Page No.
Date : 14 11 2023

Java Script

* Java Script :-

Java Script (JS) is a lightweight, interpreted or just-in-time compiled programming language with first-class-functions. While it is most well known as the scripting language for web page, many non-browser environments also use it, such as Node.JS, Apache couch DB and Adobe acrobat. Java script is a prototype-based Multi-paradigm, single threaded, dynamic language, supporting object-oriented, imperative and declarative (e.g functional programming) style.

Page No.
Date :

* Console.log(" "); in Javascript :-

The console.log() is a function in Javascript that is used to print any kind of variable defined before in it or to just print any massage that needs to be displayed to the user.

Example :-

-o console.log(" Hello Vaibhav ");

-o var a = 2 ;
console.log(a);

-o var str = " Hello Vaibhav ";
console.log(str);

-o var ch = 'a' ;
console.log(ch);

-o var a = 10;
console.log(" A value is " + a);

-o var str = " Vaibhav ";
console.log(" Hello " + str);

* Javascript is comment :-

single line comment : //----

Multiple line comment : /* --
-- */

* Variable:

=> Variables are containers for storing data. (storing data val

=> variables in three type : declare:

- Var
- let
- const.

* Var :-

Var is the keyword that tells javascript you're declaring variable. x is the name of the variable. = is the operator that tells javascript a value is coming up next. 100 is the value for the variable to store.

Syntax.

```
Var variable_name ;
```

Example :-

```
Var a ;
Var a = 10 ;          // a = 10
```

```
Var a = 10 ;
Var a = 20 ;
console.log (a) ;     // a = 20 ;
```

```
Var a = 10 ;
console.log (a) ;     // a = 10
Var a = 20 ;
console.log (a) ;     // a = 20
```

* let :-

- The let keyword was introduced in ES6 (2015)
- Variable defined with let cannot be redeclared
- Variable defined with let must be declared before use.
- Variable defined with let have block scope.

=> Cannot be redeclared :-

Variable defined let cannot be redeclared. You cannot accidentally redeclare a variable.
with let you do not this:

```
let x = 10 ;
let x = 20 ;      // syntax error :
                  x is already been
                  declared.
```

=> Block Scope :-

Before ES6 (2015) JavaScript had only Global scope and Function scope.
ES6 introduced two important new JavaScript keyword : let and const.
These two keywords provide block scope in JavaScript.
Variable declared inside a {} block cannot be accessed from outside the block.

Example :-

let :-

```
let  x = 10;
```

Block scope :

```
let  x = 10;          // x = 10
{
    let x = 2;        // x = 2
}
// x = 10
```

* Const :-

- The Const Keyword was introduced in ES6 (2015).
- Variable defined with const cannot be Redeclared.
- Variable defined with const cannot be reassigned.
- Variable defined with const have block scope.

Syntax :

```
const variable-name = value;
```

Example :

```
Const, Earth-gravity = 9.807 ;
```

cannot be reassigned

```
const earth_gravity = 9.807 ;
earth_gravity = 9.707 ; // error
earth_gravity = earth_gravity + 1.5 ; //error
```

* Block scope :-

Declaring a variable with const is similar to let when it comes to Block scope.

```
const x = 10;        // x = 10
{
  const x = 2;       // x = 2
}
// x = 10;
```

* ① Difference var and let.

Var	let.
-o Var keyword introduced in "ES1".	-o let keyword introduced in "ES6".
-o var keyword allows the duplicate variables.	-o let keyword won't allow the duplicate variables.
-o variable hoisting is done. raised with var keywords.	-o we can overcome variable hoisting with let keyword.
-o scope rule broken by var keyword.	-o let keyword obeys the scop rule.

-o Global pollution issue. arised because of var keyword.	-o we can overcome global pollution issue by using let keyword.

* Operators :-

- Arithmetic operator
- Assignment operator
- comparison operator
- logical operator.
- conditional operator.
- Type operator.

=> Arithmetic operators :-

arithmetic operator are used to perform arithmetic on numbers :

+ : addition
- : subtraction
* : Multiplication
** : Exponention (ES2016). (Math.pow(x,y)):
/ : Division
% : Modulus (Division Remainder)
++ : Increment.
-- : Decrement.

=> Assignment operators

Assignment operator assign values to javascript variable.

operator	Example	same As
=	x = y	
+=	x += y	x = x + y.
-=	x -= y	x = x - y.
*=	x *= y	x = x * y.
/=	x /= y	x = x / y.
%=	x %= y	x = x % y.
**=	x **= y	x = x ** y.

=> Comparison Operators.

==	:	equal to.
===	:	equal value and equal type.
!=	:	not equal.
!==	:	not equal value or equal type.
>	:	greater than
<	:	less than
>=	:	greater than or equal to.
<=	:	less than or equal to.
?	:	ternary operator.

=> Logical operators.

&&	:	logical and
||	:	logical or
!	:	logical not.
^	:	logical xor.

=> Type operators

type of : Returns the type of a variable.
instance of : Returns true if an object is an instance of an object type.

Exercise:
Exercise 8-

4 Arithmetic operators:

```
var a = 10;
var b = 20;
console.log(a+b);    // 30
console.log(b-a);    // 10
console.log(a*b);    // 200
console.log(b/a);    // 2
console.log(b%a);    // 0
console.log(5**2);   // 25
console.log(a++);    // 10 → 11
console.log(++a);    // 12 → 12
console.log(a--);    // 12 → 11
console.log(--a);    // 10 → 10.
```

4 Assignment operators.

```
var a = 10;
var b = 20;
let x = a;
console.log(a+=b);   // 30
console.log(a-=b);   // -10
console.log(a*=b);   // 200
console.log(b/=a);   // 2
console.log(b%=a);   // 0
console.log(b**=a);  // 25
```

=> Comparison operators.

```
console.log (10 == 10);      // true
console.log (10 === 10);     // true
console.log ("0" == false);  // true
console.log (false == "0");  // true
console.log (false === "0"); // false.
console.log (10 != 20);      // true.
console.log (0 !== false);   // true.
console.log (10 > 20);       // false.
console.log (10 < 20);       // true
console.log (10 >= 20);      // false
console.log (10 <= 20);      // true.
```

=> logical operators.

```
let a = 2, b = 3, c;
c = --a && b++;  // if first condition value
console.log(a,b,c);   is 1 after it is continue
                      // 1, 4, 3
c = --a || b++;       // 1, 3, 1

console.log (a != b);  // true

console.log (1^1);   // 0
console.log (1^0);   // 1
console.log (0^1);   // 1
console.log (0^0);   // 0

console.log (1^0^1);  // 0
console.log (0^1^0);  // 1
console.log (0^0^0);  // 0
console.log (1^1^1);  // 1
```

String :-

Javascript string is zero or more characters written inside quotes.

Example:

```
let str = "Hello vaishakh";
console.log(str); // Hello vaishakh

let a = 20;
let b = 20;

let str1 = "Hello" +a + "vaishakh"
           +b + "str-1";
// hello 20 vaishakh 20 str-1.
let str2 = 'hello' +b + 'vaishakh
           +a + 'str-2';
// Hello 20 vaishakh 20 str-2
let str3 = `Hello ${a} vaishakh
           ${b} ${"str-3"}`;
// Hello 20 vaishakh 20 str-3.
```

* Null :

The null value represents the intent absence of any object value. it is one of Javascript's primitive values and is treated as falsy for boolean operations.

* Undefined :

In Javascript, a variable without a value, has the value undefined. The type is also undefined.

```
let a ;          // value is undefined,
                    type is undefined.
```

* Empty values :

An empty value has nothing to do with undefined.
An empty string has both a legal value and a type.

```
let str = "" ;   // The value is "",
                    the typeof is string
```

* NaN : Not a number.
NaN type of number.
NaN is a number that is not a legal number.

Exercise :-

```
console.log ( undefined == undefined ) ; // true
console.log ( undefined === undefined ) ; // true.
console.log ( null == null ) ;  // true
console.log ( null === null ) ; // true
console.log ( NaN == NaN ) ; // False everytime
    NaN give different value
console.log ( NaN === NaN ) ; // False everytime
    NaN give different value.

console.log ( null == undefined ) ; // true
console.log ( null === undefined ) ; // false.
console.log ( null == 0 ) ;  // false.
console.log ( null === NaN ) ;  // false.

console.log ( typeof 0 ) ;  //
console.log ( type of null ) ;  // object.
console.log ( typeof undefined ) ; //undefined.
console.log ( typeof NaN ) ;  // Number

console.log ( 10 * "s" ) ;  //NAN
console.log ( ) ;  // Nan.

console.log (null == 0 )
console.log (undefined == 0 )
console.log ( NaN == 0 )

console.log ( undefined == null ) ;
console.log ( undefined === null ) ;
console.log ( undefined == NaN ) ;
console.log ( NaN == null ) ;
console.log ( NaN === null ) ;
```

Date :

* if ... else :-

The if...else statement executes a statement if a specified condition is truthy. If the condition is falsy, another statement in the optional else clause will be

Syntax :-

```
if (condition)
    statement 1
```

```
if condition ).
    statement 1
else
    statement 2
```

```
if (condition 1)
    statement 1
else if ( condition 2)
    statement 2
else if (condition 3)
    statement 3.
else
    statement N
```

```
if (condition 1)
    statement 1
else
    if (condition 2)
        statement 2
    else
        statement 3.
```

Date :

* Loops :-

i) for loop

A for loop repeats until a specified condition evaluates to false. The JavaScript for loop is similar to the Java and C for loop.

Syntax :-

for (initialExpression ; conditionExpression ; increment/decrement)
{ statement
y ;

ii) while loop.

A while loop executes its statements as long as a specified condition evaluates to true.

syntax :-

while (condition).
statement

iii) do.... while loop

The do.... while loop repeats until a specified condition evaluates to false.

Syntax :-

do
statement
while (condition);

* Array :-

The Array objects, as with arrays in other programming languages, enables storing a collection of multiple items under a single variable name, and has number for performing common array operations.

=> In JavaScript, array aren't primitives but are instead Array objects with the following core characteristics:

- JavaScript arrays are resizable and can contain a mix of different data types.
- JavaScript array are not associative arrays and so, array elements cannot be accessed using arbitrary strings as indexes, but must be accesed using nonnegative integers as index.
- JavaScript arrays are zero-indexed. The first element of an array is at index 0, the second is at index 1, and so on - and the last element is at the value of the array's length property minus 1.
- JavaScript array-copy operations create shallow copies.

Date :

Syntax :-

var / let / const array_name = [item1, item2, ...];

* Relationship between length and numerical properties

A Javascript array's length property and numerical properties are connected.

Several of the built-in array's methods (e.g., join(), slice(), indexOf(), etc) take into account the value of an array's length property when they're called.

Other method (eg. push(), splice(), etc) also result in updates to an array's length property.

=> Suppose array

let arr = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19];

→ console.log(arr.length); // 10.
// print array length

→ arr.length = 100; // change array length
console.log(arr.length); // 100

- `console.aur (type of aur);` // object
  // show array type

- `console.log (aror);` // [10, 11, 12, 13, 14, 15, 16, 17, 18,
  // print a array. <90 empty itm

- `console.log (aror[5]);` // 15.
  // print enter index number store value.

- `console.log (aur["5"]);` // 15.
  // print specific number (index) store value.

- `aror[50] = 550;` // array index 50 store
  `aur[49] = undefind;` // array index 49 stor
  `console.log (auir);`
  // [10, 11, 12, 13, 14, 15, 16, 17, 18,
  19, <39 empty items>, undefined,
  550, <49 empty items>.

- `aror[3.1] = 60;` // key value.
  // one kind of object.
  `console.log (auir);`
  // [10, 11, 12, 13, 14, 15, 16, 17, 18, 19
  <39 empty items>, undefined, 550, <49 empty ite
  , '3.1' : 60

=> Now all these exercises for this array.

`let aror = ["ten", 11, "twelve", 13, 14, "15", 16, "17"`

10 Splice () :-

The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place. splice method is update / modify to a array.

Syntax :-

10 splice (start);

- negative index counts back from the end of array
- if start < 0, start + array.length is used.
- if start <= array.length or start is omitted 0 is used.
- if start >= array.length, no element will be deleted, but the method will behave as an adding function, adding as many elements as provided.

```
let arr = [10, ---, 19]; // length = 10
```

10 arr.splice(5); // start index 0 to 5 after dele
console.log(arr);
// [10, 11, 12, 13, 24];

10 arr.splice(-3); // end 3 element delete.
console.log(arr); // arr.length - 3 ; = 7
// [10, 11, 12, 13, 14, 15, 16].

→ splice (start , deleteCount) ;

↳ An integer indicating the number of elements in the array to remove from start.

↳ If deleteCount is omitted, or if its value is greater than or equal to the number of elements after the position specified by start, then all the elements from start to the end of the array will be deleted. However, if you wish to pass Infinity as deleteCount to delete all elements after start, because an explicit undefined gets converted to 0.

↳ If deleteCount is 0 or negative no elements are removed. In this case, you should specify at least one new element.

let arr = [10, --, 19] ; // length = 10

→ arr.splice (5, 1) ; // index no 5 to start and one element delete.

console.log (arr) ; // [10, 11, 12, 13, 14, 16, 17, 18, 19]

→ arr.splice (-4, 2) ; // end to start 4 and 2 element delete.

console.log (arr) ;

// [10, -- 14, 15, 16, 19]

→ arr.splice (5, -1) ; // Not a delete.
arr.splice (-4, -2) ; array as it is.

* splice (start, deletcount, item1, item2, itemN)

- The elements to add to the array, beginning from start.
- If you do not specify any elements, splice() will only remove elements from the array.

let arr = [10, ---, 19] // array length = 10

- arr.splice(5, 2, 30, 31); // a[5] after 2 element delete and add 30, 31 elements

console.log(arr); // [10, --, 14, 30, 31, 17, 18, 19]

- arr.splice(5, 2, 30, 31, 32, 33);

console.log(arr); // [10, --, 14, 30, 31, 32, 33, 17, 18, 19].

- arr.splice(0, arr.length, 30, 31, 32, 33);

// [30, 31, 32, 33]

* slice() :-

The slice() method returns a shallow copy of portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be de modified.

Syntax :-

*=0 slice () // array print.

*=0 slice (start)

↳ zero-based index at which to start extraction, converted to an integer.

↳ Negative index counts back from the end of the array - if start < 0, start + array.length is used.

↳ if start < -array.length or start is omitted, 0 is used.

↳ if start >= array.length, nothing is extracted.

let arr = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19].

-o console.log (arr.slice (3));
// [13, 14, 15, 16, 17, 18, 19].

-o console.log (arr.slice (-3));
// [17, 18, 19]

*=0 slice (start, end):

↳ zero based index at which to end extraction, converted to an integer. slice() extracts up to but not including end.

- Negative index. count back from the end of the array - if end < 0, end + array.length is used.
- if end < -array.length, 0 is used.
- if end >= array.length or end is omitted, array.length is used, causing all elements until the end to be extracted.
- if end is positioned before or at start after normalization, nothing is extracted.

*→ Return value:

A new array containing the extracted elements.

```
console.log(arr.slice(3,12));
// [13, 14, 15, 16, 17, 18, 19];

console.log(arr.slice(-3,2)); // [].

console.log(arr.slice(-5,-6)); // [].

console.log(arr.slice(-5,-3));
// [15, 16].

let new-arr = arr.slice(); // create dup
console.log(new-arr); array
```

* at() :- Used to enter a index number and store element.

syntax :-

console.log (array-name.at(index));

-o console.log(arr.at(4)); // 14.

* join() :- Join all elements of an array into string.

syntax :-

-o console.log(arr_new.join());
// 10,11,12,13,14,15,16,17,18,19

-o console.log(arr_new.join(""));
// 10111213141516171819

-o console.log(arr_new.join("-"));
// 10-11-12-13-14-15-16-17-18-19-

-o console.log(typeof arr.join()); // string.

* function ();

Functions are one of the fundamental building block in javascript a function in javascript is similar to a procedure -- a set of statements that performs a task or calculates a values, but for a procedure to qualify as function, it should take some input and return an output where there is some obvious relationship between the input and the output. To use a function, you must define it some where in the scope from which you wish to call it.

Syntax :-

```
function function_name (variable)
{
    statement
}                    // defining function.

function_name (parameter) ; // calling function
```