Basic JavaScript

CodeMaster Noyon

JS Variables

Variables are containers for Storing data. Mainly, there are 2 ways to declare variables.

JavaScript variables are dynamically type.

1. Using keyword
2. Without keyword

Note: It’s a good programming practice to always use keyword before declare variables.

There are 3 ways to declare variables using keyword.

- Using `var`
- Using `let`
- Using `const`

Example:

```
var Name = ‘Noyon’;
let Age = 23;
const Gender = ‘Male’;
```

CodeMaster Noyon

JS Variables

All JavaScript variables must be identified with unique names. These unique names are called identifiers.

Note: Every identifier is called variable but every variable is not called identifier. Variables are case-sensitive.

```
let Name = ‘Noyon’;
let name = ‘Sarker’;
```

Here,
Name & name are not same

`let firstName;` → variable declaration.

`firstName = ‘Noyon’;` → value assign.

`var country = ‘Bangladesh’;` → declaration & assign in the same line.

JS Variable: var

var introduced before 2015.

1. var not mandatory to declare before use.

```
var carName;
carName = ‘Volvo’;
```

```
carName = ‘Volvo’;
var carName;
```

2. var has global scope.

```
{
    let x = 10;
};
// x can be used here
```

3. var can be re-declare and re-assign in the same scope.

```
{
    var x = 10;
    var x = 50;
};
```

Re-declared

```
{
    let x = 10;
    x = 50;
};
```

Re-assign

CodeMaster Noyon

JS Variable: let

The let keyword was introduced in ES6.

1. let must be declared before use.

```
let carName;
carName = ‘Volvo’;
```

```
carName = ‘Volvo’;
let carName;
```

2. let have Block Scope.

```
{
   let x = 10;
};
// x can not be used here
```

3. let can not be redeclared in the same scope. But re-assign possible.

```
{
   let x = 10;
   let x = 50;
};
```

Re-declared

```
{
   let x = 10;
};
let x = 50;
```

Re-declared

```
{
   let x = 10;
   x = 50;
};
```

Re-assign

CodeMaster Noyon

JS Variable: const

The const keyword was introduced in ES6.

1. const must be assigned a value when it declared.

```
const carName;
carName = ‘Volvo’;
```

```
const carName = ‘BMW’;
```

2. const have Block Scope.

```
{
   const x = 10;
};
// x can not used here
```

3. const can not be redeclared in the same scope. Not also re-assign possible.

```
{
   const x = 10;
   const x = 50;
};
```

Re-declared

```
{
   const x = 10;
};
const x = 50;
```

Re-declared

```
{
   const x = 10;
   x = 50;
};
```

Re-assign

CodeMaster Noyon

JS Variables: const

Object & Array:

When we declared an object or array with const keyword. It means we declared variable name const, not it’s elements or properties.

```
const myArray = [1,2,3,4,5];
myArray.push(7);
Output: [1,2,3,4,5,7];
```

```
const myArray = [1,2,3,4,5];
//Error
const myArray = [5,7,4];
```

```
const Person = {
    name: ‘Noyon’,
    age: 24,
};
Person.name = ‘Mithun’;
//now name property’s value is
Mithun
```

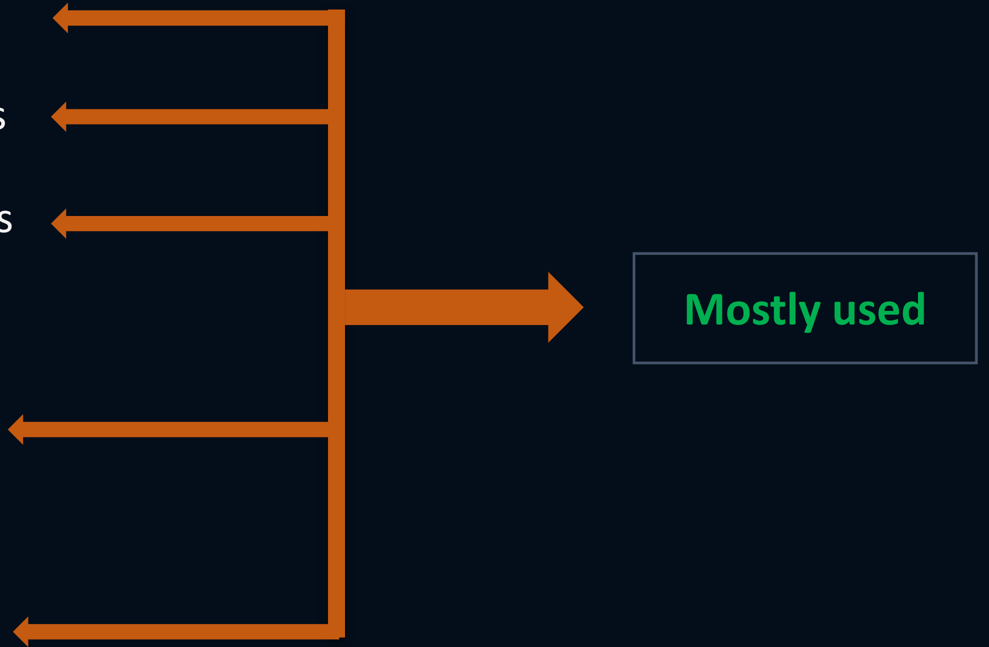
```
const Person = {
    name: ‘Noyon’,
    age: 24,
};
const Person = { //error
Roll: 2137527,
};
```

CodeMaster Noyon

JS Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators



Mostly used

JS Arithmetic Operators

Operand	Operator	Operand
100	+	50

A typical arithmetic operation operates on numbers or variables.

Those numbers or variables called operands.

Suppose, there are two variables.

```
let a = 10;
let b = 5;
```

Addition '+':

a + b = 15;

Subtraction '-':

a – b = 5

Multiplication '*':

a * b = 50

Division '/':

a / b = 2

Remainder '%':

a % b = 0;

Increment '++':

a++;

// now a = 11

Decrement '--':

b--;

//now b = 4

Exponentiation '**':

2 ** 3 = 8;

//means 2 multiplying 3 times

Operators priority: computed from left to right

Multiplication → Division → Addition → subtraction

Example: 100 – 4 * 3 + 6 / 2; ➔ 100 – 12 + 3; ➔ 100 – 15; ➔ 85;

CodeMaster Noyon

JS Assignment Operators

Operator	Example	Same as	Description
=	x = 5	x = 5	X stores 5
+=	x += 5	x = x+5	First, add x and 5, then store the to the x.
-=	x -= 5	x = x-5	First, sub x and 5, then store the to the x.
*=	x *= 5	x = x*5	First, multiply x and 5, then store the to the x.
/=	x /= 5	x = x/5	First, divide x and 5, then store the to the x.
%=	x %= 5	x = x%5	First, find remainder, then store the to the x.
**=	x **= 5	x = x ** 5	First multiply x, 5 times then store to the x

CodeMaster Noyon

JS Comparison Operators

Equal ‘==‘:

5 == 5 → It returns true.

5 == 4 → It returns false.

Not Equal ‘!=‘:

5 != 4 → It returns true.

5 != 5 → It returns false.

5 !== ‘5’ → It returns true, data type different.

Less ‘<‘:

5 < 10 → It returns true.

5 < 3 → It returns false.

Greater ‘>’:

5 > 3 → It returns true.

10 > 15 → It returns false

Different between ‘==‘ and ‘===‘:

‘==‘ just compare between two variable’s values.

Example: 5 == ‘5’; // returns true

‘===‘ compare between two variable’s values and data types.

Example: 5 === ‘5’; // returns false

Note: We can’t compare between two objects.

CodeMaster Noyon

JS Logical Operators

Logical AND (&&):

`(Condition1 && condition2)` → when both are produce true statement returns true.

Logical OR (||):

`(condition1 || condition2)` → when any one condition produce true statement returns true.

Logical NOT (!):

`(! condition)` → It returns opposite value. If condition produce true statement returns false.

CodeMaster Noyon

JS Ternary & Type Operators

Ternary operator is a short-hand of `if-else` condition.

```
(condition) ? statement1 : statement2 ;
```

➔If, condition produce true then statement1 will be execute. Else, statement2 will be execute.

Type operator:

Operator	**Description**
typeof	Returns the type of a variable
instaceof	Returns true if an object is an instance of an object type

CodeMaster Noyon

JS Data Types

There are mainly two data types.

- Primitive
- Non-Primitive

Primitive has 7 types:

- Number // 1243, 50.45
- String // ‘Noyon’, ’13’
- Boolean // true, false
- Null // variable has a value but value is null value
- Undefine // variable without a value, has the value undefine
- BigInt
- Symbol.

Non-Primitive has just one type:

- Object // Arrays, Objects

null and undefine are the special value in JavaScript.

CodeMaster Noyon

JS Functions

A JavaScript function is a block of code designed to perform a particular task.

Function Syntax:

1. Define with function keyword, followed by a name, followed by parentheses.
2. The parentheses may include parameters.

```
function addValue (parameter1, para…) {
          // code to be executed
     };
```

JS Functions

Function Definition:

```
function printName(){
    console.log('My name is Noyon.');
};

printName(); // function call
```

Note: When you call the function, ‘*My name is Noyon*’ will be printed. Function can used as variable values. We will know details about function in the next content.

```
const x = function printName(){
    console.log('My name is Noyon.');
};
x(); // function call
```

CodeMaster Noyon

JS Functions ‘Return’

Generally, function written for re-use the code.

When function reaches a return statement, the function will stop executing.

```
function addTwoNumber (a, b){
    return a + b;   // adds the value of a & b, Then returns the value.
    console.log('Hi, I am a function’);   // this line will not execute.
};
```

Function’s value can stored in a variable. Here, function is re-used.

```
let x = addTwoNumber (5, 10); //function takes two values as arguments then returns the result. x hold the result.
let y = addTwoNumber (10, 20); // Here function was re-called means re-used. This time y hold the result.
```

CodeMaster Noyon

JS Strings

Strings are for storing text. Strings are written with quotes. You can use single or double quotes. There is no different.

```
let Name = 'Noyon Sarker’;   // it’s a string.
let x = ‘1234’;   // it’s also a string.
```

Strings can also be defined as objects with the keyword new :

```
let Name = ‘I am a string’; // It’s a primitive value.
let b = new String(‘I am a string’);  // It’s a non-primitive value. It’s a object.
let c = new String(‘I am  a string');   // It’s a non-primitive value. It’s a object.

a == b;     // produce  true, because it just check the value.
a ===  b;     // produce  false, because it check the value & data-type.

b == c, b === c;   // Both produce  false,
       // both are objects and comparing two JavaScript objects always returns false.
```

CodeMaster Noyon

JS Strings Methods

JavaScript strings are primitive and immutable (not changeable). All strings methods produce a new string without altering the original string.

There are 3 methods for extracting a part of a string:

- slice (start-index, end-index); // end not included
- substring (start-index, end-index); // end not included
- substr (start-index, length);

```
slice(start, end)
```

```
let Name = 'NoyonSarker';
let newStr = Name.slice(2, 6);
console.log(newStr);   // yonS
```

```
substr(start, length)
```

```
let Name = 'NoyonSarker';
let newStr2 = Name.substr(2, 6);
console.log(newStr2); // yonSar
```

CodeMaster Noyon

JS Strings Methods

```
let Name = 'Noyon sarker’;


slice(x);  →  Name.slice(2);  // yon sarker

slice(-end, -start);  → Name.slice(-5, -2);  // ‘-’ index count from last to first index. It returns ark

slice(-x);  → Name.slice(-5); //  arker

charAt(x);  → Name.charAt(0);  // It returns the alphabet for the given index. N

at(x);  →  Name.at(1);  // similar to the charAt() method. o

charCodeAt(x);  → Name.charCodeAt(3);   // returns the code of the character at a specific index. 111
```

JS Strings Methods

JavaScript strings are primitive and immutable (not changeable). All strings methods produce a new string without altering the original string.

let Name = ‘   Noyon ’, text = 'Hi, I am an Engineer’; // variables also written this way.

- toUpperCase(); → text.toUpperCase(); // HI, I AM AN ENGINEER
- toLowerCase(); → text.toLowerCase(); // hi, i am an engineer
- concat(); → text.concat(123); // Hi, I am an Engineer123
- trim(); → Name.trim(); // trim () method remove white space. There are trimStart(), trimEnd()
- repeat(); → Name. repeat(2); // It’s return 2 times. **NoyonNoyon**
- Replace(x, y); → Name.replace('o', '-’); // replace 1st single word or letter. **N-yon**
- replaceAll (x, y); → Name.replaceAll('o', '-’); // replace all the words or letter. **N-y-n**
- split(x); → Name.split(‘o’); // split() method split the Name string,

  where it finds ‘o’ and returns an **Array** [" N", "y", "n"]

CodeMaster Noyon

JS Strings Search

`let Name = ‘Noyon Sarker’;`

JavaScript counts position from zero (0).

- `indexOf(x);` → `Name.indexOf('o’);` // It returns a index no. of first ‘o’. 1
- `lastIndexOf(x);` → `Name.lastIndexOf(‘o’);` // It returns a index no. of last ‘o’. 3
- `search(x);` → `Name.search('o’);` // It’s similar to indexOf() method. 1
- `match(x), matchAll(x);` → The match() & matchAll() return an array containing the result.
- `includes(x);` → `Name.includes('p’);` // returns Boolean value, If the argument is included in the string. Returns `false`
- `startsWith(x), endsWith(x);` → `Name.startsWith('N’);` // These are also return Boolean value. `true`

JS Strings Templates

Template Strings use back-ticks (` `). Template Strings allow both single and double quotes inside a string.

Template Strings allow multiline strings.

```
let text = `He's often called Mithun`;
```

```
let text2 = `He's often
called Mithun`;
```

Interpolation:

Template String provide an easy way to interpolate variables into the strings. It's called string Interpolation.

```
let age = 25;
let text = `He's often called Mithun. His age
is ${age}`;
```

Output: He's often called Mithun. His age is 25.

In JavaScript String, there is one string property.

It is length. I returns string length.

```
let Name = 'Noyon';
console.log(Name.length);   //  string size is 5.
```

CodeMaster Noyon

JS Number

Number can be integer or float.

```
let a = 10,  b = 20.25;  //  Both are number type
let x = '20';  //  It's a string. Because there is used quote

 console.log( a + b );  //  It returns 30.25 & type will be Number.
 console.log( a + x );  //  It returns 1020 & type will be String.
 console.log('The result is: ' + a + b);  //  The result is: 1020.25
```

In JavaScript, there is a global function `isNaN()`. To find out the value is Number or NOT.

```
let value = (a / 'Hi');  //  It produce NaN. Means Not a Number.
isNaN(value). //  It produce true. Because value is NaN.
```

Number can also be defined as objects with the keyword `new` . But it's not good programming practice.

JS Number Methods

These number methods are mostly used.

- `toString();`
- `toFixed();`
- `isInteger();`   // It check the number is integer or NOT
- `parseFloat();`   // It parses a string and returns a number.

```
let num = 15.5684;
```

- `num.toString();` // It returns ‘15.5674’ & data type will be string.
- `num.toFixed(2);` // It returns 15.57 & data type will be number.
- `Number.isInteger(num);`   // returns `false`
- `Number.parseFloat('10’);    //  returns 10 & typeof Number.`

CodeMaster Noyon

JS Arrays

An array is a special variable, which can hold more than one value. JavaScript arrays are alterable. Why use Arrays ?

Suppose, you have a list of fruits. Then the easiest way to storing all the fruits is Array.

```
let fruits = ['mango', 'orange', 'apple', 'banana'];
```

You can also create an array using `new` keyword.

```
const cars = new Array('Saab', 'Volvo', 'BMW');
```

Accessing Array Elements: You can access an array element by referring to the index number. Index start with 0.

```
console.log(cars[1]);  // Volve
console.log(cars[0]);  //  Saab
```

Changing an Array Element

```
cars[0] = 'Toyota’;
```

Now, array elements are: ["Toyota", "Volvo", "BMW"]

JS Arrays Methods

These array methods mostly used in JavaScript.

- toString()
- push()
- pop()
- shift()
- unshift()
- join()
- delete()
- concat()
- splice()
- slice()
- at()

```
let fruits = ['mango', 'orange', 'apple', 'banana’];
```

toString()

toString() method convert any type of data to string.

```
console.log(fruits.toString());     // mango,orange,apple,banana
```

push()

The push(item) method adds new element to an array at the end. push() method returns new array length.

```
fruits.push('Lichi’);    →   console.log(fruits);   //  [ "mango", "orange", "apple", "banana", "Lichi" ]
```

CodeMaster Noyon

JS Arrays Methods

pop()

The pop() method deletes the element from end of the array. pop() method returns the removed item.

```
fruits.pop();   → console.log(fruits);   // [ "mango", "orange", "apple" ]
```

unshift()

The unshift(item) method adds new element to an array at the beginning. unshift() returns new array length.

```
fruits.unshift('Lichi’);   →  console.log(fruits);   // [ "Lichi", "mango", "orange", "apple", "banana" ]
```

shift()

The shift() method remove element from the beginning of the array. shift() returns the deleted element.

```
fruits.shift();   →  console.log(fruits);   // [ "orange", "apple", "banana" ]
```

JS Arrays Methods

delete fruits[index]

Using delete leaves undefined holes in the array. It does not remove the slot. Using delete is not a good Practice.

```
delete fruits[1];   →   console.log(fruits);   // [ "mango", <1 empty slot>, "apple", "banana" ]
```

concat()

The concat() method creates a new array by merging existing arrays.

```
let cars = ['Saab', 'Toyota’];
let newArray = fruits.concat(cars); → console.log(newArray); //[ "mango", "orange", "apple", "banana", "Saab", "Toyota" ]
```

slice(para)

The slice(para) method creates a new array. Parameter says, how many items can you slice out from the beginning.

```
let newArray = fruits.slice(2);   → console.log(newArray);   // [ "apple", "banana" ]
```

CodeMaster Noyon

JS Arrays Methods

slice(para1, para2, newItem)

The slice(para1, para2, items) method can be used to add new items to an array.

The 1st para1 defines the position where new elements should be added. 2nd para2 defines how many elements should be removed.

```
fruits.slice(1, 1, 'Lichi’);   → console.log(fruits)   //[ "mango", "Lichi", "apple", "banana" ]
```

join()

The join() method convert an array to the String & each element is connect with the given parameter.

```
console.log(fruits.join('*’))   →   // mango*orange*apple*banana
```

at()

The at() method used for accessing the array. It’s similar of [index].

```
console.log(fruits.at(2)); console.log(fruits[2]); // Both are similar. Both returns apple
```

CodeMaster Noyon

JS Arrays Search

These are the search methods to find value from an array.

`indexOf();` `lastIndexOf();` `includes();`

```
let fruits = ['mango', 'orange', 'apple', 'mango' 'banana’];
```

indexOf(item)

The `indexOf()` method searches for an element value and returns its first position. If the value is not exist, it returns -1

```
console.log(fruits.indexOf('mango’)); // returns 0
```

lastIndexOf(item)

The `lastIndexOf()` method is the same as `indexOf()`, but returns the its last element position.

```
console.log(fruits.indexOf('mango’)); // returns 3
```

includes(item)

The *`includes()`* method returns the Boolean value. If the value exist in the array then it returns *`true`*.

```
console.log(fruits.includes('orange’)); // returns true.
```

CodeMaster Noyon

JS Arrays Sort

In JavaScript, there are two type of sorting. Alphabetic sort & Numeric sort.

Alphabetic sorting: 1. sort(); 2. toSorted(); 3. reverse() 4. toReversed();

```
let fruits = ['mango', 'orange', 'apple', 'mango' 'banana’];
```

sort()

The sort() method sorts an array alphabetically by ascending order. sort() method can altering the original array.

```
console.log(fruits.sort());    // [ "apple", "banana", "mango", "mango", "orange" ]
```

toSorted()

The toSorted() method is similar to sort(). It can sort the array without altering the original array. It returns a new array.

```
let newArr = fruits.toSorted();    →    console.log(newArr); // [ "apple", "banana", "mango", "mango", "orange" ]
```

reverse()

The reverse() method reverses the elements in an array.

```
console.log(fruits.reverse());    // [ "banana", "mango", "apple", "orange", "mango" ]
```

CodeMaster Noyon

JS Arrays Sort

toReversed()

`toReversed()` method as a safe way to reverse an array elements. It reverse element without altering original array. Returns a new array.

```
let newArr = fruits.toReversed();  → console.log(newArr); // [ "banana", "mango", "apple", "orange", "mango" ]
```

Sort an array by descending order.

Step- 01: First sort the array by ascending order.

Step- 02: Then reverse the array.

```
let fruits = ['mango', 'orange', 'apple', 'banana’];
```

With altering array

```
fruits.sort();
fruits.reverse();
console.log(fruits);
 [ "orange", "mango", "banana", "apple" ]
```

Without altering array

```
let newArr = fruits.toSorted();
newArr.reverse();   or    let newArr2 = newArr.toReversed();
console.log(newArr);    or  console.log(newArr2);
          [ "orange", "mango", "banana", "apple" ]
```

CodeMaster Noyon

JS Arrays Sort[Numeric]

The `sort()` method sorts values as strings. `sort()` method produce incorrect result. You can fix this by providing a compare function.

```
let numbers = [3, 4, 2, 7, 5, 8, 12, 10, 9];
```

Ascending Sort

```
function numberSort (a, b){  //  compare function
    return a - b;
};

console.log(numbers.sort(numberSort));  // [ 2, 3, 4, 5, 7, 8, 9, 10, 12 ]
```

Descending Sort

For descending order change the return statement.

```
console.log(numbers.sort((a, b)=>{
    return b – a;   // you can also write function in the sort method
})); // [ 2, 3, 4, 5, 7, 8, 9, 10, 12 ]
```

CodeMaster Noyon

JS Arrays Sort[Numeric]

Find the minimum & maximum value from an array.

```
let numbers = [3, 4, 2, 7, 5, 8, 12, 10, 9];
```

There are two process:

1. Sorts the array ascending or descending order then find first or last element.
2. Use `Math.min()` or `Math.max()` methods.

```
console.log(Math.min.apply(null, numbers));   // returns minimum value of array is 2

console.log(Math.max.apply(null, numbers));   // returns maximum value of array is 12
```

JS Arrays Iteration

Array iteration methods operate on every array item. These methods are mainly used for iteration.

```
forEach();                    map();              filter();
reduce();                     every();            some();
entries();                    spread();           from();
```

forEach()

The *forEach()* method calls a callback function that callback take 3 arguments value, index, array. If you pass one argument then callback takes array's element.

```
let numbers = [3, 4, 2, 7, 5, 8, 12, 10, 9];

numbers.forEach((value, index, array)=>{
    console.log(value, index);   // It displays every element & index first to last.
});
numbers.forEach((value)=>{
    console.log(value);   // It displays every element from first to last.
});
```

CodeMaster Noyon

JS Arrays Iteration

map()

The *map()* method creates a new array. The map() method does not change the original array.

```
let newArr = numbers.map((item)=>{

    return item * 2; //pick every element and multiple by 2 with everyone

});
console.log(newArr); // [ 6, 8, 4, 14, 10, 16, 24, 20, 18 ]
```

filter()

The *filter()* method also create a new array.

```
let newArr = numbers.filter((value)=>{

    return value % 2 == 0;   // filtering the even value from array.

});

console.log(newArr);   //  [ 4, 2, 8, 12, 10 ]
```

CodeMaster Noyon

JS Arrays Iteration

reduce()

The *reduce()* method catch every value and return a single value. It works from left-to-right.

```
let singleValue = numbers.reduce((total, value)=>{
    return total + value;
});
console.log(singleValue);   // returns 60
```

Note: first time, total has 3 & value has 4. Then add 4 & 3, and return the result to the total. Now, total has 7, value has next value 2.

every()

The *every()* method checks if all array values pass a test. It returns a Boolean value.

```
let result = numbers.every((value)=>{
    return value > 5;
});
console.log(result);    // it returns false, because every element is not large from 5
```

CodeMaster Noyon

JS Arrays Iteration

some()

The *some()* method checks if some array values or any one values pass a test. It also returns Boolean value.

```
let numbers = [3, 4, 2, 7, 5, 8, 12, 10, 9];

let result = numbers.some((value)=>{
    return value > 15;
});
console.log(result); // it returns  false, because every element is smaller than 15.
```

form()

The *form()* method returns an array object. It’s create an array from a string.

```
let Name = 'Noyon’;
let newArr = Array.from(Name);
console.log(newArr);  //    [ "N", "o", "y", "o", "n" ]
```

JS Arrays Iteration

entries()

The *entries()* method returns an array Iterator object array with key/value (index, element) pairs. Every time it returns different array.

```
let fruits = ['apple', 'mango', 'banana', 'orange'];

const pairs = fruits.entries();
for(x of pairs){
    console.log(x);   //  [ 0, "apple" ];  [ 1, "mango" ];  [ 2, "banana" ];  [ 3, "orange" ];
};
```

spread(...)

The . . . or spread operator also use for iterating an iterable object.

```
let fruits = ['apple', 'mango', 'banana', 'orange'];
let Name = 'Noyon Sarker’;

console.log(...fruits); // apple mango banana orange
console.log(...Name);    //   N o y o n S a r k e r
```

CodeMaster Noyon

JS Math

JavaScript Math object allows you to perform mathematical tasks on numbers.

round() *ceil()* *floor()* *trunc()*

`Math.round(x)` returns the nearest integer.

```
Math.round(5.5);  // 6       Math.round(5.4);  // 4
```

```
Math.ceil(x) returns the value of x rounded up to its nearest integer.
Math.ceil(4.5);  // 5       Math.ceil(4.1);  // 5       Math.ceil(4.9);  // 5
```

```
Math.floor(x) returns the value of x rounded down to its nearest integer.
Math.floor(7.4);  // 7       Math.floor(7.9);  // 7
```

```
Math.trunc(x) returns the integer part of x.
Math.trunc(6.2);  // 6       Math.trunc(5.1);  // 5
```

CodeMaster Noyon

JS Math

JavaScript Math object allows you to perform mathematical tasks on numbers.

pow() *sqrt()* *abs()* *min()* *max()* *random()*

`Math.pow(x, y)` returns the value of x to the power y: → `Math.pow(2,3); // 8`

`Math.sqrt(x)` returns the square root of x: → `Math.sqrt(25); // 5`

`Math.abs(x)` returns the absolute (positive) value of x: → `Math.abs(-9); // 9`

`Math.min()` & `Math.max()` can be used to find the lowest or highest value in a list of arguments.

`Math.min(10, 34, 2, 5); // 2` `Math.max(6, 3, 6, 28, 3); // 28`

`Math.random()` returns a random number between 0 (inclusive) and 1 (exclusive).

Like: // 0.6172410378731809 // 0.9876887972124565

CodeMaster Noyon

JS if, else

Conditional statement are used to perform different actions based on different conditions.

```
if(condition){
    // block of code to be executed, when condition is true.
}else{
    //block of code to be executed, when condition is false.
};
```

Example:

```
if(true){
    console.log('If block is executed.');
}else{
    console.log('Else block is executed.');
};
   //   If block is executed.
```

JS if, else

else if statement to specify a new condition if the first condition is false.

```
if(condition-1){
        // block of code to be executed, when condition-1 is true.
}else if(condition-2){
        //block of code to be executed, when condition-1 is false and condition-2 is true.
} else if(condition-3){
  //block of code to be executed, when condition-1 & condition-2 both are false and condition-3 is true.
}else{
        //block of code to be executed, when condition-1,2,3 are false
};
```

If any condition is true, then next codes are not to be executed.

CodeMaster Noyon

JS if, else

Following the format for the good programming practice.

```
if(false){
    console.log('condition-1 is executed');
}else if(true){
    console.log('condition-2 is executed.');
}else if(true){
    console.log('conditon-3 is executed.');
}else{
    console.log('Else block is executed.')
};
```

// condition-2 is executed.

JS Switch

The *switch* statement is used to perform different actions based on different conditions. With *break* keyword.

```
let Name = 'Noyon’;

switch(Name){
    case 'Noyon':
        console.log('Noyon is here.');
        break;
    case 'Purna':
        console.log('Purna is here');
        break;
    case 'Rabbi':
        console.log('Rabbi is here');
        break;
};
// output:  Noyon is here.
```

JS Switch

The *switch* statement Without *break* keyword.

```
let Name = 'Noyon’;

switch(Name){
    case 'Noyon':
        console.log('Noyon is here.');
    case 'Purna':
        console.log('Purna is here');
    case 'Rabbi':
        console.log('Rabbi is here’);
};
// output:  Noyon is here.
            Purna is here.
            Rabbi is here.
```

CodeMaster Noyon

JS Switch

The *switch* statement with *default* case. When *switch* value does not match with any *case* then *default* case will be execute.

```
let Name = ‘Pritim';

switch(Name){
    case 'Noyon':
        console.log('Noyon is here.');
        break;
    case 'Purna':
        console.log('Purna is here');
        break;
    case 'Rabbi':
        console.log('Rabbi is here');
        break;
    default:
        console.log('This name is not exist.');
};
// output: This name is not exist.
```

CodeMaster Noyon

JS Loops

Loop can execute a block of code a number of times. There are mainly two types of loop.

- General loops
- Special loops

There are three general loops.

1. `for` loop
2. `While` loop
3. `do while` loop

There are two special loops. Special loops are used for objects and iterable object.

1. `for of` loop
2. `for in` loop

CodeMaster Noyon

JS for loop

The for statement creates a loop with three expressions.

1. Initialize state
2. Condition checking state
3. Updating state

```
for(intialization; condition-check; value-update){
   // block of code to be executed until condition is false.
}
```

```
for(let i = 0; i < 5; i++){
    console.log('Hello World !’)
};
```

```
// ‘Hello World !’ Will be printed 5 times.
```

JS while loop

The while statement creates a loop with three expressions.

1. Initialize state
2. Condition checking state
3. Updating state

```
let i = 0;  // initialization state
while(i < 5){  // condition checking state
    console.log('My name is Noyon.')
    i++;  // updating state
};
```

// ‘ My name is Noyon. ‘ To be executed until condition is false.

CodeMaster Noyon

JS do while loop

The `do while` loop is a variant of the `while` loop. This loop will execute the code block once, before checking the condition.

```
let i = 0;

do{
    console.log('Hello world.');
    i++;
}while(i < 5); // ‘ Hello world. ‘ to be executed 5 times cause condition is true.
```

```
let j = 0;
do{
    console.log('Hello world.');
    j++;
}while(j < 0); // ‘ Hello world. ‘ to be executed 1 time though condition is not true.
```

CodeMaster Noyon

JS for of loop

The for of loop is used for iterable objects Arrays and Strings.

```javascript
let Name = 'Noyon';
for(let letter of Name){
    console.log(letter);
};
```

Output:

```
N
o
y
o
n
```

```javascript
let fruits = ['Mango', 'Orange', 'Apple', 'Banana', 'Lichi'];

for(let item of fruits){
    console.log(item);
};
```

Output:

```
Mango
Orange
Apple
Banana
Lichi
```

JS for in loop

The for in statement loops through the properties of an object. We know details about object to the next. Each iteration for in loop return the object’s key.

```
const Person = {
    Name: 'Noyon',
    Age: 24,
    Gender: 'Male',
    Nationality: 'Bangladeshi’
}; // this is a object.

for(let i in Person){
    console.log(i);
};
```

Output:

```
Name
Age
Gender
Nationality
```

→ → In the next slide we learn how to find the key, value pair using for in loop.

CodeMaster Noyon

JS for in loop

Finding the kay, value pair from for in loop.

```
const Person = {
    Name: 'Noyon',
    Age: 24,
    Gender: 'Male',
    Nationality: 'Bangladeshi’
}; // this is a object.


for(let i in Person){
    console.log(`${i} : ${Person[i]}`);
};
```

<u>Output:</u>

```
Name : Noyon
Age : 24
Gender : Male
Nationality :
Bangladeshi
```

JS Sets

A JavaScript Set is a collection of unique values.

```
let numbers = new Set([2,4,6,2,7,9,3,4]);   // numbers is a set, here 2 & 4 have multiple time
console.log(numbers);   //   [ 2, 4, 6, 7, 9, 3 ]
```

Set methods:

add();   delete();   has();   value();

There is one property:

size

size

The size property returns the length of a Set.

```
console.log(numbers.size);    //   6
```

JS Sets

add(x)

The add(x) method adds new element to the Set. If you add equal elements, only the first will be saved.

```
let numbers = new Set([2,4,6,2,7,9,3,4]);
numbers.add(15);
numbers.add(10);
console.log(numbers);   //   [ 2, 4, 6, 7, 9, 3, 15, 10 ]
```

delete(x)

The delete(x) method removes an element from a Set.

```
let numbers = new Set([2,4,6,2,7,9,3,4]);
numbers.delete(2)
console.log(numbers); // [ 4, 6, 7, 9, 3 ]
```

CodeMaster Noyon

JS Sets

has(x)

The has(x) method returns true if a value exists in the Set.

```
let numbers = new Set([2,4,6,2,7,9,3,4]);
console.log(numbers.has(2));  // true
```

value()

The value() method returns a new iterator object containing all the values in a Set.

```
let numbers = new Set([2,4,6,2,7,9,3,4]);
numbers.value();  // returns a iterable object
```

JS Map

A Map holds key-value pairs where the keys can be any datatype.

```
let fruits = new Map([
    ['Mango', 500],
    ['Orange', 300],
    ['Apple', 400],
    ['Lichi', 200]
]);  //  fruits is a Map. Map defined & assigned
```

The Map has one property and some methods.

```
 size;        set();        get();        delete();        has();        entries();
```

size

The size property returns the length of a Map.

```
console.log(fruits.size);   // 4
```

JS Map

set(key, value)

The *set*() method adds the value for a key in a Map.

```
let fruits = new Map();    // Map define
fruits.set('Mango', 300);  // adds elements to the Map
fruits.set('Apple', 200);
```

get(key)

The *get()* method returns the value for a key in a Map.

```
console.log(fruits.get('Mango’));  // 300
```

delete(key)

The delete() method removes a Map element for a specific key.

```
fruits.delete('Mango’);  // removes Mango element
```

CodeMaster Noyon

JS Map

has(x)

The has(x) method returns true if a value exists in the Map.

```
let fruits = new Map([
    ['Mango', 300],
    ['Apple', 200],
    ['Orange', 400]
]);
console.log(fruits.has('Mango’));  // true
```

entries()

The entries() method returns a new iterator object containing all the values in a Map.

```
fruits.entries();  // returns a iterable object
```

CodeMaster Noyon

JS typeof & type conversion

typeof

typeof returns the data type of a JavaScript variable.

```
console.log(typeof 4);     // number
console.log(typeof 'Noyon’)    // string
```

Type conversion

Type conversion refers to the converts the data type of a value.

```
let number = 10;  // here type Number
console.log(typeof number); // number

String(number);  // type conversion. Here type is String
console.log(typeof number); // string
```

CodeMaster Noyon

JS Errors

try catch

The try statement defines a code block to run. The catch statement defines a code block to handle any error.

```
try {
    console.lo('Hello World.');
} catch (err) {
    console.log(err.message);
};
```

// console.lo is not a function

JavaScript has a built in error object that provides error information when an error occur. Object provides two properties.

- name
- Message

  // err.message

  // err.name

JS Errors

throw Error

throw statement allows you to create a custom error.

```
let number = 5;

try{
    if(number > 0){
        throw `It's a positive number`;
    }else if(number < 0){
        throw `It's a negative number`;
    }else{
        throw `It's 0.`
    }
}catch(err){
    console.log(err);
}; // It's a positive number
```

CodeMaster Noyon

JS this

In JavaScript, the this keyword refers to an object. Alone, this refers to the global object.

```
const Person = {
    firstName: 'Noyon',
    lastName: 'Sarker',
    fullName: function(){
        return this.firstName +" "+ this.lastName;
    }
}; // In the example, this refers to the person object.

console.log(Person.fullName());  // Noyon Sarker
```

JS Scope

Scope determines the accessibility of variables. JavaScript variables have 3 types of scope:

- Block scope
- Function scope
- Global scope

```
let a = 10;      var b = 10;      const c = 10;   // a, b, c are global scope
```

Before ES6, JavaScript variables had only Global scope & Function scope.

ES6 introduced two important new JavaScript keyword: `let` & `const.` These two keywords provide Block scope in JavaScript.

Block Scope:

```
{
    let a = 10;
};
// a can NOT be used here
```

```
{
    var a = 10;
};
  // a CAN be used here
```

Function Scope: Variables declared with `var,` `let` & `const` are quite similar when declared inside a function.

```
function a (){
    var x = 10;
};
// x can NOT be used here
```

```
function a (){
    let y = 10;
};
// y can NOT be used here
```

```
function a (){
    const z = 10;
};
// z can NOT be used here
```

CodeMaster Noyon

JS “use strict”

“use strict”; Defines that JavaScript code should be executed in “strict mode”.

Strict mode is declared by adding “use strict” to the beginning of a script or a function.

```
"use strict"
num = 5;
// there will be ERROR
```

```
"use strict"
var num = 5;
// No ERROR is here
```

```
"use strict"
function myFunc(){
    num = 10;
};
// No ERROR
```

```
"use strict"
function myFunc(){
    num = 10;
};
myFunc();// when you call the
function, you get the ERROR
```

```
"use strict"
function myFunc(){
    var num = 10;
};
myFunc(); // No ERROR
```

CodeMaster Noyon