# JAVASCRIPT

1. What is JavaScript ?

* A web-page programming language.
* Add interactive behaviour to webpages.
* Build web and mobile applications.
* Create command line tools.

1. Comments?

* Single line comments - //
* multi line comments - /\*comment here\*//

1. Variables in JavaScript?

* A variable is container for storing data.

Var 🡺 The most common variable. Can be reassigned but only accessed within a function. Variables defined with Var move to top when code is executed.

Const 🡺 Cannot be reassigned and not accessible before they appear within code.

Let 🡺 Similar to Const , however let variable can be reassigned but not re-declared.

1. DATA TYPES: -

Numbers 🡪 var age = 23

Strings 🡪 var Name = “Naveen”

Boolean 🡪 var a = True

var a = False

1. Including JavaScript in an HTML Page: -

<script type = “text/JavaScript”>

//JS code goes here

</script>

1. Call an External JavaScript File: -

<script src=” myscript.js”><script>

1. OPERATORS :- To perform the mathematical operations such as addition , subtraction , division , multiplication ,etc....

|  |  |
| --- | --- |
| BASIC OPERATORS | |
| Addition (+) | It performed addition operation. |
| Subtraction (-) | It performed subtraction operation. |
| Multiplication  (\*) | It performed multiplication operation. |
| Division (/) | It will store the quotient. |
| Grouping operator (….) | Operation within brackets is executed earlier than those outside. |
| Modulus (%) | It will store the remainder. |
| Increment ++ | It will increment number. |
| Decrement -- | It will decrement number. |

|  |  |
| --- | --- |
| COMPARISON OPERATORS | |
| == | Equal to |
| === | Equal value and Equal type |
| != | Not equal |
| !== | Not equal value or not equal type. |
| > | Greater than |
| < | Lesser than |
| >= | Greater than or equal to |
| <= | Lesser than or equal to |

|  |  |
| --- | --- |
| LOGICAL OPERATORS | |
| Logical and  (&&) | Both conditions must be true |
| Logical OR  (||) | Either conditions can be true |
| Logical Not  (!) | Typical used to reverse a condition’s Boolean value.  (true 🡪 false and false 🡪 True) |

|  |  |
| --- | --- |
| BITWISE OPERATORS | |
| & | Bitwise and |
| | | Bitwise or |
| ~ | Not |
| ^ | Xor |
| << | Left shift |
| >> | Right shift |
| >>> | Zero fill right shift |

1. Operator Precedence :-

* Parentheses ()
* Exponents
* Multiplication and division
* Addition and subtraction

1. Outputting Data: -

|  |  |
| --- | --- |
| Alter () | Output data in an alert box in the browser window. |
| Confirm () | Opens up a yes/no dialog and return true / false depending on user click. |
| Console.log () | Writes information to the browser console, good for debugging purpures |
| Document. Write () | Write directly to the HTML document |
| Prompt () | Creates a dialogue for user input |

1. Math Methods: -

|  |  |
| --- | --- |
| abs (x) | Returns the absolute (positive) value of x |
| acos (x) | The arccosine of x, in radians. |
| asin (x) | The arcsine of x, in radians. |
| atan (x) | The arctangent of x, in radians. |
| atan2 (x,y) | Arctangent of the quotient of its arguments. |
| Ceil (x) | Value of x rounded up to its nearest integer. |
| Cos (x) | The cosine of x, in radians. |
| Exp (x) | Value of Exponential |
| Log (x) | The natural logarithm (base E) of x |
| Max (x,y,z) | Its find’s the maximum value of the numbers |
| Min (x,y,z) | Its find’s the minimum value of the numbers. |
| Pow (x,y) | X to the power of y |
| Random () | Return a random number between 0 and 1. |
| Round (x) | The value of x rounded to its nearest integer |
| Floor (x) | The value of x rounded down to its nearest integer |
| Sin (x) | The sine of x, in radians |
| Sqrt (x) | Square root of x |
| Tan (x) | The tangent of an angle. |

1. Math Properties: -

|  |  |
| --- | --- |
| E | Euler’s number |
| LN2 | The natural logarithm of 2 |
| LN10 | Natural Logarithm of 10 |
| LOG2E | Base 2 Logarithm of E |
| LOG10E | Base 10 Logarithm of E |
| PI | The number PI |
| SQRT1\_2 | Square root of ½ |
| SQRT2 | Square root of 2 |

1. TYPE CONVERSIONS: -

* change the datatype of a value to another (strings, Numbers, Boolean).

1.Strings 🡺 Numbers

age = “21”

age=Number(age)

2.Numbers 🡺 strings

age = 21

age = string(age)

3.Boolean 🡺 True / false

* If Boolean is empty, it will display the false otherwise if any value will be found it will display true like Boolean(“Naveen”)
* If we want to check the type of variable to use “type of”.

1. STRING METHODS: -

|  |  |  |
| --- | --- | --- |
| charAt() | Return a character at specified position inside a string. | Var txt =”Hello World”;  Console.log(txt.charAt(0));  //prints ‘H’ |
| Concat() | Concatenates (joins) two or more string into one. | Var str1 = “Hello”;  Var str2 =”World”;  Console.log(str1.concat(str2));  //prints “Hello World” |
| Indexof() | Provides the position of the first occurrence of a specified within a string. | Var txt =”Let’s find where pen occurs!”;  Console.log(txt.indexof(“pen”));  //prints 17. |
| LastIndexOf() | Same as indexof () but with the last occurrence, searching backwards | Var txt = “A dev knows a dev”;  Console.log(txt.lastindexOf(“dev”));  //prints 14 |
| Replace() | Find the replace specified text in a string. | Var txt =”Hello dev!”;  Console.log(txt.replace(“dev”, “World”));  //prints “Hello World” |
| Search () | Executes a search for a matching text and returns its position at start | Var str = “hello dev!”  Console.log(str.search(“dev”));  //prints 6 |
| Slice (start ,end) | Extracts a section of a string and returns it as a new string without modifying the original string. | Var str = “Developers world!”;  Console.log(str.slice(0,10));  //prints “Developers” |
| TolowerCase () | Convert strings to lower case | Var txt = “Coding”  Console.log(txt.touppercase());  //prints “CODING” |
| ToUpperCase () | Convert strings to upper case | Var txt = “CODING”  Console.log(txt.tolowercase());  //prints “coding” |
| ValueOf() | Returns the primitive value (that has no properties or methods) of a string object. |  |
| Trim() | It will remove an empty space of the string . | Var str = “ Trim Both side ”  Console.log(str.trim());  //prints Trim Both Side |
| tostring | It will convert the number into string. | Var str = 15;  Console.log(str.tostring());  //prints “15” |
| Includes() | It will check the value or string is present or not. | Var str = “My name is Naveen” ;  Console.log(str.includes(“name”));  //prints true |
| CharcodeAt() | Gives you the unicode of character at the position | Var str = “TEST”  Console.log(str.charcodeAt(0)) ;  // returns 84. |
| Match () | Retrieves the matches of a string against a search pattern. It match pattern it will display the //g and //gi | Var str=”loopers loppers loopers ”  Console.log(str.match(/ers/g))  //print [“ers” , ‘’ers” , “ers”] |
| formCharCode () | Return a string created from the specified sequence to UTF-16 code units. |  |
| Split (separator, limit ) | A string object into array of strings at a specified position. If we have (‘’) , it will separate a string in array and if we have (‘ ’) , it will separate word to as string. If we have ( ) it will consider the entire string as array. | Var str=”1,2,3,4,5”;  Console.log(str.split(“,”));  //prints [“1” , “2” , ”3” , “4” , “5”] |
| Length () | It will display the length of the string which include space character. | Var str=”Naveen”  Console.log(str.length())  //prints “6” |
| ReplaceAll() | It will replace all the position of string. | Var number=”123-456-789”  Console.log(number.replaceall(“-”,”/”))  //prints “123/456/789” |
| Substring ( start , length) | Also similar to slice () but can’t accept negative indices. | Var name = “Naveen”  name. Substring(1)  Console.log(name)  //prints aveen |

1. METHOD CHANGING :- calling one method after another method in one condition line of code.

EX :- let username= “naveen”

let letter = username. charAt(0). touppercase().

Console.log(letter)

1. CONDITIONAL STATEMENTS :-

If statement = a basic form of decision making if a condition is true, then do something if not, then don’t do it !

Syntax :-

If (condition){

Statements ;

}

If-else statement = a basic form of decision making if a condition is true then do something and if a condition is false then also do something.

Syntax: -

If (condition) {

Statements.

}

else {

Statements.

}

else if statements = it will check multiple conditional statements.

Syntax: -

If (condition) {

Statements.

}

else if (condition) {

Statements.

}

else {

Statements.

}

Nested loop: -a loop inside another loop is called nested loop

1. SWITCHES :- Statement that examines a value for match against many case clauses. More efficient that many “else if” statements.

Example :-

Let grade = “A”

Switch (grade){

case “A”:

console.log(“you did great”)

break.

case “B” :

console.log(“you did good”)

break.

default:

console.log(grade ,” is not letter grade”)

}

1. ITERATIVE CONTROLS :-

WHILE LOOP :- Repeat some code. While some conditions is true potentially infinite. Setup conditions under a loop executes.

EX :-

Let iteration = 0 ;

While(iteration < 5){

Iteration++

Console.log(iteration)

}

//prints 1 2 3 4 5

DO-WHILE :-

* Do something , then check the condition , repeat if condition is true.

EX:-

Let iteration = 0;

do{

iteration++;

console.log(iteration)

}while(iteration < 5)

//prints 1 2 3 4 5

FOR LOOP :- repeat some code a certain amount of time .

SYNTAX :-

For (before loop ; condition for loop ; after loop){

//statements

}

EX :-

For (let i=o ; i < 5 ; i++){

Console.log(i)

}

//prints 0 1 2 3 4

FOR….IN LOOP :- It will fetch the index values of array or keys of an object .

Let arr = [3,5,7]

For(let i in arr){

Console.log(i)

}

//prints 0 1 2

Let arr =[3,5,7]

For(let i in arr){

Console.log(i, arr[i] )

}

//prints 0 3 , 1 5 ,2 7

FOR….OF LOOP :- It will fetch the values of array or values of an object.

Let arr=[3,5,7]

For (let i of arr){

Console.log(i)

}

//prints 3 5 7

1. BREAK :- breaks out of a loop entirely. Used to stop and exit the cycle at certain condition.

For (let i =1 ; i<=20 ; i+=1){

if(i==13){

break.

}

console.log(i)

}

//prints 1 2 3 4 5 6 7 8 9 10 11 12

1. CONTINUE :- skip parts of the cycle if certain conditions are met.

For (let i =1 ; i<=20 ; i+=1){

if(i==13){

continue;

}

console.log(i)

}

//prints 1,2,3,4,5,6,7,8,9,10,11,12,14,15,16,17,18,19,20.

1. NESTED LOOP :-a loop inside of another loop.

for(let i=1 ; i<=3;i++){

for(let j=1 ; j<=3 ; j++){

console.log(j)

}

}

1. FUNCTIONS :- define code once and use it many times to perform some code , call the function name.

Return :- return a value back to perform to the place where are invoked a function.

Named function :-

Function myfunction(){

Console.log(“codenature”);

}

Named function with arguments :-

Function myfunction(parameter1){

Console.log(parameter1)

}

myfunction(10)

//prints 10

Anonymous function :- A function which is attached to the variable is known as Anonymous function.

Let myfunction = function(){

Console.log(“codenature”)

}

Anonymous function with arguments :-

Let myfunction = function(parameter1){

Console.log(parameter1);

}

Myfunction(10)

//prints 10

1. TERNARY OPERATOR :- Shorthand for an “if else ” statement . Take 3 operands such as

SYNTAX :- condition ? Expression if true : Expression if false

1. VARIABLE SCOPE :- where a variable is accessible .

* Let – variable are limited to block scope { }
* Var – variable are limited to a function () { }
* Global variables is declared outside any function (if global , var will change browser’s window properties )

1. TEMPLATE LITERALS :- delimited with (‘) instead of double or single quotes allows embedded variables and expressions.

Example :-

Let username = “Bro”.

Let items =3.

Console.log(“Hello ${username}”).

Console.log(“you have ${items} items in your cart”);

// Hello Bro

You have 3 items in your cart.

1. TOLOCALSTRING :-returns a string with a language sensitive representation of this number.

SYNTAX :-

Number.tolocalestring(locale,{options}).

locale = specify the language (undefined = default set browser)

options = objects with formatting options

Example :-

Let mynum = 123456.789

mynum = mynum.tolocalestring(“en-US”) //123,456.789 //US English

mynum = mynum.tolocalestring(“hi-IN”) //1,23,456.789 //Hindi

mynum = mynum.tolocalestring(“de-DE”) //123.456,789 //standard German

mynum = mynum.tolocalestring(“en.us”,{style: "currency”, currency : “USD”}) //$123,456.789

Example :-

Let mynum= 1;

mynum = mynum.tolocalestring(undefined,{style : “percent”}) //100%

mynum = mynum.tolocalestring(undefined,{style : “unit” , unit :”celsius”}) //1oC

1. ARRAY :- think of it as a variable that can store multiple values and each of the elements separated by comma within a square brackets. But the index of the elements starts with zero indexing.

|  |  |  |
| --- | --- | --- |
| Push() | Add new element at the ending position | [1,2,3,4].push(4) 🡪 [1,2,3,4] |
| Pop() | Remove the last element of an array. | [1,2,3].pop() 🡪 [1,2] |
| Unshift() | Add new element at starting position. | [1,2,3].unshift(0)🡪[0,1,2,3] |
| Shift() | Removes a first element of an array. | [1,2,3].shift()🡪[2,3] |
| Concat() | Join several arrays into one array. | [‘a’ ,’b’].concat(‘c’)🡪 [‘a’,’b’,’c’] |
| length | To find the length of an array. | [1,2,3].length 🡪3 |
| Indexof() | Returns the primitive value of the specified object. | [1,2,3].indexof(2) 🡪1 |
| lastIndexof() | Gives the last position at which a gives element appears in an array. | [1,2,2,3].lastIndexof(2)🡪2 |
| Sort() | Sort elements alphabetically order or Numbers | [2,3,1].sort() 🡪 [1,2,3] |
| Reverse() | Sort elements in descending order | [1,2,3].reverse() 🡪[3,2,1] |
| Slice() | Pulls a copy of a portion of an array into a new array. | [1,2,3,4,5].slice(1,4) 🡪[2,3,4] |
| Join() | Combines elements of an array into single string and returns the string. | [‘a’,’b’,’c’].join(‘-’)🡪a-b-c |
| At() | It will fetch the value based on the index position the index either negative or positive. | [2,3,4,5,6].at(-2) 🡪5 |

1. LOOP AN ARRAY :-

Let prices = [5,10,15,20,25]

For (let i = 0 ; i < prices.length ; i++ ){

Console.log(price[i]);

} //5 10 15 20 25

Let prices = [5,10,15,20,25]

For (let i = price.length-1 ; i >=0 ; i-=1 ){

Console.log(price[i]);

} //25 20 15 10 5

Let prices =[5,10,15,20,25]

For (let i of prices){

Console.log(i)

} //5 10 15 20 25

1. 2D ARRAY **:-** An array of arrays is known as 2D array.

Example :-

Let fruits =[“apple” , “orange” , “bananas”];

Let meats = [“eggs”, “chicken”, ”fish”];

Let vegetables = [“carrots“ , “onions” , “potatoes”];

Let grocery = [fruits , vegetables , meats];

for (let list of grocery){

Console.log(list).

}

output :-

[“apple” , “orange” , “bananas”]

[“eggs”, “chicken”, ”fish”]

[“carrots“ , “onions” , “potatoes”]

for (let list of grocery){

for(let food of list){

console.log(food);

}

}

//apple orange bananas eggs chicken fish carrots onions potatoes.

* Access of the 2D array based on the index values as shown in below.

Grocery[0][0] = “mangoes”;

Console.log(fruits) for (let list of grocery){

Console.log(list).

}

//[“mangoes” , “orange” , “bananas”]

1. SPREAD OPERATOR(…):- allows an iterable such as array or string to be expanded in places where zero or more arguments are expected (unpacks the elements).

Example :-

Let numbers = [1,2,3,4,5,6,7,8,9]

Console.log(numbers) //{1,2,3,4,5,6,7,8,9}

Console.log(…numbers) //1,2,3,4,5,6,7,8,9

Let maximum = Math.max(number) //NaN

Let maximum = Math.max(…number)

Console.log(maximum) //9

Let class1 = [“Naveen” , “Ramesh” ,” Leela”]

Let class2 =[“Ramu” , “Venkatesh” ,”Manoj”]

Class1.push(class2)

//[“Naveen” , “Ramesh” ,” Leela” , “[“Ramu” , “Venkatesh” ,”Manoj”]”]

Class1.push(…class2)

//[“Naveen” , “Ramesh” ,” Leela” , “Ramu” , “Venkatesh” ,”Manoj” ]

Console.log(..class1) //Naveen Ramesh Leela Ramu Venkatesh Manoj

1. REST PARAMETERS(…) :- represents an indefinite number of parameters are packs arguments into an array.

Let a=1;

Let b=2;

Let c=3;

Console.log(sum(a , b , c ));

Function sum(..numbers){

let total = 0;

for (let number of numbers){

total += number;

}

return total

} //6

1. CALL BACKS :-a function passed as an argument to another function. Ensures that a function is not going to run before a task is completed. When one function has to wait for another function . Helps us avoid errors and potential problems. Helps us develop asynchronous code.

Example :-

sum(2,3,displayConsole);

function sum(x , y , callback){

let result=x + y;

callback(result);

}

Function displayConsole(output){

console.log(output);

} //5

1. ARRAY MEHODS :-

|  |  |  |
| --- | --- | --- |
| Map() | Creates a new array by applying a provided functions to each element of the original array. | Const numbers = [1,2,3,4,5]  Const sqrtnum=numbers.map((num) => num\*num);  Console.log(sqrtnum) //[1,4,9,16,25] |
| Filter() | Creates a new array with all elements that pass provided test case. | Const numbers = [1,2,3,4,5]  Const evennum=numbers.filter((num) => num%2==0 );  Console.log(evennum) //[2,4] |
| Reduce(total,  element) | Applies a reducer function to each element of the array resulting in a single output value. | Const numbers=[1,2,3,4,5]  Const sum =numbers.reduce((accumulator,num) => accumulator + num , 0);  Console.log(sum) //15 |
| forEach(index,  element, array) | Executes a provided function once for each array of element. | Const numbers=[1,2,3,4,5]  numbers.forEach((num) => console.log(num));  //1 2 3 4 5 |
| Find() | Returns the first element in the array that satisfies the provided testing function | Const numbers = [1,2,3,4,5]  Const firsteven=numbers.find((num) => num%2==0 );  Console.log(firsteven) //2 |
| Every() | Tests whether all elements in the array pass the provided testing functions. | Const numbers = [1,2,3,4,5]  Const alleven=numbers.Every((num) => num%2==0 );  Console.log(alleven) //2 |
| Some() | Tests whether at least one element in the array satisfies the provided testing function. | Const numbers = [1,2,3,4,5]  Const haseven=numbers.some((num) => num%2==0 );  Console.log(haseven) //true |
| Sort() | Sort elements based on the string only whenever we have to pass a values can be sort in different order. | Const numbers=[1,2,4,3,5]  numbers.sort((x,y)=>x-y)  console.log(numbers) //[1,2,3,4,5] |

1. ARRAY.FOREACH() :- Executes a provided callback function once for each array element. It will pass the element , index , array .

Example :-

Let students =[“naveen” , “ramesh” , “leela”]

students.forEach(capitalize);

students.forEach(print);

Function capitalize(element,index,array){

array[index] = element[0].toUpperCase() + element .substring(1);

}

Function print(element){

console.log(element);

}

1. ARRAY.MAP() :- executes a provided callbacks function once for each array element and creates a new array .

Example :-

Let numbers = [1,2,3,4,5];

Let squares =numbers.map(square);

squares.forEach(print);

Function square(element){

return math.pow(element,2);

}

Function print(element){

console.log(element)

}

1. ARRAY.FILTER() :- creates a new array with all elements that pass the test provided by a function.

Example :-

Let ages = [18,16,21,17,19,90];

Let adults = age.filter(checkage);

adults.forEach(print);

Function checkage(element){

return element >= 18 ;

}

Function print(element){

console.log(element);

}

1. ARRAY.REDUCE() :- reduces an array to a single value.

Example :-

Let prices =[5,10,15,20,25];

Let total =prices.reduce(checkout);

Function checkout(total, element){

return total+element;

}

Console.log(`The total is : ${total}`);

1. ARRAY.SORT() :-

Let grades =[100,50,90,60,80,70];

Let grades =grades.sort(ascendingsort);

grades.forEach(print);

Function descendingsort(x,y){

return y-x;

}

Function ascendingsort(x,y){

return x-y;

}

Function print(){

console.log(element);

}

1. ARROW FUNCTION {=>} :- compact alternative to a traditional function expression.

Example :- 1

Let greeting = function (username){

console.log(“hello ${username}”)

}

or

Let greeting =(username) =>console.log(“hello ${username}”);

Greeting(“bro”);

Example :-2

const percent =function(x , y){

return x/y \*100;

}

Console.log(‘${percent(75,100)} %’);

or

const percent = (x,y) => x/ y \*100;

console.log(‘${percent(75,100)} %’);

Example :- 3

Let grades = [100,50,90,60,80,70];

grades.sort(descendingsort);

grades.forEach(print);

Function descendingsort(x,y){

return y-x;

}

Function print(){

console.log(element);

}

Or

Let grades = [100,50,90,60,80,70];

grades.sort(function(x,y){

return y-x;

});

grades.forEach(function(element){

console.log(element);

});

Or

Let grades = [100,50,90,60,80,70]

grades.sort((x,y) => y-x ;

grades.forEach((element) =>console.log(element));

Example :- 4

Let cards =[“A”,”2”,”3”,”4”,”5”,”6”,”7”,”8”,”9”,”10”,”J”,”Q”,”K”]

Shuffle(cards);

Console.log(cards);

cards.forEach(card => console.log(card))

Function shuffle(array){

let currentIndex = array.length;

while (currentIndex != 0){

let randomIndex =Math.floor(Math.random() \* array.length)

currentIndex -=1 ;

let temp =array[currentIndex];

array[currentIndex] = array[randomIndex];

array[randomIndex] = temp;

}

return array;

}

1. NESTED FUNCTIONS :- function inside other functions. Outer function have access to inner functions .Inner Function are “hidden” from the outside the outer function. It provides data security.

Example :-

Let username = “Bro”

Let userInbox = 0

Login()

Function login(){

displayUsername();

displayUserInbox();

}

Function displayUsername(){

consloe.log(“welcome ${username}”);

}

Function displayUserInbox(){

cosloe.log(“welcome ${username}”);

}

Or

Let username = “Bro”

Let userInbox = 0

Login()

Function login(){

displayUsername();

displayUserInbox();

Function displayUsername(){

consloe.log(“welcome ${username}”);

}

Function displayUserInbox(){

cosloe.log(“welcome ${username}”);

}

}

1. MAP :-object that holds key-value pairs of any data type.

|  |  |
| --- | --- |
| Get () | It is used to fetch an element to the map. |
| Set() | Added element to the map. |
| Delete() | Delete an element on the map. |
| Has() | Check element in the map. |
| Size | It is used to display the size of the map |

example :-

const store = new Map ([

[“t-shirt “ , 20 ],

[“jeans” , 30 ] ,

[“socks” , 10 ],

[“underwear” , 50]

]);

Let shoppingcart = 0;

shopping += store.get(“t-shirt”); // 20

shopping += store.get(“underwear”); // 70

Console.log(store.set(“hat”,40))

Console.log(store.delete(“hat”)

Console.log(store.has(“hat”)) // False

Console.log(store.size) // 4

Store.forEach((value, key) => console.log(“${key} ${value}”));

1. OBJECT :-A group of properties and methods.

* Properties = what an object has methods
* Methods =what an object can do
* use dot access properties/methods.

Example :-

Const car = {

model : “mustang” ,

color : “red” ,

year : 2023 ,

drive : function(){

console.log(“you drive the car”),

}

break : function(){

console.log(“you step on the break”)

}

}

Console.log(car.model); mustang

Console.log(car.color); red

Console.log(car.year); 2023

car.drive() you drive the car.

car.break() you step on the breaks

1. THIS :- Reference to particular object and the object depends on the immediate context.

Example :-

Const car = {

model : “mustang” ,

color : “red” ,

year : 2023 ,

drive : function(){

console.log(“you drive the ${this.model}”),

}

}

car.drive() // you drive the mustang.

1. CLASS :- Class is a blueprint for creating objects define what properties and methods they have use a constructor for unique properties.

Example :-

Class player1{

score = 0 ;

pause() {

console.log(“you paused the game”);

}

exit(){

console.log(“you exited the game”);

}

}

Const player1 = new player();

Player1.score += 1;

Console.log(player1.score); 1

Player1.pause(); you paused the game

Player.exit(); you exited the game

1. CONSTRUCTOR :-a special method of a class accepts arguments and assigns properties.

Example :-

Class student{

constructor(name, age, gpa){

this.name = name ;

this.age = age ;

this.gpa = gpa;

}

study(){

console.log(“${this.name} is studying”);

}

}

Const student1 = new student(“Naveen” , 30 , 3.2);

Console.log(student1.name); #Naveen

Console.log(student1.age); #30

Console.log(student1.gpa); #3.2

Student1.study() #Naveen is studying.

1. STATIC :- Belongs to the class, not the objects properties :useful for caches , fixed-configuration , method :useful for utility functions.

Example :-

Class car{

numberOfCars =0

constructor (model){

this.model = model ;

this.numberOfCars += 1 ;

}

}

const car1 = new car(“Mustang”);

Console.log(car1.numberOfCars); #1

Example :-

Class car{

static numberOfCars =0

constructor (model){

this.model = model ;

car.numberOfCars += 1 ;

}

startRace(){

console.log(“3,2,1 ……..G0 !”);

}

static EndRace(){

console.log(“stop”);

}

}

const car1 = new car(“Mustang”);

const car2 = new car(“Ferrari”);

const car3 = new car(“corvette”);

const car4 = new car(“BMW”);

Console.log(car.numberOfCars); #4

Car1.startRace() #3,2,1 ……G0 !

Car.EndRace() #Stop

1. INHERITANCE :- A child class can inherit all the methods and properties from another class.

Example :-

Class Animal {

alive = true ;

eat() {

console.log(“This ${this.name} is eating”);

}

sleep(){

console.log(“This ${this.name} is sleeping”);

}

}

Class Rabbit extends Animal {

name = “Rabbit” ;

run() {

console.log(“This ${this.name} is running”);

}

}

Class Fish extends Animal {

name = “Fish” ;

swim(){

console.log(“This ${this.name} is swimming”);

}

}

Class Hawk extends Animal {

name = “Hawk” ;

flying(){

console.log(“This ${this.name} is flying”);

}

}

Const rabbit =new Rabbit();

Const fish =new Fish();

Const hawk =new Hawk();

Rabbit.eat(); #this rabbit is eating

Fish.swim(); #this fish is swimming

Hawk.sleep(); #this hawk is sleeping

1. SUPER :- super keyword refers to the parent commonly used to invoke constructor of a parent class.

Example :-

Class Animal {

constructor(name , age){

this.name = name ;

this.age = age ;

}

}

Class Rabbit Extends Animal {

constructor (name , age , runspeed){

super(name, age);

this.runspeed =runspeed ;

}

}

Class Fish Extends Animal {

constructor (name , age , swimspeed){

super(name,age);

this.swimspeed =swimspeed ;

}

}

Class Hawk Extends Animal {

constructor (name , age , flyspeed){

super(name,age);

this.flyspeed =flyspeed ;

}

}

Const rabbit = new Rabbit (“rabbit” , 1 , 40);

Const fish = new Fish(“fish” , 2 , 80);

Const hawk =new Hawk (“hawk” , 3 ,200);

Console.log(rabbit.name); #rabbit

Console.log(rabbit.age); #1

Console.log(rabbit.runspeed); #40

Console.log(fish.name); #fish

Console.log(fish.age); #2

Console.log(fish.swimspeed); #80

Console.log(hawk.name); #hawk

Console.log(hawk.age); #3

Console.log(hawk.flyspeed); #200

1. GET AND SET :-

* Get = blinds an object property to a function when that property is accessed.
* Set = blinds an object property to a function when property is assessed value.

**Example :-**

**Class car {**

**constructor (power){**

**this.\_gas = 25 ;**

**this.\_power = power ;**

**}**

**get power(){**

**return ‘${this.\_power} hp’ ;**

**}**

**set gas(value){**

**if (value >50){**

**value = 50;**

**}else if (value < 0){**

**value = 0 ;**

**}**

**}**

**}**

1. **object as arguments to functions :-**

**Example :-**

**Class car {**

**constructor (model , year , color){**

**this.model =model ;**

**this.year = year ;**

**this.color =color ;**

**}**

**}**

**Const car1 = new car(“Mustang” , 2023 , “red”);**

**Const car2 = new car(“Corvette” , 2024 , “blue”);**

**Const car3 = new car(“Lambo” , 2022 , “yellow”);**

**displayInfo(car2);**

**Function displayInfo(car){**

**console.log(car.model);**

**console.log(car.year);**

**console.log(car.color);**

**}**

**Function changeColor (car , color){**

**car.color = color ;**

**}**

1. **create an array as object :-**

**Example :-**

**Class car {**

**constructor (model , year , color){**

**this.model =model ;**

**this.year = year ;**

**this.color =color ;**

**}**

**drive() {**

**conole.log(‘you drive the ${this.model}’);**

**}**

**}**

**Const car1 = new car(“Mustang” , 2023 , “red”);**

**Const car2 = new car(“Corvette” , 2024 , “blue”);**

**Const car3 = new car(“Lambo” , 2022 , “yellow”);**

**Const cars = [car1 , car2 , car3];**

**Console.log(cars[0].model) #Mustang**

**Console.log(cars[1].model) #corvette**

**Console.log(cars[2].model) #lambo**

**Cars[0].drive(); #you drive the mustang**

**Cars[1].drive(); #you drive the corvette**

**Cars[2].drive(); #you drive the Lambo**

**startRace(cars); # you drive the mustang**

**you drive the corvette**

**you drive the Lambo**

**Function startRace (cars){**

**for (const car of cars){**

**car.drive();**

**}**

**}**

1. **ANONYMOUS OBJECTS :-**

* **objects without a name not directly referenced loss syntax. No need for unique names.**

**Example :-**

**Class card {**

**constructor(value,suit){**

**this.value = value ;**

**this.suit = suit ;**

**}**

**}**

**Let card1 =new card(“A”,”Hearts”);**

**Let card2 =new card(“A”,”Spades”);**

**Let card3 =new card(“A”,”Diamonds”);**

**Let card4 =new card(“A”,”clubs”);**

**Let card5 =new card(“2”,”Hearts”);**

**Let card6 =new card(“2”,”Spades”);**

**Let card7 =new card(“2”,”Diamonds”);**

**Let card8 =new card(“2”,”clubs”);**

**Let cards = [card1 , card2 , card3 , card4 , card5 , card6 , card7 , card8 ]**

**Console.log(card1.value + card1.suit) 🡺Ahearts**

**Console.log(cards[0].value + cards[0].suit) 🡺Ahearts**

**Let card = [new card(“A”,”Hearts”),**

**new card(“A”,”Spades”),**

**new card(“A”,”Diamonds”),**

**new card(“A”,”clubs”)**

**new card(“2”,”Spades”),**

**new card(“2”,”Diamonds”),**

**new card(“2”,”clubs),**

**new card(“2”,”Hearts”)]**

1. **ERRORS :-**

* **objects with a description of something went wrong.**
* **Can’t open a file**
* **lose connection**
* **user types of incorrect input (Type Error)**
* **throw = executes a user-defined error.**

1. **SETTIMEOUT () :-**

* **Invokes a function after a number of milliseconds.**
* **asynchronous function (doesn’t pause execution)**

**Example :-**

**Function FirstMessage(){**

**alter(“Buy this course for $50 ”)**

**}**

**Function SecondMessage(){**

**alter(“This is not a scam”)**

**}**

**Function ThirdMessage(){**

**alter(“Do IT”)**

**}**

**SetTimeout(FirstMessage, 3000);**

**SetTimeout(SecondMessage, 6000);**

**SetTimeout(ThirdMessage, 9000);**

1. **SETINTERVAL () :-**

* **Invokes a function repeatedly after a number of milliseconds.**
* **Asynchronous function(doesn’t pause execution)**

**Example :-**

**Let count = 0;**

**Let max = window.prompt(“count up of what # ? ”);**

**Max=Number(max);**

**Const Mytimer = setInterval(countup,1000);**

**Function countUp(){**

**count++1;**

**console.log(count);**

**if(count >= max){**

**clearInterval(Mytimer);**

**}**

**}**

1. **CONSOLE.TIME() AND CONSOLE.TIMEEND() :-**

* **starts a timer you can use to track how long an operation takes give each timer a unique name.**

**Example :-**

**//start**

**Console.time(“time response”);**

**//end**

**Console.timeEnd(“Response time”);**

1. DATE AND TIME :-

|  |  |  |
| --- | --- | --- |
| **Let date = new Date()** | **It will display the date with time.** | **1/3/2023, 2:03:04 AM** |
| **Let date =new Date(0)** | **It will display the starting date of the calendar** | **1/1/1970 , 2:45:36 PM** |
| **Let date =new Date(2023,0,1,2,3,4,5)** | **It will display the date with numbers format.** | **1/1/2023,2:03:4 AM** |
| **Let date = new Date(“January 1,2023 00:00:00”)** | **It will display the date with string format.** | **1/1/2023,00:00:00 AM** |
| **Let dayOfMonth = date.getDate()** | **It will display the day of the month (1-31).** | **1** |
| **Let dayOfweek =date.getDay()** | **It will display the current day (0-6).** | **2** |
| **Let month = date.getMonth()** | **It will display the month(0-11).** | **6** |
| **Let hour = date.getHours()** | **It will display the hours (0-23).** | **4** |
| **Let minutes = date.getMinutes()** | **It will display the minutes (0-59).** | **30** |
| **Let seconds = date.getSeconds()** | **It will display the Seconds (0-59).** | **35** |
| **Let ms = date.getMilliseconds()** | **It will display the milliseconds (0-999).** | **45** |
| **Date.SetFullyear(2024)** | **It will set the year (YYYY).** | **1/2/2024, 10:08:22AM** |
| **Date.setMonth(11)** | **It will set the month (0-11).** | **1/12/2024, 10:06:22 AM** |
| **Let year = date.getFullyear()** | **It will display the year (YYYY).** | **2023** |
| **Let time = date.getTime()** | **Get the milliseconds since January 1 , 1970** | **10:06:22 AM** |
| **Date.setDate(25)** | **Set the day as a number(1-31)** | **25/3/2023** |
| **Date.setHours(3)** | **Set the hours(0-23)** | **1/2/2023 , 3:08:23 AM** |
| **Date.setMilliseconds(25)** | **Set the milliseconds(0-999)** | **1/2/2023 , 3:08:25 AM** |
| **Date.setMinutes(4)** | **Set the minutes (0-59)** | **1/2/2023 , 3:04:25 AM** |
| **Date.setSeconds(57)** | **Set the seconds (0-59)** | **1/2/2023 , 3:08:57 AM** |
| **Date.setTime()** | **Set the time** | **1/2/2023 , 3:08:25 AM** |

1. **PROMISES :-**

* **object that encapsulates the results of an asynchronous operation.**
* **let asynchronous methods return values like synchronous methods.**
* **”I promise to return something in the future”.**
* **The state is ‘pending’ then : ‘fulfilled’ or ‘rejected’**
* **The results is what can be returned**
* **2 parts producing & consuming.**

1. **SYNCHRONUS AND ASYNCHRONUS :-**

|  |  |
| --- | --- |
| **SYNCHRONOUS CODE** | **ASYNCHONOUS CODE** |
| **It an ordered sequence.** | **Out of sequence.** |
| **Start now , finish now** | **Start now , finish sometime later.** |
|  | **Ex :-**   * **Access a database** * **Fetch a file** * **Tasks that take time** |

**ES6 MODULE :-**

* **The idea behind a module is that it’s a file of reusable code.**
* **we can import sections of pre-written code to use whenever.**
* **Great for any general utility values and functions**
* **Helps to make your code more reusable and maintainable.**
* **Think of modules as separate chapters of a book.**

**Example :-**

**Import {pi , getcircumference,getArea} from “./math\_utilise.js”;**

**Console.log(pi) 3.1415**

**Let circumference =getcircumference(10);**

**Console.log(circumference) 62.80**

**Let area =getarea(10);**

**Console.log(area); 314.159**

**Import as Mathutil from “./math\_utilise.js”;**

**Console.log(Mathutil.pi) 3.1415**

**Let circumference = Mathutil.getcircumference(10);**

**Console.log(circumference) 62.80**

**Let area =Mathutil.getarea(10);**

**Console.log(area); 314.159**

**Export Const pi = 3.1415;**

**Export Function getcircumference(radius){**

**return 2 \* pi \* radius;**

**}**

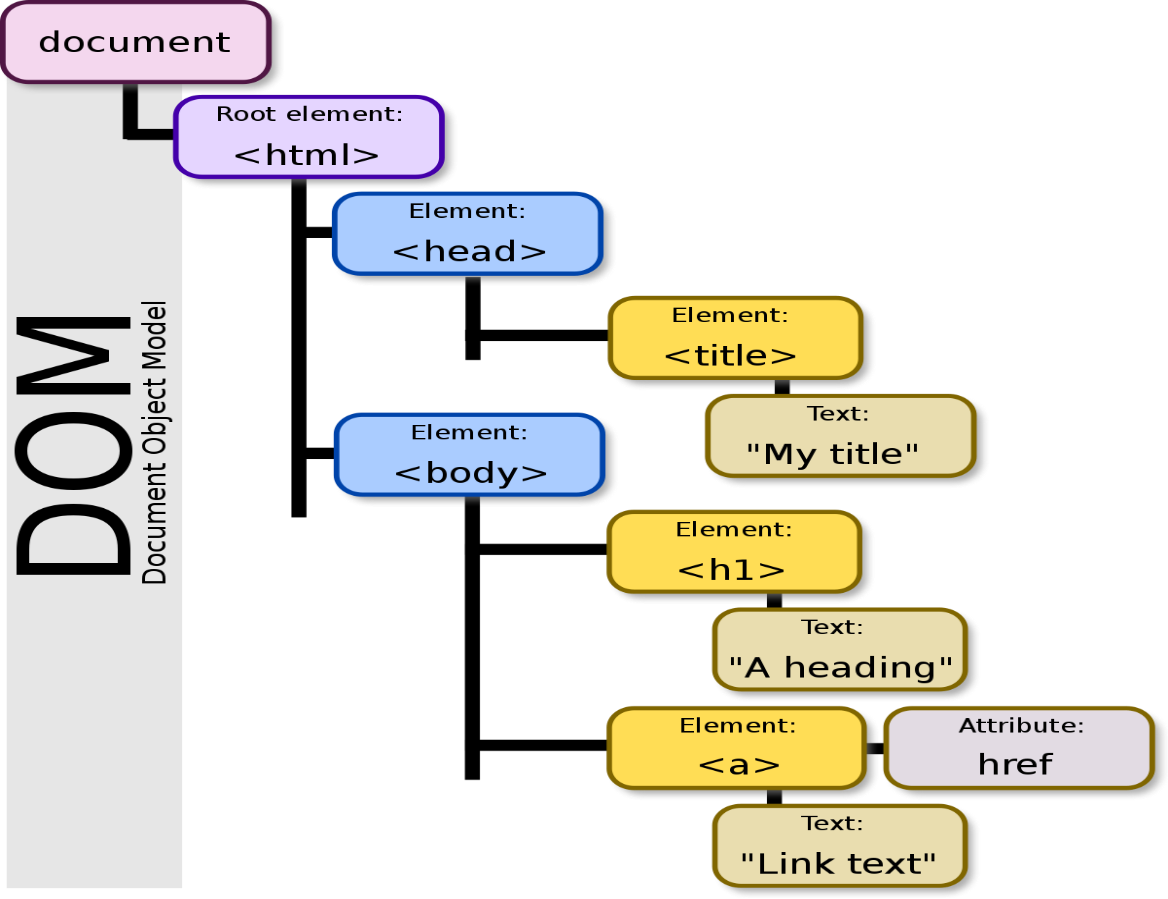
**Export Function getradius(radius){**

**return pi \* radius \* radius;**

**}**

1. **DOM MODE :-**

* **DOM means an interface for changing the content of page.**

****

1. ELEMENR SELECTORS :-

|  |  |
| --- | --- |
| **GetElementById ( )** | **It will select tag based on the id.** |
| **GetElementByName ( )** | **It will select the based on the name of the tag.** |
| **GetElementsByTagName ( )** | **Provides a collection of all child elements with the specified tag name.** |
| **GetElementsByClassName ( )** | **It will select tag based on the class name.** |
| **querySelector ( )** | **In querySelector we can select the tag based on the selector symbol.** |
| **getAttribute ( )** | **Returns the specified attribute value of an element node.** |
| **querySelectorAll ()** | **It will select all the tag name.** |
| **getAttributeNS ()** | **Returns string value of the attribute with the specified namespace and name.** |
| **getAttributeNode()** | **Get the specified attribute node.** |
| **getAttributeNodeNS ()** | **Returns the attribute node for the attribute with the specified tag name.** |
| **getElementsByTagNameNS()** | **Returns a live HTMLCollection of Elements with a certain tag name belonging to the given namespace.** |
| **hasAttribute ()** | **Returns true if an element has any attributes ,otherwise False.** |
| **removeAttribute ()** | **Removes the specified attribute from an element.** |
| **RemoveAttributeNS ()** | **Removes the specified attribute from an element within a certain namespace.** |
| **hasAtrributeNS ()** | **Provides a true /false value indicating whether the current element in a given namespace has the attribute.** |
| **RemoveAttributeNode ()** | **Take away a specified attribute node and returns the removed node.** |
| **SetAttribute ()** | **Sets or change the specified attribute to specified value.** |
| **SetAttributeNS ()** | **Adds a new attribute or changes the value of an attribute with the given namespace and name.** |
| **setAttributeNode ()** | **Sets or change the specified attribute node.** |
| **setAttributeNodeNs ()** | **Adds a new namespaced attribute node to an element** |

1. DOM TRANVERSAL :-

|  |  |
| --- | --- |
| **.firstElementChild** | **1 2 3 4 5 6** |
| **.lastElementChild** | **1 2 3 4 5 6** |
| **.parentElement** | **1 2 3 4 5 6** |
| **.nextElementSibling** | **1 2 3 4 5 6** |
| **.previousElementSibling** | **1 2 3 4 5 6** |
| **.Children[]** | **1 2 3 4 5 6**  **Based on children index number it will add the colors.** |
| **Array.from(.children)** | **1 2 3 4 5 6** |