JavaScript

“The important thing is a clear understanding of how to call an element”

* To show output in javascript:-

1. alert() function:

syntax : alert(‘ ’);

1. write() function:  
   syntax : document.write(“ ”);  
   note: to use write() function we need ‘document’ object.
2. log() function:  
   syntax : document.log(“ ”);  
   note: to use log() function we need ‘console’ object and it shows the output in console area.  
     
   Note : we can use alert() function to show output as pop-up window and we can use write()function to show output directly.

* javascript file placement:-  
  There are three types of javascript file –  
  1. Inline javascript  
  2. Internal javascript  
  3. External javascript
* Keywords, Data types, Comment:-  
  1. Keywords in javascript which has special meaning -

break, case, catch, continue, debugger, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with.

2. Data Types -   
 String: letters and numbers

Number: only decimal and simple numbers

Boolean: true or false (machine language)

Objects:

3. Comment -  
// single line comment…

/\* multiple line comment… \*/

* Variables:-  
  - Variable names can only contain letters, numbers, underscores, or dollor signs.

(A-Z, a-z, 0-9, \_ , $)

- Variable names must begin with a letter, an underscore (\_) or a dollor sign ($), not any numbers.

- Variable names can’t contain spaces.

- Reserved words (like JavaScript Keywords) can’t be use as names.

- Variable names are case sensitive.

Note: var, catch is a key word, can’t use as a variable. And must write ‘var’ before variable name.

\* variables writing:  
 Example: var name;  
 name = “kamal”;   
 document.write(name );  
 \* multiple variables writing:  
 Example: var name, age, country;  
 name = “kamal”;  
 age = 25;  
 country = “Banglaadesh”;  
 document.write(name <br />);   
 document.write(age <br />);   
 document.write(country);

* Number Method |to Fixed()|toPrecision() Method:-  
  \* toFixed() – Method will show you how many numbers are after dot(.)  
  Example: var number = 2.546;  
   console.log(number.toFixed(2));  
  Output: 2.55 (in the console area)  
  \* toPrecision() –Method will show you total number which we enter in that method.  
  Example: var number = 2.546;  
   console.log(number. toPrecision (2));   
   console.log(number. toPrecision (3));  
  Output: 2.5 (in the console area)  
  Output: 2.55 (in the console area)  
  \* to convert number to string :-  
   we can use parseInt(); - Method: for simple number  
  \* to convert number to string :-  
   we can use parsefloat(); - Method: for decimal number  
  Example: var num = “string”;  
   num = parseInt(num);  
   console.log(typeof(num));  
  Output: number (in the console area)  
  \* Number() Method :-  
   we can use “console.log(Number(“ ”));”
* String concatenation (যুক্ত করা):-  
  \* to add one string to another we need to put (+) sign between them and they will be concatenated.
* Library functions for string:-  
  \* property:- .length; -it can show you the character length of a variable.  
  syntax :- variable\_name.length;  
  \* prompt() function – it can take values from users.  
  \* charAt() function – it shows the index number which we want to show.   
  \* toUpperCase() function  
  \* toLowerCase() function  
  \* concat() function – it can add one string to another.  
  \* slice() function – it can cut off those number of index which you select.
* Arithmetic and assignment operator:-  
  \* arithmetic operator:   
   +, -, \*, /, %(Modulus), \*\*(Exponent), ++(increament), --(decreament).  
  \* assignment operator:  
   =, +=, -=, \*=, /=, %=, \*\*=.  
   Example: var x = 3; x += 4;  
   document.write(x);  
   Output: 7
* Area various shapes:-  
  Example: var base = parseFloat(prompt(“Enter Base : ”));  
   var height = parseFloat(prompt(“Enter Height : ”));  
    
   var area = base\* height;  
     
   document.write(“Area : ” + area);
* Temperature Converter:-  
  Formula of temperature converting:  
  Farenheit to Celsius: (F-32) \* (5/9)   
  Celsius to Farenheit: (C \* (9/5)+32)
* Relational and Logical Operator:  
  *Relational operator* : >, >=, <, <=, ==, ===, !=, !==  
  \* ==/!= : it checks both sides value, however it’s integer or string, it only checks the value’s equal or not, if values are equal it shows true and if not equal it shows false.  
  \* ===/!== : it checks both sides value, however it’s integer or string and it checks both datatype’s and value’s equal or not, if both are equal it shows true and if not equal it shows false.  
  *Logical operator* : &&, ||, !  
  \* && (Logical AND): it shows true if both side’s value’s condition is true, otherwise, it shows false.  
  \* || (Logical OR) : it shows true if anyone from both side's value’s condition is true, otherwise, it shows false.  
  \* ! (Logical NOT) : it is a different logic and it always shows its condition’s opposite. If the condition is true it shows false, and if the condition is false it shows true.
* Control Statement (if, else if, else) :  
  \* There are two types of control statement :-  
   1. Conditional Loop statement:-  
   - if, else if, else  
   - switch  
   2. Loop control statement:-  
   - for  
   - while  
   - do while  
  \* How (if, else if, else) works :-  
  Example :-   
   var num = 6;  
   if (num>0)  
   document.write(“Positive”);   
   else if (num<0)  
   document.write(“Negitive”);   
   else  
   document.write(“Zero”);  
  Output :- Positive  
  Note:- We will always use if and else consistently if will set the first condition and else will set the last condition, and remaining middle conditions will set as else if.
* Letter Grade program:-  
  Example:- (Output will show which number you enter)  
  var marks = parseFloat(prompt("Enter your marks here:"));  
  if (marks >= 80)  
   document.write("<h1>" + "Your marks is - " + marks + " and you got A+." + "</h1>");  
  else if (marks >= 70)  
   document.write("<h1>" + "Your marks is - " + marks + " and you got A." + "</h1>");   
  else if (marks >= 60)  
   document.write("<h1>" + "Your marks is - " + marks + " and you got A-." + "</h1>");   
  else if (marks >= 50)  
   document.write("<h1>" + "Your marks is - " + marks + " and you got B." + "</h1>");   
  else if (marks >= 40)  
   document.write("<h1>" + "Your marks is - " + marks + " and you got C." + "</h1>");   
  else if (marks >= 33)  
   document.write("<h1>" + "Your marks is - " + marks + " and you got D." + "</h1>");   
  else  
   document.write("<h1>" + "Your marks is - " + marks + " and you are Fail." + "</h1>");
* Logical operator program:  
  Example:- (Output will show which number you enter)  
  var marks = parseFloat(prompt("Enter your marks here:"));   
  if (marks > 100 || marks < 0)  
   document.write("<h1>" + "Your marks is : " + marks + ", and it is an invalid mark + "</h1>");  
  if (marks >= 80 && marks <= 100)  
   document.write("<h1>" + "Your marks is : " + marks + ", and you got A+." + "</h1>");  
  else if (marks >= 70 && marks <= 79)  
   document.write("<h1>" + "Your marks is : " + marks + ", and you got A." + "</h1>");   
  else if (marks >= 60 && marks <= 69)  
   document.write("<h1>" + "Your marks is : " + marks + ", and you got A-." + "</h1>");   
  else if (marks >= 50 && marks <= 59)  
   document.write("<h1>" + "Your marks is : " + marks + ", and you got B." + "</h1>");   
  else if (marks >= 40 && marks <= 49)  
   document.write("<h1>" + "Your marks is : " + marks + ", and you got C." + "</h1>");   
  else if (marks >= 33 && marks <= 39)  
   document.write("<h1>" + "Your marks is : " + marks + ", and you got D." + "</h1>");   
  else  
   document.write("<h1>" + "Your marks is : " + marks + ", and you are Fail." + "</h1>");  
    
  Example -2(Vowel/Consonant marking program):   
  var letter = prompt("Enter your letter : ");  
  if (letter == "a" || letter == "e" || letter == "i" || letter == "o" || letter == "u")  
   document.write("<h1>" + " - It is 'Vowel'" + "</h1>");  
  else if (letter == "A" || letter == "E" || letter == "I" || letter == "O" || letter == "U")  
   document.write("<h1>" + " - It is 'Vowel'" + "</h1>");  
  else  
   document.write("<h1>" + " - It is 'Consonant'" + "</h1>");
* Switch statement :  
  Keywords : case(take the condition), break, default.  
  syntax : switch (variable\_name){  
   case “” :  
   type here what to print…  
   break;  
  }  
  Example : (multiple switch statement)  
  var digit = prompt(“Enter your letter : ”);  
  switch (digit) {   
   case “a” :  
   case “b” :  
   case “c” :  
   case “d” :  
   case “e” :   
   document.write("<h1>" + " - It is ‘Vowel’" + "</h1>");  
   break;   
   default :   
   document.write("<h1>" + " - It is 'Consonant'" + "</h1>");  
  }
* for Loop statement :  
  \* loop is mainly use for doing one thing repeatedly.  
  - for loop syntax :  
   for (initialization(starting), condition, update(++/--)) {  
   type here what to print ….  
   }  
  Example : ( 1 – 10 number counting program)  
  for (var x=1; x<=10; x+1){  
   document.write(“ ” + x);  
   }  
  Example -2 : ( 1-10 addition/ 1+2+3+…+10)  
  sum = 0;  
  for(var x = 1; x <= 10; x = x+1){  
   sum = sum + x;  
  }  
  document.write("<br />" + "<h1>1 + 2 + 3 + ... + 10 = " + sum + "</h1>");
* while loop statement :  
  while loop syntax :   
  initialization(starting); (example : var x=1)  
  while (condition (example: x<=10) ){  
   type here what to print ….   
   (example: document.write(“ ”+x); )  
   update(++/--) (example: x=x+1) ;  
  }  
  EX: create a program of the sum of all numbers that are divisible by 3 and 5 from 1-100.
* Do while loop statement :  
  do while loop syntax :   
  initialization(starting);   
  do{  
   type here what to print ….  
   update(++/--);  
  } while (condition);  
  Exercise :  
  var x = 1;  
  do{  
   document.write(“ ” + x);  
   x++;  
  } while (x <= 20);
* break and continue keyword :  
  break is used as a program end, where we put condition to break it.  
  continue is work for the program to break in the specific condition which we want to avoid or break.  
  Example :   
  for (var x=1; x <= 50; x++){  
   if(x == 10){(this program will break before printing 10)  
   break;  
   }  
   document.write(“ ” + x);  
  } (to print 10 we need to use break after print function)
* Ternary operator :  
  we can use that function instead of if/else if/else function  
  Example :  
  var number = Number(prompt(“Enter number : ”));  
  result = number > 0 ? “Positive” : number < 0 ? “Negative” : “Zero” ;  
  document.write(result);
* Function :  
  function structure :  
  function function\_name(parameter){  
   javascript codes to be written here….;  
  }  
  function\_name(parameter);  
    
  Example :  
  function sum(num1, num2){  
   var result = num1 + num2;  
   document.write(“Sum = ” + result);  
  }  
  sum(10,5);
* Array :  
  array syntax :  
  var array\_name = new Array(length in number);  
  array\_name[index(starts with 0)] = “name”;  
    
  other way to use array :  
  var array\_name = [value];  
    
  Example :  
  var cars = new array(5);  
  cars[0] = “Audi”;   
  cars[1] = “BMW”;   
  cars[2] = “Honda”;   
  cars[3] = “Ferrari”;   
  cars[4] = “Ford”;  
  document.write(cars);  
    
  \* array functions :  
   1. array.push() – add a new value in the end.   
   2. array.pop() – remove the last value.   
   3. array.shift() – remove first array element.   
   4. array.unshift() – add an element in the beginning.   
   5. array.indexof() – array value index number.
* Loop in Array :  
  Example :  
  var num = [10, 20, 30, 40, 50];  
  for (var i=0; i<=5; i++ ){  
  document.write(num[i]);  
  }
* Array related Methods :  
  \* push()  
  \* pop()  
  \* concat()  
  \* shift()  
  \* unshift()  
  \* slice()  
  \* splice(pos, noe, ele1, ele2…) – to add elements.  
  \* splice(pos-no, pos-no) – to remove elements.   
  \* sort() – sorted the element alphabetically.   
  \* reverse() – reverse the elements alphabetically.  
  \* to sort elements numerically we need to use anomynous function in the following way -  
  - var numbers = [20, 5, 25, 15, 1];  
  numbers.sort(function(a, b){  
   return a-b; /for reverse sorting type (b-a)  
  });
* Object :   
  \* Object is one kind of a variable but it can store multiple value at a time.  
  object syntax : (single object)  
  var object\_name = {  
   property = value;  
  }  
  to store multiple value at a time – we need to add a constructor (it can store property as a template)  
  \* constructor is a special kind of function. Its name's first word must be written in capital letter.  
  - function Name (parameter(variable\_name)){  
   this.variable\_name = value;  
  }  
  var name1 = new Name (value);  
  Example :  
  function Student (name, age, gpa) {  
   this.name = name;   
   this.age = age;   
   this.gpa = gpa;  
  }  
  var student1 = new Student(“Kamal”, 24, 4.50);   
  var student2 = new Student(“Rahim”, 22, 5.00);  
  document.write(student1);  
    
  - to add a display function in the function :-  
  function Name (parameter(variable\_name)){  
   this.variable\_name = value;  
   this.display = function(){  
   document.write(this. variable\_name);  
   }  
  }  
  var name1 = new Name (value);  
  Example :  
  function Student (name, age, gpa) {  
   this.name = name;   
   this.age = age;   
   this.gpa = gpa;   
    
   this.display = function(){  
   document.write(this.name);   
   document.write(this.age);   
   document.write(this.gpa);  
   }  
  }  
  var student1 = new Student(“Kamal”, 24, 4.50);   
  var student2 = new Student(“Rahim”, 22, 5.00);  
    
  student1.display();
* Math object :  
  \* Math.sqrt(x) – returns the square root of x.  
  Example :- Math.sqrt(25); Output :- 5  
    
  \* Math.abs(x) - returns the absolute value of x.  
  Example :- Math. abs(-5); Output :- 5  
    
  \* Math.sin(x) - returns the sine of x (x is in radians).  
  Example :- Math. sin(2); Output:- 0.9092974268256817  
    
  \* Math.pow(x,y) - returns the value of x to the power of y.  
  Example :- Math.pow(2,4); Output :- 16  
    
  \* Math.floor(x) - returns x, rounded downwards to the nearest integer.  
  Example :- Math.floor(2.5); Output :- 2  
    
  \* Math.ceil(x) - returns x, rounded upwards to the nearest integer.  
  Example :- Math.ceil(3.6); Output :- 4  
    
  \* Math.round(x) – rounds x to the nearest integer.  
  Example :- Math.round(4.5); Output :- 5  
    
  \* Math.max (x,y,z,..,n) - returns the number with the highest value.  
  Example :- Math.max(2,5,-6,3); Output :- 5  
    
  \* Math.min (x,y,z,..,n) - returns the number with the lowest value.   
  Example :- Math.max(2,5,-6,3); Output :- -6  
    
  \* Math.random() – returns a random number between 0 and 1.  
  Example :- Math.random();   
  Output: 0.3814651015773891  
    
  \* To generate random number from 1-5 :-  
  Example : Math.floor(Math.random()\* 5) + 1;  
  Output : then it can generate numbers from 1-5.

## Math Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [abs(x)](https://www.w3schools.com/jsref/jsref_abs.asp) | Returns the absolute value of x |
| [acos(x)](https://www.w3schools.com/jsref/jsref_acos.asp) | Returns the arccosine of x, in radians |
| [acosh(x)](https://www.w3schools.com/jsref/jsref_acosh.asp) | Returns the hyperbolic arccosine of x |
| [asin(x)](https://www.w3schools.com/jsref/jsref_asin.asp) | Returns the arcsine of x, in radians |
| [asinh(x)](https://www.w3schools.com/jsref/jsref_asinh.asp) | Returns the hyperbolic arcsine of x |
| [atan(x)](https://www.w3schools.com/jsref/jsref_atan.asp) | Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians |
| [atan2(y, x)](https://www.w3schools.com/jsref/jsref_atan2.asp) | Returns the arctangent of the quotient of its arguments |
| [atanh(x)](https://www.w3schools.com/jsref/jsref_atanh.asp) | Returns the hyperbolic arctangent of x |
| [cbrt(x)](https://www.w3schools.com/jsref/jsref_cbrt.asp) | Returns the cubic root of x |
| [ceil(x)](https://www.w3schools.com/jsref/jsref_ceil.asp) | Returns x, rounded upwards to the nearest integer |
| [cos(x)](https://www.w3schools.com/jsref/jsref_cos.asp) | Returns the cosine of x (x is in radians) |
| [cosh(x)](https://www.w3schools.com/jsref/jsref_cosh.asp) | Returns the hyperbolic cosine of x |
| [exp(x)](https://www.w3schools.com/jsref/jsref_exp.asp) | Returns the value of Ex |
| [floor(x)](https://www.w3schools.com/jsref/jsref_floor.asp) | Returns x, rounded downwards to the nearest integer |
| [log(x)](https://www.w3schools.com/jsref/jsref_log.asp) | Returns the natural logarithm (base E) of x |
| [max(x, y, z, ..., n)](https://www.w3schools.com/jsref/jsref_max.asp) | Returns the number with the highest value |
| [min(x, y, z, ..., n)](https://www.w3schools.com/jsref/jsref_min.asp) | Returns the number with the lowest value |
| [pow(x, y)](https://www.w3schools.com/jsref/jsref_pow.asp) | Returns the value of x to the power of y |
| [random()](https://www.w3schools.com/jsref/jsref_random.asp) | Returns a random number between 0 and 1 |
| [round(x)](https://www.w3schools.com/jsref/jsref_round.asp) | Rounds x to the nearest integer |
| [sin(x)](https://www.w3schools.com/jsref/jsref_sin.asp) | Returns the sine of x (x is in radians) |
| [sinh(x)](https://www.w3schools.com/jsref/jsref_sinh.asp) | Returns the hyperbolic sine of x |
| [sqrt(x)](https://www.w3schools.com/jsref/jsref_sqrt.asp) | Returns the square root of x |
| [tan(x)](https://www.w3schools.com/jsref/jsref_tan.asp) | Returns the tangent of an angle |
| [tanh(x)](https://www.w3schools.com/jsref/jsref_tanh.asp) | Returns the hyperbolic tangent of a number |
| [trunc(x)](https://www.w3schools.com/jsref/jsref_trunc.asp) | Returns the integer part of a number (x) |

* Guessing game program :  
  1. Guess a number from 1 to 5. Take the number from users.  
  2. Generate a number from 1 to 5.  
  3. If the guess number matches the random number; then show won else lose.  
  4. Run the game for 5 times.  
  5. Show the number of won and lost.
* Date object and it’s methds :  
  There are four ways of instantiating a date:  
  \* var d = new Date();  
  \* var d = new Date(milliseconds);  
  \* var d = new Date(dateString);  
  \* var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);  
    
  \* Some important methods of Date objects :-  
  - getFullYear();  
  - getMonth();  
  - getDate();  
  - getDay();  
  - getHours();  
  - getMinutes();  
  - getSeconds();
* Introduction to DOM (Document Object Model) | select HTML elements :  
    
  \* *Finding HTML elements* :-  
  Method :- document.getElementById(id);   
  Method :- document.getElementsByTagName(tag name) [index]; -> to find elements by tag name.  
  Method :- document.getElementsByClassName(class name)[index]; -> to find elements by class name.  
    
  \* *Changing HTML elements* :-  
  Property :- element.innerHTML = “new html content”; -> change the inner HTML of an element.  
  Property :- element.attribute = “new value”; -> change the attribute value of an HTML element.   
  Property :- element.style.property = “new style”; -> change the style of an HTML element.  
  note : to write multiple css style use -.style.cssText = “ ”;  
  Method :- element.setAttribute(attribute, value); -> change the attribute value of an HTML element.
* Query selector() :  
  \* *Finding HTML elements* :-  
  Method :- document.querySelector(“#id/.class/tag name”);   
  Method :- document.querySelectorAll(“tag name”)[index]; -> to find elements we need an index number. Note : querySelectorAll() = getElementsByTageName()/getElementsByClassName.
* Event Handler to Onclick event :  
  Property :- onclick = “ function\_name() ” -> to use this method we need a created function first.  
  Example :   
  - html code :-  
  <button onclick= “btn1()”>Button 1</button>  
  <button onclick= “btn2()”>Button 2</button>  
  - js code :-  
  function btn1(){ alert(“This is button 1”); }   
  function btn2(){ alert(“This is button 2”); }  
  Note : to use hover effects in js we need to create function first then add it where we want to show.
* More on DOM | adding, deleting, removing elements :  
  \* DOM methods and property -  
  - DOM Methods – getElementById()  
  - DOM property – innerHTML   
  \* Find/get :- Methods  
  -> document.getElementById(id)   
  -> document.getElementsByClassName(name)   
  -> document.getElementsByTagName(name)   
  -> document.querySelector(id/class/tag name)   
  -> document.querySelectorAll()[index]  
  \* Changing HTML elements :-  
  -> property :-  
   - element.innerHTML = new HTML content  
   - element.attribute = new value  
   - element.style.property = new style  
  -> method :-  
   - element.setAttribute(attribute, value) = change the attribute value of an HTML element  
  \* Create an HTML element :-  
  -> document.createElement()  
  -> document.createTextNode()  
  \* Add HTML element :-  
  -> appendChild()  
  \* Create an HTML element with insertBefore(before, after)  
  -> insertBefore(parameter1, parameter2)  
  Note : parameter1 = before element  
   parameter2 = after element  
  \* Remove HTML element :-  
  -> removechild()  
  \* Replace HTML element :-  
  -> replaceChild(newChild, oldChild)  
  \* To show class :-  
  -> classList | to add class use .add()| to remove class use .remove()
* Image slider program :-  
  \* important thing is that image slider needs onclick() methods to control.
* Changing CSS style dynamically :-  
  \* important thing is that changing CSS style needs javascript methods to control.
* Event Listener :-  
  \* Property :- element.addEventListener(“type of event”, function\_name )  
  note:there will be no brackets after the function\_name.  
  \* without creating function we also can use addEventListener() method :-  
  element.addEventListener (“click”, function(){})  
  note : we need to create an anonymous function().
* Event Listener with multiple elements :-  
  \* we can use addEventListener() with multiple elements. To use this method with multiple elements just need a loop function named for() loop.  
  Example :-  
  var heading = document.querySelector("h1");  
  var button = document.querySelectorAll(".myButton");  
  var len = document.querySelectorAll(".myButton").length;  
  for (i = 0; i < len; i++) {  
   button[i].addEventListener("click", function () {  
   var text = this.innerHTML; (to show text of the element)  
   heading.innerHTML = text + " is clicked";  
   });  
   }
* How to Play Audio :-  
  -> this program needs all the functions like for() loop function, switch() function, addEventListener() method all together and a new method audio() is also required to play the audio.  
  \* Method :-  
  -> Audio(“source of audio”) - (this is required to find the audio file)  
  -> element.play() - (this is required to play the audio)
* How to add animation :-  
  \* Adding an animation with javascript is very simple. First we need to create an animation class in CSS stylesheet. Then we need to create a function and then add the CSS style class into it.
* Keypress Listener :-  
  \* To add keypress listener just need an event named “keypress” and also a property named “key” to show which key is pressed.  
  Example :-  
  *html code*:-  
  <h1>Press any to show…</h1>  
  *js code*:-  
  document.addEventListener("keypress", function (event) {  
   var text = event.key;  
   document.querySelector("h1").innerHTML = "You have pressed " + "'" + text.toUpperCase() + "'.";  
  });
* Error Handling | try, catch, finally block (keyword) :-  
  \* try | catch | finally :-   
  try{ test codes  
   main codes will be written here….  
  }catch(parameter needed){ handle error  
   handle the errors founded….  
  }finally{ it always works if error caught or not  
   always show codes which are written here….  
  }
* Error Handling | throw keyword :-  
  \* throw “ message we need to show ”  
  note : throw keyword needs to use in the try keyword.  
  Example :-  
  html file:  
  <p>Enter number between 5 - 10</p>  
  <input type="text" id="numberText">  
  <button id="checkButton">Check</button>  
  <p id="message"></p>  
  js file :  
  var btn = document.querySelector("#checkButton");  
  var input = document.querySelector("#numberText");  
  btn.addEventListener("click", function () {  
   var num = input.value;  
   try { if (num < 5) {  
   throw "The input value is too low. Please enter value from 5 - 10."; }  
   else if (num > 10) {  
   throw "The input value is too high. Please enter value from 5 - 10."; }  
   else{throw "The input value is valuable."; }  
   } catch (error) {  
   document.querySelector("#message").innerHTML = error; } });
* Canvas | drawing graphics on webpage :-  
  Example :- ( Bangladesh flag drawing program )  
  html file :  
  <canvas id="myCanvas" width="400px" height="300px"></canvas>  
  js file :  
  var c = document.querySelector("#myCanvas");  
  var ctx = c.getContext("2d");  
  ctx.fillStyle = "green";  
  ctx.fillRect(12, 12, 376, 276);  
  ctx.lineWidth = 2;  
  ctx.strokeStyle = "black";  
  ctx.strokeRect(10, 10, 380, 280);  
  var centerX = c.width / 2;  
  var centerY = c.height / 2;  
  ctx.beginPath();  
  ctx.arc(centerX, centerY, 80, 0, 2 \* Math.PI, false);  
  ctx.fillStyle = "red";  
  ctx.fill();  
  ctx.stroke();  
    
    
    
  Colors, Styles, and Shadows

|  |  |
| --- | --- |
| **Property** | **Description** |
| [fillStyle](https://www.w3schools.com/tags/canvas_fillstyle.asp) | Sets or returns the color, gradient, or pattern used to fill the drawing |
| [strokeStyle](https://www.w3schools.com/tags/canvas_strokestyle.asp) | Sets or returns the color, gradient, or pattern used for strokes |
| [shadowColor](https://www.w3schools.com/tags/canvas_shadowcolor.asp) | Sets or returns the color to use for shadows |
| [shadowBlur](https://www.w3schools.com/tags/canvas_shadowblur.asp) | Sets or returns the blur level for shadows |
| [shadowOffsetX](https://www.w3schools.com/tags/canvas_shadowoffsetx.asp) | Sets or returns the horizontal distance of the shadow from the shape |
| [shadowOffsetY](https://www.w3schools.com/tags/canvas_shadowoffsety.asp) | Sets or returns the vertical distance of the shadow from the shape |

|  |  |
| --- | --- |
| **Method** | **Description** |
| [createLinearGradient()](https://www.w3schools.com/tags/canvas_createlineargradient.asp) | Creates a linear gradient (to use on canvas content) |
| [createPattern()](https://www.w3schools.com/tags/canvas_createpattern.asp) | Repeats a specified element in the specified direction |
| [createRadialGradient()](https://www.w3schools.com/tags/canvas_createradialgradient.asp) | Creates a radial/circular gradient (to use on canvas content) |
| [addColorStop()](https://www.w3schools.com/tags/canvas_addcolorstop.asp) | Specifies the colors and stop positions in a gradient object |

## Line Styles

|  |  |
| --- | --- |
| **Property** | **Description** |
| [lineCap](https://www.w3schools.com/tags/canvas_linecap.asp) | Sets or returns the style of the end caps for a line |
| [lineJoin](https://www.w3schools.com/tags/canvas_linejoin.asp) | Sets or returns the type of corner created, when two lines meet |
| [lineWidth](https://www.w3schools.com/tags/canvas_linewidth.asp) | Sets or returns the current line width |
| [miterLimit](https://www.w3schools.com/tags/canvas_miterlimit.asp) | Sets or returns the maximum miter length |

## Rectangles

|  |  |
| --- | --- |
| **Method** | **Description** |
| [rect()](https://www.w3schools.com/tags/canvas_rect.asp) | Creates a rectangle |
| [fillRect()](https://www.w3schools.com/tags/canvas_fillrect.asp) | Draws a "filled" rectangle |
| [strokeRect()](https://www.w3schools.com/tags/canvas_strokerect.asp) | Draws a rectangle (no fill) |
| [clearRect()](https://www.w3schools.com/tags/canvas_clearrect.asp) | Clears the specified pixels within a given rectangle |

## Paths

|  |  |
| --- | --- |
| **Method** | **Description** |
| [fill()](https://www.w3schools.com/tags/canvas_fill.asp) | Fills the current drawing (path) |
| [stroke()](https://www.w3schools.com/tags/canvas_stroke.asp) | Actually draws the path you have defined |
| [beginPath()](https://www.w3schools.com/tags/canvas_beginpath.asp) | Begins a path, or resets the current path |
| [moveTo()](https://www.w3schools.com/tags/canvas_moveto.asp) | Moves the path to the specified point in the canvas, without creating a line |
| [closePath()](https://www.w3schools.com/tags/canvas_closepath.asp) | Creates a path from the current point back to the starting point |
| [lineTo()](https://www.w3schools.com/tags/canvas_lineto.asp) | Adds a new point and creates a line to that point from the last specified point in the canvas |
| [clip()](https://www.w3schools.com/tags/canvas_clip.asp) | Clips a region of any shape and size from the original canvas |
| [quadraticCurveTo()](https://www.w3schools.com/tags/canvas_quadraticcurveto.asp) | Creates a quadratic Bézier curve |
| [bezierCurveTo()](https://www.w3schools.com/tags/canvas_beziercurveto.asp) | Creates a cubic Bézier curve |
| [arc()](https://www.w3schools.com/tags/canvas_arc.asp) | Creates an arc/curve (used to create circles, or parts of circles) |
| [arcTo()](https://www.w3schools.com/tags/canvas_arcto.asp) | Creates an arc/curve between two tangents |
| [isPointInPath()](https://www.w3schools.com/tags/canvas_ispointinpath.asp) | Returns true if the specified point is in the current path, otherwise false |

## Transformations

|  |  |
| --- | --- |
| **Method** | **Description** |
| [scale()](https://www.w3schools.com/tags/canvas_scale.asp) | Scales the current drawing bigger or smaller |
| [rotate()](https://www.w3schools.com/tags/canvas_rotate.asp) | Rotates the current drawing |
| [translate()](https://www.w3schools.com/tags/canvas_translate.asp) | Remaps the (0,0) position on the canvas |
| [transform()](https://www.w3schools.com/tags/canvas_transform.asp) | Replaces the current transformation matrix for the drawing |
| [setTransform()](https://www.w3schools.com/tags/canvas_settransform.asp) | Resets the current transform to the identity matrix. Then runs [transform()](https://www.w3schools.com/tags/canvas_transform.asp) |

## Text

|  |  |
| --- | --- |
| **Property** | **Description** |
| [font](https://www.w3schools.com/tags/canvas_font.asp) | Sets or returns the current font properties for text content |
| [textAlign](https://www.w3schools.com/tags/canvas_textalign.asp) | Sets or returns the current alignment for text content |
| [textBaseline](https://www.w3schools.com/tags/canvas_textbaseline.asp) | Sets or returns the current text baseline used when drawing text |

|  |  |
| --- | --- |
| **Method** | **Description** |
| [fillText()](https://www.w3schools.com/tags/canvas_filltext.asp) | Draws "filled" text on the canvas |
| [strokeText()](https://www.w3schools.com/tags/canvas_stroketext.asp) | Draws text on the canvas (no fill) |
| [measureText()](https://www.w3schools.com/tags/canvas_measuretext.asp) | Returns an object that contains the width of the specified text |

## Image Drawing

|  |  |
| --- | --- |
| **Method** | **Description** |
| [drawImage()](https://www.w3schools.com/tags/canvas_drawimage.asp) | Draws an image, canvas, or video onto the canvas |

## Pixel Manipulation

|  |  |
| --- | --- |
| **Property** | **Description** |
| [width](https://www.w3schools.com/tags/canvas_imagedata_width.asp) | Returns the width of an ImageData object |
| [height](https://www.w3schools.com/tags/canvas_imagedata_height.asp) | Returns the height of an ImageData object |
| [data](https://www.w3schools.com/tags/canvas_imagedata_data.asp) | Returns an object that contains image data of a specified ImageData object |

|  |  |
| --- | --- |
| **Method** | **Description** |
| [createImageData()](https://www.w3schools.com/tags/canvas_createimagedata.asp) | Creates a new, blank ImageData object |
| [getImageData()](https://www.w3schools.com/tags/canvas_getimagedata.asp) | Returns an ImageData object that copies the pixel data for the specified rectangle on a canvas |
| [putImageData()](https://www.w3schools.com/tags/canvas_putimagedata.asp) | Puts the image data (from a specified ImageData object) back onto the canvas |

## Compositing

|  |  |
| --- | --- |
| **Property** | **Description** |
| [globalAlpha](https://www.w3schools.com/tags/canvas_globalalpha.asp) | Sets or returns the current alpha or transparency value of the drawing |
| [globalCompositeOperation](https://www.w3schools.com/tags/canvas_globalcompositeoperation.asp) | Sets or returns how a new image are drawn onto an existing image |

## Other

|  |  |
| --- | --- |
| **Method** | **Description** |
| save() | Saves the state of the current context |
| restore() | Returns previously saved path state and attributes |
| createEvent() |  |
| getContext() |  |
| toDataURL() |  |

**JavaScript** **ES6** version (2015) :-

* Variable and function declaration :-  
  ES6 : ECMAScript 6  
  ECMAScript was created to standardize JavaScript, and **ES6** is the 6th version of ECMAScript, it was published in 2015, and is also known as ECMAScript 2015.  
  \* New variable declaration in ES6 :-  
   let / const (we can’t change it after declaring a variable)  
  Example :- (ES5)  
  -> var x = 6;  
  if (true){  
   var x = 5;  
  } console.log(x);  
  - output : 5  
  -> let x = 6; (ES6)  
  if (true){  
   let x = 5;  
  } console.log(x);  
  - output : 6   
  -> const x = 6; (ES6)  
  if (true){  
   x = 5;  
  } console.log(x);  
  - output : It shows the error.  
  \* To show the output in ES6 we need to use (` `) – backtick sign in the console.log() or document.write() and avoid to use (“ ”) – Double quotes.  
  \* To concat a string to another in ES6 we don’t need (+)– plus sign anymore. We just need to write the variable name into the {} – curly brackets with a ($) – dollor sign. Example : console.log(`The sum is : ${sum}`);  
  And note that we don’t need any sign to add string to another just maintain the rule of writing variable names.  
  \* Function declaration in ES6 :-  
  const function\_name = (parameter) => {  
   codes to be executed here…  
  }
* Hoisting and Strict mode :-  
  \* Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).  
  Example in var declaration :- it works correctly.  
  x = 20;  
  console.log(x);  
  var = x;   
  Example in let declaration :- it shows the reference error.  
  x = 20;  
  console.log(x);  
  let = x;   
  Example in const declaration :- it shows the syntax error.  
  x = 20;  
  console.log(x);  
  const = x;  
  \* Strict :-  
  To use strict mode we need to use a string - “use strict” at the top of our script area. It mainly secured our js codes.  
  - Strict mode makes it easier to write "secure" JavaScript.  
  - Strict mode changes previously accepted "bad syntax" into real errors.
* Default and Rest parameter :-  
  \* default parameter :-  
  (text = "Hello I am learning js")  
    
  parameter this the default parameter value  
  If the user don’t entry any value into it will show the default value of its parameter.  
  \* rest parameter :-  
  If we create a function with some parameter but we don’t know how much data users use here then we just can do one thing we can put here some extra parameter.  
  example : (numbers = x, y, …z)  
   rest parameter
* Spread operator :-  
  \* …variable is a spread operator. Spread operator and rest operator is used by same symbol.  
  \* Difference between spread operator and rest parameter :-  
  1. We can use spread operator in everywhere, but can’t use rest parameter in everywhere, we can just use it as a last parameter.  
  2. Rest parameter packed the values in an array, but spread operator unpacked the values.  
  Example :- (concatenate objects using spread operator)  
  let number1 = [1, 4, 6];  
  let number2 = [2, 3, 5];  
  let number = [...number1, ...number2];  
  document.write(`The number collection : ` + number);
* Object literals :-  
  \* object writing in ES6 :  
  Example :  
  function Info(name, age){  
   return{  
   name,  
   age Object in ES6  
   }  
  }  
    
  To write a function under a object :-  
  Example :- (default rule of declaring function in object)  
  let message = {  
   body : function (){  
   return `I am learning js`  
   }  
  } console.log(message.body());  
  Example :- (ES6 function in object)  
  let message = {  
   body(){ (concise syntax)  
   return `I am learning js`  
   }  
  } console.log(message.body());  
  Note : we can create a function\_name using space between the name like an example : ‘my function’ ,  
  and an important thing is that we must use single quotation(‘ ’) not double quotation.
* for…of and for…in loop :-  
  \* for…of syntax :-  
  for(let variable of iterable){  
   code block to be executed here…  
  }  
  \* Variable can be declared with let, const and var.  
  \* iterable - An object that has iterable properties.  
    
  Example :-  
  var names = [“s1”, “s2”, “s3”]  
  for(let name of names){  
   console.log(name);  
  }  
    
  \* for…in syntax :-  
  for(let variable in iterable){  
   code block to be executed here…  
  }  
  Example :-  
  var students = {  
   ID:1023,Name: “Rahim”,GPA:4.50  
  }  
  for(let x in students){  
   document.write(x);  
  } output will be the property  
  => to show the value of the property :-  
   document.write(student[x]);
* for and forEach :-  
  \* for loop syntax :  
   for (initialization(starting), condition, update(++/--)) {  
   type here what to print ….}  
  \* forEach loop syntax :  
  array.forEach( function(currentValue(required), index, arr), thisValue)  
  Example :- (forEach with an anonymous function)  
  var numbers = [10,20,30,40]  
  numbers.forEach(function(x){   
   console.log(x);  
  });  
  Output : 10,20,30,40  
  Example 1 :- (square all number in the array numbers)  
  var numbers = [2,4,6,8];  
  var squareNumber = [];  
  numbers.forEach(function(x){  
   squareNumber.push(x\*x);   
  });  
  document.write(squareNumber);   
  Output : 4,16,36,64  
  Example 2 :- (to add 5 with all the array index)  
  var numbers = [10,20,30,40];  
  numbers.forEach(function(x,index,array){  
   array[index] = x+5;  
  });  
  document.write(numbers);  
  Output : 15,25,35,45
* map(), filter() function :-  
  \* map syntax :-  
  array.map(function(currentValue(required), index, array), thisValue );  
  note : map() return values in an array. We don’t even need to create an array to put the objects in it.  
  Example :-  
  var numbers = [10,20,30,40]  
  var newNumbers = numbers.map(myFunction)  
  function myFunction(x){  
   return x\*x;  
  }  
  document.write(newNumbers);   
  \* filter syntax :-  
  array.filter(function(currentValue(required), index, array), thisValue );  
  note : filter() return values in an array, but there need a condition for it to return the value.  
  Example :-  
  var numbers = [10,20,30,40,15,45]  
  var newNumbers = numbers.filter(myFunction)  
  function myFunction(x){  
   return x>20;  
  }  
  document.write(newNumbers);
* Arrow Function :-  
  const function\_name = (parameter) => {  
   codes to be executed here…  
  }  
  \* If the function has only one statement, and the statement returns a value, you can remove the brackets  and the return keyword.  
  => const myFunction = () => “Hello world”;  
  \* if you have only one parameter, you can skip the parentheses ‘()’ as well.  
  => const myFunction = val => “Hello world”;
* Arrow with map(), filter() :-  
  \* It can make long codes to very simple.  
  Example :-  
  var studentInfo = [  
   {Id: 13723,Name: "Rahim",GPA: 4.50},  
   { Id: 34769,Name: "Kamal",GPA: 4.36},  
   {Id: 47839,Name: "Karim",GPA: 5.00},  
   {Id: 64353,Name: "Bijoy",GPA: 4.00} ]  
  const studentInformation = () => {  
   return studentInfo.filter((x) => { return x.GPA > 4.00}).map((y) => { return y.Name}) }
* Destructuring array and objects :-  
  => Array destructure :-  
  Example :-  
  let numbers = [10,20,30,40,50];  
  let [num1, num2, num3, num4, num5] = numbers;  
  document.write(`${num4} `);  
  document.write(`${num2}`);  
  => swap destructure :-  
  Example :-  
  let a = 10;  
  let b = 30;  
  [a, b] = [b, a];  
  document.write(`${b} `);  
  document.write(`${a} `);  
  => object destructure :-  
  Example :-  
  let studentInfo = { ID : 38573,Name : "Rahim",Age : 26 }  
  const {ID, Name, Age} = studentInfo;  
  document.write(`${ID} | ${Name} | ${Age}`);  
  => nested object destructure :-  
  Example :-  
  let studentinfo = {  
   student1 : { ID : 43856,Name : "Rahim",GPA : 4.50},  
   languages: { native : 'Bangla',beginner : 'English'} }  
  const {student1, languages} = studentinfo;  
  document.write(student1.Name, student1.GPA);  
  document.write(languages.native);  
  console.log(languages.native);
* Array methods - find() | findIndex() :-  
  \* find() method :-  
  find(callback, value) – return the first value of the first element that pass certain condition.  
  Example :-  
  let numbers = [13, 5, 14, 55, 7]  
  let evenNumberValue = numbers.find(x => x % 2 === 0);  
  document.write(` The given numbers are : ${numbers} <br>`);  
  document.write(` The first Even number is : ${evenNumberValue} <br>`);  
  \* findIndex() method :-  
  findindex(callback, value) – return the first index of the first element that pass certain condition.  
  Example :-  
  let numbers2 = [13, 8, 15, 43, 9]  
  let evenNumberValue2 = numbers2.findIndex(x => x % 2 === 0);  
  document.write(` The given numbers are : ${numbers2} <br>`);  
  document.write(` The first Even number index is : ${evenNumberValue2 + 1}<br>`);
* String methods : startsWith(), endsWith(), includes()  
  \* all these methods are case sensitive.  
  => string.startsWith() :-  
  startsWith(searchString, position(optional) ) – check a string starts with another string.  
  Example :-  
  const message = "Hello I am learning js";  
  document.write(`${message.startsWith('Hello')} <br>`);  
    
  => string.endsWith() :-   
  startsWith(searchString, length(optional) ) – check a ends starts with another string.   
  Example :-  
  const message = "Hello I am learning js";  
  document.write(`${message.startsWith('Hello')} <br>`);  
    
  => string.includes() :-   
  startsWith(searchString, position(optional) ) – check if a string contains another string.   
  Example :-  
  const message = "Hello I am learning js";  
  document.write(`${message.startsWith('Hello')} <br>`);
* ES6 modules( export/ import ) :-  
  \* To export and import js codes we need two files, one for export and other one for import, which needs to linked up with html file like that -   
  <script src= “index.js” type= “module”></script>  
    
   important for export and import  
  Example :-  
  export file:   
  export let text = "Welcome to module";  
  import file :  
  import {text} from './myModule.js'  
  console.log(text);  
  note : we can change variable name from export and import files – {text as message}
* ES6 class :-  
  \* To use class in ES6 need class keyword :-  
  syntax : class className {  
   constructor () { … }  
   }  
  => to use class we always need constructor() method :  
  Example :-  
  class Car {  
   constructor (name, year) {  
   this.name = name;  
   this.year = year;  
   }  
  }
* Synchronous vs Asynchronous :-  
  **\*** Synchronous **: JavaScript** is **synchronous**. This means that it will execute your code block by order after hoisting.  
  **\*** Asynchronous **:** It is not a blocked code function, it will always show output if there were errors or not, and you also can set a time.
* Callback and higher order function :-  
  \* A **callback function** is a **function** that is passed as an argument to another **function**, to be “called back” at a later time. A **function** that accepts other **functions** as arguments is called a **higher**-**order function**, which contains the logic for when the **callback function** gets executed.  
  Example :-  
  const square = (x) => {  
   document.write(`Square of ${x} : ${x \* x} <br>`); }  
  square(4);  
  const higherOrderFunction = (num, callback) => {  
   callback(num); }  
  higherOrderFunction(6, square);
* ES6 promise :-  
  \* how to create a promise |  
  steps : pending,resolved,reject  
  syntax :  
  const variable = new Promise ( (resolve, reject) => {  
   blocks to be executed here… } );  
  Example :- (part-1 : promise crating part)  
  const promise1 = new Promise((resolve, reject) => {  
   let completedPromise = true;  
   if (completedPromise) {  
   resolve(`Complete promise 1 <br>`);  
   } else {  
   reject(`Not completed promise 1 <br>`);  
   }  
  });  
  => follow the following part to use this promises…  
  \* how to use a promise  
  Example :- (part-2 : promise display part)  
  promise1  
   .then((res) => {  
   document.write(`${res}`);  
   })  
   .catch((error) => {  
   document.write(`${error}`);  
   });  
  \* how to run multiple promise - all()  
  Example :-   
  const promise1 = new Promise((resolve, reject) => {  
   let completedPromise = true;  
   if (completedPromise) {  
   resolve(`Complete promise 1 <br>`);  
   } else {  
   reject(`Not completed promise 1 <br>`);  
   }  
  });  
  Promise.all([promise1, promise2])  
  .then(([res1, res2]) => document.write(res1, res2));  
  \* race()  
  Example :-  
  const promise3 = new Promise ((resolve, reject) => {  
   setTimeout(() => {  
   resolve ("This is promise 3");  
   }, 2000);  
  });  
  const promise4 = new Promise ((resolve, reject) => {  
   setTimeout(() => {  
   resolve ("This is promise 4");  
   }, 1000);  
  });  
  Promise.race([promise3, promise4]).then((response) => document.write(response));  
  \* promise chaining :-  
  Example :-  
  const taskOne = () => {  
   return new Promise((resolve, reject) => {  
   resolve(`Task 1 is completed <br>`)  
   }); };  
  const taskTwo = () => {  
   return new Promise((resolve, reject) => {  
   setTimeout(() => {  
   resolve(`Task 2 is completed <br>`)  
   }, 2000); }); };  
  const taskThree = () => {  
   return new Promise((resolve, reject) => {  
   reject(`Task 3 is not completed <br>`)  
   }); };  
  const taskFour = () => {  
   return new Promise((resolve, reject) => {  
   resolve(`Task 4 is completed <br>`)  
   }); };  
  taskOne()  
   .then((response) => document.write(response))  
   .then(taskTwo)  
   .then((response) => document.write(response))  
   .then(taskThree)  
   .then((response) => document.write(response))  
   .then(taskFour)  
   .then((response) => document.write(response))  
   .catch((error) => document.write(error));
* ES7 Async and Await :-  
  Example :-  
  const taskOne = () => {  
   return new Promise((resolve, reject) => {  
   resolve(`Task one is completed <br>`);  
   }); };  
  const taskTwo = () => {  
   return new Promise((resolve, reject) => {  
   setTimeout(() => {  
   resolve(`Task two is completed <br>`);  
   }, 2000);  
   }); };  
  const taskThree = () => {  
   return new Promise((resolve, reject) => {  
   reject(`Task three is not completed <br>`);  
   }); };  
  const taskFour = () => {  
   return new Promise((resolve, reject) => {  
   resolve(`Task four is completed <br>`);  
   }); };  
    
  async function myFunction() {  
   try {  
   const t1 = await taskOne();  
   document.write(t1);  
   const t2 = await taskTwo();  
   document.write(t2);  
   const t3 = await taskThree();  
   document.write(t3);  
   const t4 = await taskFour();  
   document.write(t4);  
   } catch (error) {  
   document.write(error);  
   } };  
  myFunction();
* Note : To get easier codes for callback() there comes promise() function and to get a lot easier code there comes another function named async and await function.