reading data from an array and displaying the same.
- This can be done in several ways:
  1. For Loop
  2. ForEach Loop
  3. ForOf Loop
  4. ForIn Loop

Steps for working with For Loop

```
for (i = 0; i < array.length; i++)
{
  array[i]
}
```

- For loop is index based.
- Forward / backward iterations are possible.
- We can start the loop from the middle also.

Steps for working with ForEach Loop

```
array.forEach( function(variable)
{
    variable
```