Let’s roll by 09:05 !!!

Javascript 1

Gopi Krishnan R

Topics to be Discussed

- What’s Javascript?
- How to use JS?
- Variable
- Identifiers
- Data Types
- Operators
- Function

Javascript

Javascript is a high level programming language. Together with HTML & CSS, JS is one of the three core technologies of the world wide web that is the internet.

Initially JS was implemented only on web browsers. Today it runs on the server, mobile, IOT devices, inside pdf readers etc

It is a very popular language now. Since most of the applications we use on a day to day basis run on the browsers now, Js is one language you cannot avoid in your journey to become a web developer.

Although there are similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design.

JS Inclusion

- Internal JS
- External JS

Variable and Scope

- Scope
  - Global scope
  - Block scope or local scope
- Variable
  - var
  - let
  - const

Identifiers

An identifier is simply a name. In JavaScript, identifiers are used to name variables and functions. A JavaScript identifier must begin with a letter, an underscore (_), or a dollar sign ($).

Data Types

The latest ECMAScript standard defines overall nine types:

- Six **Data Types** that are primitives, checked by typeof operator:
  - Number : typeof instance === "number"
  - String : typeof instance === "string"
  - Boolean : typeof instance === "boolean"
  - BigInt : typeof instance === "bigint"
  - Symbol : typeof instance === "symbol"
  - undefined : typeof instance === "undefined"
- Structural Types:
  - Object : typeof instance === "object". Special non-data but Structural type for any constructed object instance also used as data structures: new Object, new Array, new Map, new Set, new WeakMap, new WeakSet, new Date and almost everything made with new keyword;
  - Function : a non-data structure, though it also answers for typeof operator: typeof instance === "function". This is merely a special shorthand for Functions, though every Function constructor is derived from Object constructor.
- Structural Root Primitive:
  - null : typeof instance === "object". Special primitive type having additional usage for its value: if object is not inherited, then null is shown;

Operators

Arithmetic Operators

Javascript supports all the basic arithmetic operations - * (multiplication), / (division), % (modulo: remainder after division), + (addition), and - (subtraction).

Unary Arithmetic Operators

Unary operators modify the value of a single operand to produce a new value.

Increment (++) - The ++ operator increments (i.e., adds 1 to) its single operand, which must be an lvalue (a variable, an element of an array, or a property of an object). The operator converts its operand to a number, adds 1 to that number, and assigns the incremented value back into the variable, element, or property.

Decrement (--) - The -- operator expects an lvalue operand. It converts the value of the operand to a number, subtracts 1, and assigns the decremented value back to the operand.

Relational Operations

Relational expressions always evaluate to a boolean value, and that value is often used to control the flow of program execution in if , while , and for statements.

The == and === operators check whether two values are the same, using two different definitions of sameness.

Comparison Operations

The comparison operators test the relative order (numerical or alphabetics) of their two operands:

Less than (<) - The < operator evaluates to true if its first operand is less than its second operand; otherwise it evaluates to false.

Greater than (>) - The > operator evaluates to true if its first operand is greater than its second operand; otherwise it evaluates to false.

Less than or equal (<=) - The <= operator evaluates to true if its first operand is less than or equal to its second operand; otherwise it evaluates to false.

Greater than or equal (>=) - The >= operator evaluates to true if its first operand is greater than or equal to its second operand; otherwise it evaluates to false.

Logical Operations

The logical operators && , || , and ! perform Boolean algebra and are often used in conjunction with the relational operators to combine two relational expressions into one more complex expression.

instanceof Operator

The instanceof operator expects a left-side operand that is an object and a right-side operand that identifies a class of objects. The operator evaluates to true if the left-side object is an instance of the right-side class and evaluates to false otherwise.

typeof Operator

The typeof operator returns the type of the argument. It’s useful when we want to process values of different types differently or just want to do a quick check.

It supports two forms of syntax:

1. As an operator: typeof x.
2. As a function: typeof(x).

Function

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: {}

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Let’s roll by 09:05 !!!

Javascript 2

Gopi Krishnan R

Topics to be Discussed

- JS Conditional Statement
- Loops
- Javascript Jump Statement
- Function

JS Conditional Statement

- if else
- switch

JS Loops

- for loop
- while loop
- do/while loop
- for/in loop

JS Jump Statement

- break
- continue
- return

Function

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: {}

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Let’s roll by 09:05 !!!

Javascript 3

Gopi Krishnan R

Topics to be Discussed

- Regular Function
- Callback Function
- Anonymous Function

Function

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: {}

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Callback 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. It’s the combination of these two that allow us to extend our functionality.

- Synchronous
- Asynchronous

```
function createQuote(quote, callback){
  var myQuote = "Like I always say, " + quote;
  callback(myQuote); // 2
}

function logQuote(quote){
  console.log(quote);
}

createQuote("eat your vegetables!", logQuote); // 1

// Result in console:
// Like I always say, eat your vegetables!
```

Synchronous

Asynchronous

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())
  .then(json => console.log(json))
```

Anonymous Function

The meaning of the word 'anonymous' defines something that is unknown or has no identity. In JavaScript, an anonymous function is that type of function that has no name or we can say which is without any name. When we create an anonymous function, it is declared without any identifier. It is the difference between a normal function and an anonymous function. Not particularly in JavaScript but also in other various programming languages also. The role of an anonymous function is the same.

```
let x = function () {
    console.log('It is an anonymous function');
};
x();
```

Another benefit of anonymous functions are the are **not preserved in the memory like named functions**. As soon as the use/calling of anonymous functions is done it is destroyed from the memory. However, this is not the case with named functions. **If a named function is declared, it remains in the memory even if it is not being used**.

Let’s roll by 09:05 !!!

Javascript 4

Gopi Krishnan R

Topics to be Discussed

- Arrays
- Ways to Declare Array in JS
- JS Array inbuilt function

Array

An array is an ordered collection of values. Each value is called an element, and each element has a numeric position in the array, known as its index. JavaScript arrays are untyped: an array element may be of any type, and different elements of the same array may be of different types.

Create and Use Array

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]; //most used method
var arr = new Array(); //another less used method
```

```
var fruits = ["Apple", "Orange", "Plum"];
alert( fruits[0] ); // Apple
alert( fruits[1] ); // Orange
alert( fruits[2] ); // Plum
```

We can replace an element:

```
fruits[2] = 'Pear'; // now ["Apple", "Orange", "Pear"]
```

...Or add a new one to the array:

```
fruits[3] = 'Lemon'; // now ["Apple", "Orange", "Pear", "Lemon"]
```

An array can store elements of any type.

```
// mix of values
var arr = [ 'Apple', { name: 'John' }, true, function() { alert('hello'); } ];

// get the object at index 1 and then show its name
alert( arr[1].name ); // John

// get the function at index 3 and run it
arr[3](); // hello
```

Methods in Javascript

- push() and pop()
  - The push() method appends one or more new elements to the end of an array and returns the new length of the array. The pop() method does the reverse: it deletes the last element of an array, decrements the array length, and returns the value that it removed.
- unshift() and shift()
  - The unshift() and shift() methods behave much like push() and pop() , except that they insert and remove elements from the beginning of an array rather than from the end.
- toString()
  - An array, like any JavaScript object, has a toString() method. For an array, this method converts each of its elements to a string (calling the toString() methods of its elements, if necessary) and outputs a comma-separated list of those strings.

.join()

The Array.join() method converts all the elements of an array to strings and concatenates them, returning the resulting string.

```
var a = [1, 2, 3];

a.join();

a.join(" ");
```

.reverse()

Reverses an array. It also returns the array arr after the reversal. The reversal is done in-place.

```
let arr = [1, 2, 3, 4, 5];

arr.reverse();

alert( arr ); // 5,4,3,2,1
```

.concat()

The Array.concat() method creates and returns a new array that contains the elements of the original array on which concat() was invoked, followed by each of the arguments to concat().

```
var a = [1,2,3];

a.concat(4, 5)

a.concat([4,5]);
```

.slice()

The Array.slice() method returns a slice, or subarray, of the specified array. Its two arguments specify the start and end of the slice to be returned.

```
var a = [1,2,3,4,5];

a.slice(0,3); // Returns [1,2,3]
```

.splice()

The Array.splice() method is a general-purpose method for inserting or removing elements from an array. Unlike slice() and concat() , splice() modifies the array on which it is invoked. Note that splice() and slice() have very similar names but per- form substantially different operations.

splice() can delete elements from an array, insert new elements into an array, or perform both operations at the same time.

```
var a = [1,2,3,4,5,6,7,8];

a.splice(4);                // Returns [5,6,7,8]; a is [1,2,3,4]

a.splice(1,2); // Returns [2,3]; a is [1,4]

a.splice(1,1); // Returns [4]; a is [1]
```

.forEach()

The arr.forEach method allows to run a function for every element of the array.

Syntax:

```
arr.forEach(function(item, index, array) {

  // ... do something with item

});
```

.map()

The arr.map method is one of the most useful and often used. It calls the function for each element of the array and returns the array of results.

Syntax:

```
let result = arr.map(function(item, index, array) {

  // returns the new value instead of item

});
```

.sort()

Array.sort() sorts the elements of an array in place and returns the sorted array. When sort() is called with no arguments, it sorts the array elements in alphabetical order.

If an array contains undefined elements, they are sorted to the end of the array. To sort an array into some order other than alphabetical, you must pass a comparison function as an argument to sort() . This function decides which of its two arguments should appear first in the sorted array. If the first argument should appear before the second, the comparison function should return a number less than zero. If the first argument should appear after the second in the sorted array, the function should return a number greater than zero. And if the two values are equivalent (i.e., if their order is irrelevant), the comparison function should return 0.

```
var numbers = [2, 5, 1, 3, 8, 9, 4, 2, 1];

numbers.sort( function(a, b) { return a - b; } )
```

filter()

The filter() method returns an array containing a subset of the elements of the array on which it is invoked. The function you pass to it should be predicate: a function that returns true or false . The predicate is invoked just as for forEach() and map() . If the return value is true , or a value that converts to true , then the element passed to the predicate is a member of the subset and is added to the array that will become the return value. Examples:

```
a = [5, 4, 3, 2, 1];

smallvalues = a.filter(function(x) { return x < 3 });          // [2, 1]

everyother = a.filter(function(x,i) { return i%2==0 });      // [5, 3, 1]
```

every() and some()

The every() and some() methods are array predicates: they apply a predicate function you specify to the elements of the array, and then return true or false.

The every() method is like the mathematical "for all" quantifier ∀: it returns true if and only if your predicate function returns true for all elements in the array:

```
a = [1,2,3,4,5];

a.every(function(x) { return x < 10; });          // => true: all values < 10.

a.every(function(x) { return x % 2 === 0; });     // => false: not all values even.
```

The some() method is like the mathematical "there exists" quantifier ∃: it returns true if there exists at least one element in the array for which the predicate returns true, and returns false if and only if the predicate returns false for all elements of the array:

```
a = [1,2,3,4,5];

a.some(function(x) { return x%2===0; });    // => true a has some even numbers.
```

reduce(), reduceRight()

The reduce() and reduceRight() methods combine the elements of an array, using the function you specify, to produce a single value. This is a common operation in functional programming and also goes by the names "inject" and "fold".

```
a = [1,2,3,4,5]''

sum = a.reduce(function(x,y) { return x+y }, 0);          // Sum of values

product = a.reduce(function(x,y) { return x*y }, 1);      // Product of values

max = a.reduce(function(x,y) { return (x>y)?x:y; });      // Largest value
```

indexOf() and lastIndexOf()

indexOf() and lastIndexOf() search an array for an element with a specified value, and return the index of the first such element found, or –1 if none is found. indexOf() searches the array from beginning to end, and lastIndexOf() searches from end to beginning.

```
a = [0,1,2,1,0];

a.indexOf(1);                    // => 1: a[1] is 1

a.lastIndexOf(1);          // => 3: a[3] is 1

a.indexOf(3);                    // => -1: no element has value 3
```