JAVASCRIPT TUTORIAL

BASIC PROGRAM IN JAVASCRIPT

```
<script>
        alert("Tutor Joes");
    </script>
```

This will display an alert box with the message "Tutor Joes" and an OK button that the user must click to close the alert box.You can also use the prompt() function to display an input box and get input from the user.

DIFFERENCES BETWEEN VAR, LET, AND CONST IN JAVASCRIPT

Differences	var

Scope
• **Var acts as global scope**
• **Also access outside the block**
	`if(true)`
	`{`
	
	`  var msg="Welcome to Tutor Joes";`
	`}`
	`console.log(msg);`

• **Variable redeclaration**
• **Var allow variable redeclaration**
• **That is 25 change to 45**
	`var a=25;`
	`console.log(a)`
	
	`var a=45;`
	`console.log(a)`

Value assignment it allows	`var a=25;`
	`console.log(a);`
	`a=45;`
	`console.log(a);`

Difference	let
Scope • Let act as local scope • Only access in local scope	`if(true)` `{` `  let msg="Welcome to Tutor Joes";` `console.log(msg);→ Work` `}` `console.log(msg);→ Error`
Variable redeclaration • Let does not allow variable redeclaration • Show error	`let a=25;` `console.log(a)` `let a=45;→ error`
Value assignment • It allows	`let a=25` `console.log(a);` `a=45;` `console.log(a);`

Difference	const
Scope • Const also acts as local scope • Only access in local scope	`if(true)` `{` `  const msg="Welcome to Tutor Joes";` `}` `console.log(msg);→ error`
Variable redeclaration • const does not allow variable redeclaration • Show error	`const a=25;` `console.log(a)` `const a=45;→ error`
Value assignment • It does not allow • Show error	`const a=25;` `console.log(a);` `a=45;  //Constant Error` `console.log(a);`

Normal variable declare in const, it does not change but only change in object

```
const student={'name':"ram","age":12};
console.table(student);
console.log(student.name);
student.name="Joes";
console.table(student);
```

DATA TYPES IN JAVASCRIPT

Primitive data types: These are the basic data types that include numbers, strings, booleans, and special values like null and undefined.

Data Type	Description
String	A string is a collection of alphanumeric characters.
Number	Numbers are for numbers. We can't put a letter on here.
Boolean	Booleans have two values. True and false.
Null and Undefined	null and undefined stand for empty. That means they have no value assigned to them.
Symbols	Symbol is a primitive data type of JavaScript.It's a very peculiar data type. Once you create a symbol, its value is kept private and for internal use.
Array	An array is a type of object used for storing multiple values in single variable.
Object Literals	It is a comma-separated list of name-value pairs wrapped in curly braces.
Date	JavaScript does not have a date data type. However, you can use the Date object and its methods to work with dates and times in your applications.

Sample Program

```
//Data Types in JavaScript
/*
JS Dynamic Programming
```

```
String
Number   eg:  1.25,25
Boolean  eg:  True,False
Null
Undefinded
Symbols  E6


Array
Object Literals
Date
*/

var a=25.5;
varfname="Tutor Joes";
varisMarried=true;
var phone=null;
let b;
console.log(typeof b);


//ES6 2015

const s1=Symbol() //dlkfngsgs6565df6
console.log(s1)

const s2=Symbol() //fdfgdfg4345345
console.log(s2)

console.log(s1==s2);

//  Refernces type
var courses=['C','C++','Java'];
var student={
  'name':'Joes',
  'age':22
}
var d=newDate();
console.log(d);
console.log(typeof d);
```

TYPE CONVERSION

Few Examples of Type Conversion

- Strings to Numbers
- Numbers to Strings
- Dates to Numbers
- Numbers to Dates
- Boolean to Numbers
- Numbers to Boolean

Type conversion Methods

- String(value) : Converts the given value to a string.
- Number(value) : Converts the given value to a number.
- Boolean(value) : Converts the given value to a boolean.
- parseInt(value) : Converts the given value to an integer.
- parseFloat(value) : Converts the given value to a floating-point number.

JavaScript also has some unary operators that perform type conversion

- +value : Converts the given value to a number.
- -value : Converts the given value to a number.
- !value : Converts the given value to a boolean.

It is also possible to convert a value to a different type using the valueOf() and toString() methods.

JavaScript also has some automatic type coercion which happens when different types are being used together in an operation. For instance, if a string is added to a number, JavaScript will convert the string to a number before performing the addition.

It's important to keep in mind that type conversion can lead to unexpected results if not handled properly.

TYPE COERCION

Type Coercion refers to the process of automatic or implicit conversion of values from one data type to another. This includes conversion from Number to String, String to Number, Boolean to Number etc. when different types of operators are applied to the values.

Type coercion in JavaScript refers to the process of converting a value from one data type to another automatically. This happens when different data types are used in the same operation or when a value is compared to a value of a different data type.

For example, when a string is added to a number, JavaScript will automatically convert the string to a number before performing the addition. Similarly, when a non-boolean value is used in a boolean context, JavaScript will convert the value to a boolean using a set of rules.

JavaScript uses a set of rules to determine the type of a value when performing type coercion, these rules are called type coercion rules. For example, in JavaScript empty string, 0, null, undefined, NaN are considered as falsy values, and all other values are considered as truthy.

Type coercion can also occur when comparing values of different data types. For example, when comparing a string to a number, JavaScript will convert the string to a number before making the comparison.

It's important to be aware of type coercion when writing JavaScript code, as it can lead to unexpected behavior if not handled properly. To avoid type coercion issues, it's best practice to explicitly convert the data types when necessary.

ARITHMETIC OPERATORS

Sno	Operator	Usage

1.	+	Addition
2.	-	Subtraction
3.	*	Multiplication
4.	**	Exponentiation (2016)
5.	/	Division
6.	%	Modulus (Remainder)
7.	++	Increment
8.	--	Decrement

ASSIGNMENT OPERATORS

Sno	**Operator**	**Usage**
1.	=	Assigns a value
2.	+=	Adds a value to a variable.
3.	-=	Subtracts a value from a variable.
4.	*=	Multiplies a variable.
5.	/=	Divides a variable.
6.	%=	Assigns a remainder to a variable.

COMPARISON OPERATORS

Sno	**Operator**	**Usage**

1.	==	equal to
2.	===	equal value and equal type
3.	!=	not equal
4.	!==	not equal value or not equal type

RELATIONAL OPERATOR

Sno	**Operator**	**Usage**
1.	>	greater than
2.	<	less than
3.	>=	greater than or equal to
4.	<=	less than or equal to

LOGICAL OPERATOR

Sno	**Operator**	**Usage**
1.	&&	and
2.	||	or
3.	!	not

IDENTITY OPERATOR OR STRICT EQUALITY

X	Y	==	===
Undefined	Undefined	True	True

Null	Null	True	True
True	True	True	True
False	False	True	True
‘joes’	‘joes’	True	True
0	0	True	True
+0	-0	True	True
+0	0	True	True
-0	0	True	True
0	FALSE	True	False
“”	FALSE	True	False
“”	0	True	False
‘0’	0	True	False
‘15’	15	True	False
new String(‘joes’)	‘joes’	True	False
null	undefined	True	False

BITWISE OPERATOR

Bitwise Operators	
Bitwise AND (&)	**Bitwise AND assignment (&=)**
Bitwise OR (|)	**Bitwise OR assignment (|=)**
Bitwise NOT (~)	**~a=-a-1**
Bitwise XOR (^)	**Bitwise XOR assignment (^=)**
Left shift (<<)	**Left shift assignment (<<=)**
Right shift (>>)	**Right shift assignment (>>=)**
Unsigned right shift (>>>)	**Unsigned right shift assignment (>>>=)**

IF STATEMENT

Syntax

```
if(condition){
    -----
}
```

IF ELSE STATEMENT

Syntax

```
if(condition){
    -----
}
else{

}
```

ELSE IF STATEMENT(ELSE IF LADDER)

Syntax

```
if(condition)

{

    -----

}

elseif(condition)

{

    ----

}

else

{

    -----

}
```

NESTED IF STATEMENT

Syntax

```
if(condition)

{

    if(condition)

    {


    }

}
```

Own Example

New function

```
let avg=99.7346585958568939;
```

```
console.log(avg.toFixed(2));
console.log(avg.toExponential(2));
console.log(avg.toPrecision(3))
```

SWITCH STATEMENT

Syntax

```
switch(choice)
{
    case choice:
        ---
        break;
    case choice:
        ---
        break;
    default:
        ---
        break;
}
```

LOOPING STATEMENT

While loop

Syntax

```
while(coodition)
{
    //code execute
```

}

Do While loop

It is exit check loop

Syntax

```
do{
   //code to be executed if the condition is true}
}
while(condition);
```

For loop

```
for (initialize variable; condition; statement)
{
      //code to be executed
}
```

TEMPLATE STRING

Template Literals use back-ticks (` `) rather than the quotes (" ") to define a string

Quotes Inside Strings
With template literals, you can use both single and double quotes inside a string.

Multiline Strings
Template literals allows multiline strings

Interpolation

Template literals provide an easy way to interpolate variables and expressions into strings. The method is called string interpolation

Very Important methods in array

- foreach
- map
- slice
- splice
- concat
- sort
- fill
- includes
- join
- reverse
- push
- pop
- shift
- unshift
- indexof
- lastindexof
- every
- some

- ➢ find
- ➢ findindex
- ➢ from
- ➢ isArray
- ➢ filter
- ➢ Flat
- ➢ reduce

forEach

- This method calls a function for each element in an array.
- This method is not executed for empty elements.

map

- ➢ Creates a new array from calling a function for every array element.
- ➢ Calls a function once for each element in an array.
- ➢ Does not execute the function for empty elements.
- ➢ Does not change the original array.

slice

- ➢ returns selected elements in an array, as a new array.
- ➢ selects from a given start, up to a (not inclusive) given end.
- ➢ does not change the original array.

Splice

method adds and/or removes array elements.This method also overwrites the original array.

concat

Concatenates (joins) two or more arrays.returns a new array, containing the joined arrays, It does not change the existing arrays.

sort

sorts the elements of an array.Mainly this method sorts the elements as strings in alphabetical and ascending order. And this function overwrites the original array.

fill

The fill() method fills specified elements in an array with a value.Start and end position can be specified. If not, all elements will be filled.

includes

- This method returns **true** if an array contains a specified value.
- This method returns **false** if the value is not found.
- This is also case sensitive.

join()

The join() function in JavaScript is a method of the Array object, it is used to join all elements of an array into a single string. The elements of the array are separated by a specified delimiter or separator, which can be a string or a character.

The join() function can also be used to join arrays of numbers, booleans, and other data types. The join() method does not modify the original array, it only returns a new string.

It's important to note that this method will not work if the array contains undefined or null elements. If it's possible that some elements in the array are undefined or null, it's a good practice to filter them before using the join function.

It's important to note that the join() method does not modify the original array, it only returns a new string

reverse()

The reverse() function in JavaScript is a method of the Array object, it is used to reverse the order of the elements in an array. It modifies the original array in place, meaning that it changes the order of the elements in the original array, and it doesn't return a new array.

It's important to note that, this way of reversing the properties of an object is not a standard way and it's not a recommended one because it's not a good practice to modify the Array's prototype, and it may cause unexpected behavior.

push()

The push() function in JavaScript is a method of the Array object, it is used to add one or more elements to the end of an array. It modifies the original array in place, meaning that it adds new elements to the end of the original array, and it doesn't return a new array.

The push() method can also be used to add elements of any data type, including strings, booleans, and objects. It also increases the length of the array by the number of elements added.

It's important to note that, the push() method modifies the original array, it doesn't return a new array and you can use it to add elements to the end of an array regardless of its size

pop()

The pop() function in JavaScript is a method of the Array object, it is used to remove the last element from an array and returns the removed element. It modifies the original array in place, meaning that it removes the last element of the original array, and it doesn't return a new array.

It's important to note that, the pop() method modifies the original array and it also decreases the length of the array by 1. If you call the pop method on an empty array, it returns undefined.

shift()

The shift() function in JavaScript is a method of the Array object, is used to remove and return the first element of an array. It modifies the original array and changes its length. If the array is empty, undefined is returned.

unshift()

In JavaScript, the unshift() function is used to add one or more elements to the beginning of an array and returns the new length of the array. It modifies the original array by adding new elements to the beginning of the array.

indexOf()

In JavaScript, the indexOf() function JavaScript is used to search an array for a specific element and return the first index at which the element can be found. If the element is not present in the array, it will return -1.

lastIndexOf()

In JavaScript, the lastIndexOf() function JavaScript is used to search an array or a string for a specific element and return the last index at which the element can be found. If the element is not present in the array or string, it will return -1. The lastIndexOf() function is similar to the indexOf() function, but instead of searching from the beginning of the array or string, it starts searching from the end.

It is important to note that the lastIndexOf() method checks for strict equality (===) between the elements

every() & some()

In JavaScript, the every() and some() functions are used to perform a test on all elements of an array and return a Boolean value indicating whether all or some of the elements pass the test, respectively.

The every() function takes a callback function as an argument, which is called for each element in the array. The callback function is passed three arguments: the current element, the index of the current element, and the array itself. If the callback function returns true for every element in the array, the every() function returns true. If the callback function returns false for any element in the array, the every() function returns false.

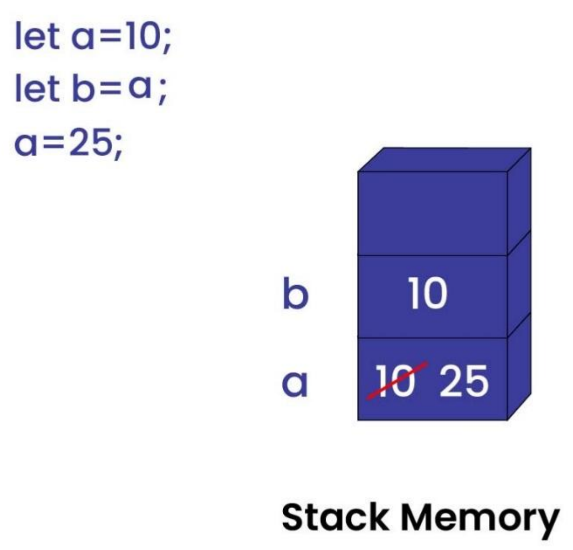
The some() function also takes a callback function as an argument, and it also calls it for each element in the array. If the callback function returns true for any element in the array, the some() function returns true. If the callback function returns false

THE DIFFERENCE BETWEEN PRIMITIVE AND REFERENCE DATA TYPES

Primitive data type

Primitive data types are stored in the stack memory, and when a variable is assigned a primitive value, it is assigned a copy of that value. This means that when you change the value of a primitive variable, it does not affect any other variables that have the same value.

Primitive Data Type



```
let a=10;
let b=a;
a=25;
```

```
let name ="Tutor Joes";  // String
let age =30;             // Number
letisStudent=false;  // Boolean
letx;                    // Undefined
let id =Symbol();        // Symbol
console.log(typeofname)
console.log(typeof age)
console.log(typeofisStudent)
console.log(typeof x)
console.log(typeof id)
```

Output:

```
script.js:6 string

script.js:7 number

script.js:8 boolean

script.js:9 undefined

script.js:10 symbol
```

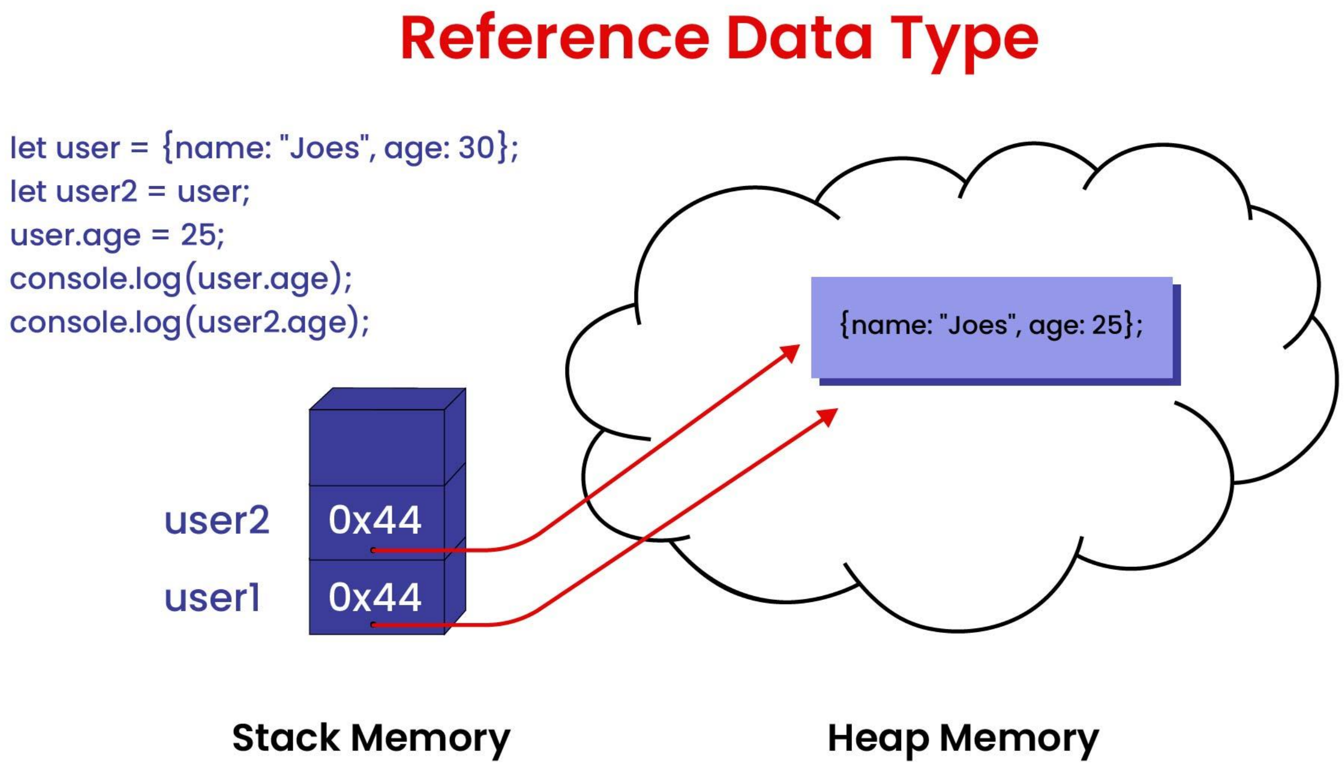
```
let a=10;
let b=a;
console.log("A: ",a,"B: ",b)
a=25;
console.log("A: ",a,"B: ",b)
```

Output:

```
script.js:3 A:  10 B:  10
script.js:5 A:  25 B:  10
```

Reference data type

Reference data types, on the other hand, are objects and arrays. They are stored in the heap memory, and when a variable is assigned a reference value, it is assigned a reference to the object or array in the heap. This means that when you change the value of a reference variable, it will affect any other variables that reference the same object or array.



Object clone in JavaScript

In references data type, this method is used to take copy of one variable2 and stored in other variable2, so change the variable1 therefore variable2 not affect. Some method are given below.

Object.assign() method

The Object.assign() method is used to copy the values of all enumerable own properties from one or more source objects to a target object. It returns the target object

```
const obj1 ={ a: 1, b: 2 };
const obj2 ={ c: 3, d: 4 };
const obj3 =Object.assign({}, obj1, obj2);
console.log(obj3); // { a: 1, b: 2, c: 3, d: 4 }
```

spread operator (…)

The spread operator (...) can also be used to create a shallow copy of an object

```
const obj1 ={ a: 1, b: 2 };
const obj2 ={ c: 3, d: 4 };
const obj3 = {...obj1, ...obj2};
console.log(obj3); // { a: 1, b: 2, c: 3, d: 4 }
```

Another example

```
letoriginalArray= [1, 2, 3];
letclonedArray= [...originalArray];
console.log(clonedArray);  // [1, 2, 3]
```

slice() method

```
letoriginalArray= [1, 2, 3];
letclonedArray=originalArray.slice();
console.log(clonedArray);  // [1, 2, 3]
```

concat() method

```
letoriginalArray= [1, 2, 3];
letclonedArray= [].concat(originalArray);
console.log(clonedArray);  // [1, 2, 3]
```

Array.from() method:

```
letoriginalArray= [1, 2, 3];
letclonedArray=Array.from(originalArray);
console.log(clonedArray);  // [1, 2, 3]
```

JSON.parse() and JSON.stringify():

Using the **JSON.parse() and JSON.stringify(): JSON.stringify()** method convert the javascript object into json format (that is string) and **JSON.parse()** method converts json string into javascript object.

```
letoriginalArray= [1, 2, 3];
letclonedArray=JSON.parse(JSON.stringify(originalArray));
console.log(clonedArray);  // [1, 2, 3]
```

It's important to note that all the above methods create a shallow copy of the array, which means it will copy the elements of the original array but not the objects inside the array