

# IT-TALENTS Association





# Java Script

Java Script -ը ի սկզբանե ստեղծվել է վեբ կայքին «շունչ» հաղորդելու համար: Այս լեզվով գրված կոդը անվանում են script: Պլանավորվում էր, որ JS-ը կհամարվի Java-ի «փոքր եղբայր» -ը, բայց այն շուտ կատարելագործվեց և այժմ Java-ի հետ չունի ոչ մի ընդհանուր բան:

Ցանկացած լեզվով գրված կոդի իրականացման համար կա երկու տարբերակ՝ կոմպիլացիա և ինտերպրիտացիա:

1. Կոմպիլացիա. Եթե գրված կոդը հատուկ գործիքի (կոմպիլյատոր) միջոցով փոխակերպվում է ուրիշ լեզվով կոդի՝ մեքենայականի (օ, 1):
  2. Ինտերպրիտացիա. Եթե գրված կոդը փոխանցվում է հատուկ գործիքի (ինտերպրետատոր), որը տեղում իրականացնում է այն ինչպես որ կա:  
Մերօրյա ինտերպրետատորները JS կոդը իրականացնելուց առաջ փոխակերպում են այն մեքենայականի, կամ դրան մոտիկ մի բանի, օպտիմիզացնում են այն, իրականացնում և այդ ընթացքում աշխատում են այն օպտիմիզացնել:
- Այդ պատճառով է, որ JS-ը աշխատում է շատ արագ: Բոլոր հիմնական browser-ներում տեղադրված է JS ինտերպրետատոր, ինչի հետևանքով էլ հնարավոր է JS կոդը իրականացնել վեբ էջում:

Բացի վեր էջերում կիրառելուց JS-ն ունի մեծ հնարավորություններ, այն լիարժեք ծրագրավորման լեզու է, JS-ով կարող եք ծրագրավորել ինչ ուզեք՝ սառնարան, լվացքի մեքենա և այլն: Միակ անհրաժեշտությունն այն է, որ վերոնշյալ սարքերում լինի տեղադրված JS ինտերպրետատոր: Վեբում JS-ը յուրահատուկ է առնվազն ստորև թվարկված իր երեք հատկություններով՝

1. Լիարժեք ինտեգրվում է HTML/CSS-ի հետ:
2. Շատ պարզ եղանակներով կարելի է լուծել մեր առջև ծառացած նույնիսկ բարդագույն խնդիրները:
3. Համատեղելի է բոլոր տարածված browser-ների հետ:

Script -ը կարելի է կապել HTML-ի հետ հետևյալ երեք տարբերակով՝ JS-ով գրված կոդը կարելի է տեղադրել HTML-ի ցանկացած հատվածում script թեգի միջոցով:

`<script>...</script>` (internal տարբերակ):

JS կոդ կարելի է գրել HTML-ի թեգի մեջ (inline տարբերակ):

Կարելի է ունենալ առանձին JS ֆայլ, որը արդեն `<script src=""></script>` ձևով հասանելի ենք դարձնում HTML-ում, փակվող BODY թեգի հենց վերևում, src-ում գրվում է JS ֆայլի ճանապարհը հաշվարկած HTML ֆայլի տիրույթից:

Ինտերապրետատորը աշխատելու է սինխրոն, այսինքն կոդը սկսում է կարդալ վերևից և `script` - ի հանդիպելիս դադարեցնելու է `HTML`-ի կառուցումը `browser`-ում, կարդալու և իրագործելու է `script`-ում գրված կոդը, այնուհետև շարունակելու է կառուցել `HTML`-ը `browser`-ում:

Անհրաժեշտության դեպքում մենք կարող ենք ստիպել ինտերապրետատորին աշխատել ասինխրոն, այսինքն՝ և՛ կառուցել `HTML`-ը `browser`-ում, և՛ կարդալ ենթադրենք `HTML`-ին էքստերնալ տարբերակով միացված `script`-երը միաժամանակ: Ասինխրոն աշխատանք կարելի է կազմակերպել `script` թեգի `asinc` կամ `defer` գրելով:

`Script` թեգում `asinc` գրելով մենք պարտադրում ենք ինտերապրետատորին `HTML`-ին միացված `script`-երը «կարդալ» միաժամանակ, բայց այս դեպքում արդեն պահպանվելու է միացված `script`-երի իրականացման հերթականությունը, իրականացվելու են նրանք ըստ `HTML`-ին միացված լինելու հերթականության՝ վերևից ներքև:

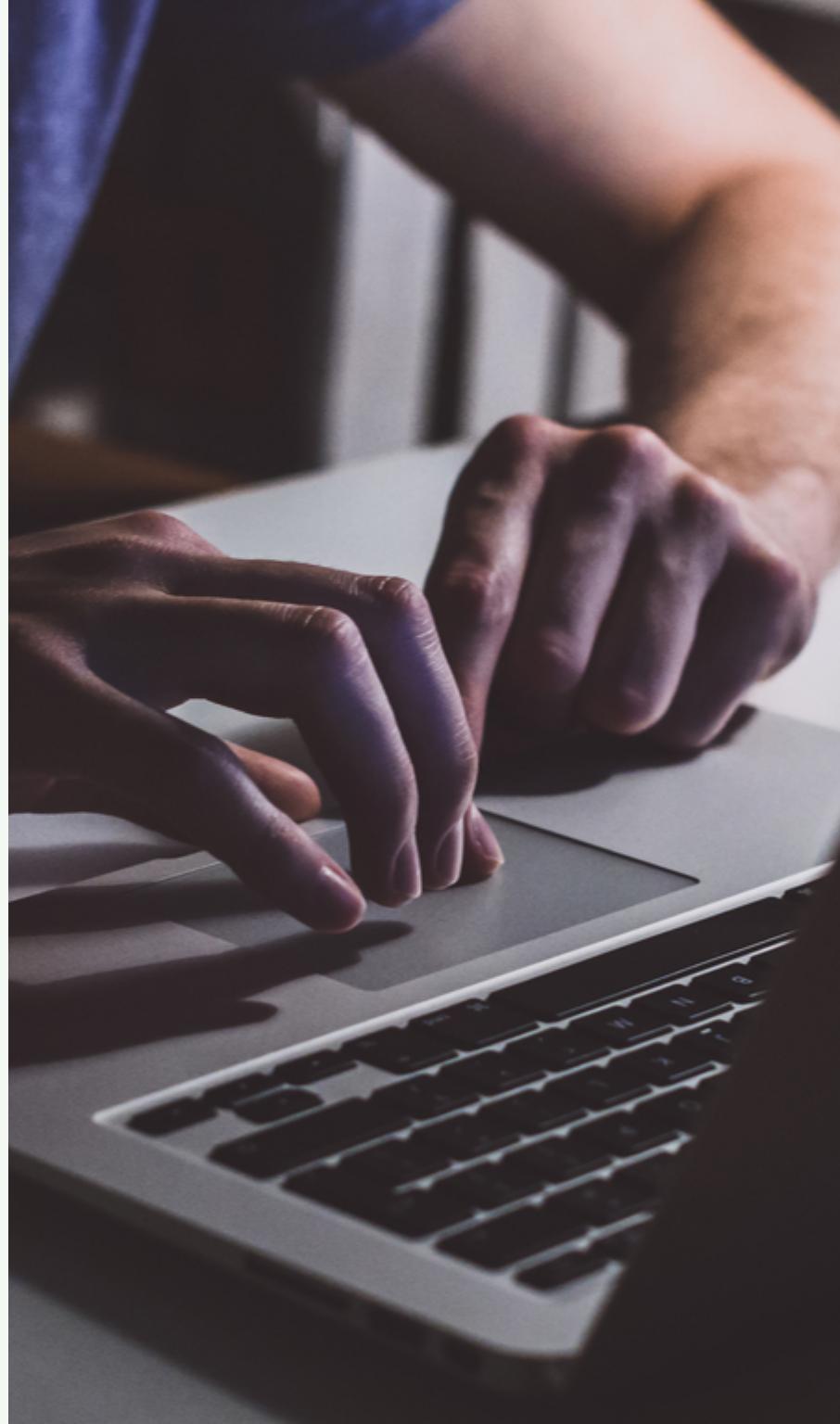
Այժմ տեսնենք թե ինչ տեսք ունի JS կոդը.

Ինչպես նշեցինք ներածության մեջ կարելի է JS գրել թեզի  
մեջ:

```
<!doctype html>
<html lang="en">
<head>
    <title>lesson-1</title>
</head>
<body>
<span onclick="alert('Hello')">Click Me!!</span>
</body>
</html>
```

span-ի վրա click անելուց աշխատեց  
JS-ում ներկառուցված բազմաթիվ  
ֆունկցիաներից մեկը `alert()`-ը. այն  
ուղղակի browser-ում բացեց նոր  
պատուհան, որում տպեց այն ինչ  
հանձնարարել էինք՝ Hello.

```
<!doctype html>
<html lang="en">
<head>
    <title>lesson-1</title>
</head>
<body>
<script>
    alert('Hello')
</script>
</body>
</html>
```



Պրոյեկտների մեջ հիմնականում օգտագործվում է էքստերնալ տարբերակը, քանի որ JS կոդը լինում է ծավալուն, առանձին JS ֆայլում JS կոդ գրելու օրինակ.

```
<!doctype html>
<html lang="en">
<head>
    <title>lesson-1</title>
</head>
<body>
<script src='lesson-1.js'></script>
</body>
</html>
```

Իսկ lesson-1.js ֆայլում.

```
'use strict';
alert('Hello');
```

JS ֆայլի սկզբում 'use strict' գրելով ակտիվացնում ենք խիստ ռեժիմ, որպեսզի ինտերպրետատորը կարդա և իրագործի կողը որպես JS-ի վերջին տարբերակի կող: Հին և նոր տարբերակների մեջ տարբերություններ կան, օրինակ փոփոխական հայտարարելու ձևի և այլն: Գրագետ գրելածն է ամեն առանձին հրաման գրել նոր տողից, իսկ նախորդից հետո դնել ;// կետ և ստորակետ:

```
alert('Hello world 1 !!!');// Այս ձևով (երկու թեք գիծ) կարող ենք
```

```
տողային մեկնաբանություն գրել: alert('Hello world 2 !!!');
```

```
/* comment -----  
-----comment */
```

Եթե alert-ի մեջ գրենք թվաբանական գործողություն, ապա alert-ի պատուհանում կտպվի այդ գործողության արժեքը:

Անցնենք փոփոխականներին, որպեսզի լավ պատկերացնենք թե ինչ է փոփոխականը, փորձենք հասկանալ դրա դերը առօրյայից վերցրած օրինակով. Պատկերացնենք ունենք արկղեր, դրանցից մեկի մեջ լցնենք ինձոր, մյուսի մեջ՝ տանձ և այլն: Ամեն արկղի վրա համապատասխանաբար գրենք տանձ, ինձոր: Հետո երբ ինձոր պետք լինի բնականաբար կինտրենք այն արկղում որի վրա գրված է ինձոր: Փաստորեն մենք արկղին տվեցինք անուն, պահեցինք դրա մեջ ինչ-որ բան և պետք եղած ժամանակ դիմեցինք հենց այն արկղին, որին սկզբից անուն էինք տվել:

Մոտավոր նույն պատկերն էլ ունենում ենք JS-ում: Հայտարարում ենք փոփոխական և տալիս ենք նրան անուն `var message;`, պահում ենք նրա մեջ ինֆորմացիա. `message = "New message";`

Հարկ եղած դեպքում դիմում ենք նորից այդ փոփոխականին արդեն առանց հայտարարելու, միայն անունը տալով, ստանում ենք նրա պարունակությունը: `alert(message);`

Հայտարարել փոփոխական և արժեք վերագրել կարելի է նաև մեկ տողով՝ `var message = 'New message';`

Հայտարարելուց և արժեք վերագրելուց հետո կարող ենք այդ նույն փոփոխականին նոր արժեք տալ (փոխել արժեքը):

`message = 'success';`

Նոր արժեք վերագրելուց հետո, եթե փորձենք տպել այդ փոփոխականի արժեքը կստանանք վերջին վերագրված արժեքը, քանի որ ինտերպրետատորը կողը կարդում և իրականացնում է վերևից ներքեւ, վերջին կատարված փոփոխության արդյունքն էլ կմնա որպես փոփոխականի արժեք:

Մեկ տողով կարելի է հայտարարել մի քանի փոփոխական և նույնիսկ արժեք վերագրել՝

```
var brand = "Nissan", model = "X-Trail", year = 2015;
```

Այս հայտարարումից հետո կարող ենք արդեն մեզ հայտնի alert-ի օգնությամբ տպել նրանց ինչպես առանձին-առանձին՝

```
alert(brand);
alert(model);
alert(year);
```

Այնպես էլ խմբակային՝

```
alert(brand + " " + model + " " + year);
```

Այս դեպքում փոփոխականների արանքում կցագրել ենք բացատանիշ:

Փոփոխականների արժեքը կարող ենք կրկնօրինակել՝

```
var a = 15;
var b = 25;
a = b;
alert(a);
```

JS-ում փոփոխականներին անուն տալիս պետք է հիշել հետևյալը՝ այն չպետք է սկսվի թվով, չպետք է պարունակի "-" (մեջտեղի գիծ), իսկ դոլարի նշան "\$" կամ տակի գիծ "\_" կարող է պարունակել, նույնիսկ կարող է սկսվել դրանցով:

Պետք է իմանալ, որ

```
var name = "js";
var Name = 'JS';
```

JS-ն ունի **reserve** արված բառեր, որոնք չի կարելի օգտագործել որպես ինֆորմացիայի պահպանման եղանակ.

```
return
class
export
var...
```

Բացի փոփոխականներից JS-ն ունի ինֆորմացիայի պահպանման ևս մեկ տարբերակ՝ հաստատունները:

ES5-ում հաստատունները ձևակերպվում են պայմանավորվածությունների հիման վրա: Կա պայմանավորվածություն, որ երբ փոփոխականը գրվում է մեծատառերով, ապա համարում ենք, որ այն հաստատուն է: Նրա արժեքը չի կարելի փոխել: ES6-ն արդեն ունի հաստատուն հայտարարելու ձև՝ `const`, որի արժեքը չենք կարող փոխել:

# Փոփոխականի տիպերը JS-ում

## Number.

Եթե հայտարենք փոփոխական ու վերագրենք թվային արժեք՝ `var a = 123;` և փորձենք տպել այս փոփոխականի `type`-ը `alert(typeof a); // typeof()` մեթոդը վերադարձնում է փոփոխականի `type`-ը: Այս դեպքում մեր `alert`-ը կտպի `number`: Այսպիսով ծանոթացանք JS-ի փոփոխականի տիպերից մեկի հետ՝ `number`: `Number` տիպը ունի ենթատիպեր.

Օրինակ՝

```
var varType = 1 / 0;  
alert(varType); //infinity  
alert(typeof varType); //number
```

Այս ենթատիպը կոչվում է անսահմանություն:

```
var varType = "string" * 2;  
alert(varType); //NaN(Not a Number)  
alert(typeof varType); //number
```

Սա էլ մյուս ենթատիպը՝ `NaN`(`Not a Number`).

## String.

Փոփոխականի հաջորդ տիպը, որը կներկայացնենք `string`-ն է.

```
var str = "Hello JS";  
document.write(str + '<br>');//document.write()-ը տպում է document-ում(html-ում)  
alert(typeof str); //string(text)
```

## Boolean (bool) // TRUE FALSE

```
var a = 5;
var b = 15;
var varType = a === b;
alert(a === b); //false, որովհետև a-ն հավասար չէ b;
alert(typeof varType); // boolean (bool) // TRUE FALSE

var varType = a < b;
alert(varType); // true, որովհետև 5 < 15
alert(typeof varType); // boolean (bool) // TRUE FALSE
```

## Null

```
var name = null;
alert(name); //null (դատարկություն, կամ հայտնի չէ)
alert(typeof name); //object (օֆիցիալ ընդունված bug)
```

Undefined

```
var a;  
alert(a); // undefined(հայտարարված է բայց արժեքը չունի)  
alert(typeof a); // undefined
```

Object { }

```
var user = {  
    firstName: 'Steve',  
    lastName: 'Jobs',  
    age: 30  
};  
alert(typeof user); // object  
alert(user.firstName); // Steve  
alert(user.lastName); // Jobs  
alert(user.age); // 30
```

Ամփոփելով կարող ենք ֆիքսել, որ JS-ում կա փոփոխականի վեց տիպ՝

1. number
2. string
3. boolean
4. null
5. undefined
6. object

Այժմ ծանոթանանք JS-ի հիմնական օպերատորների հետ:

Նախ նշենք, որ օպերատորները կարող են լինել երկու տեսակ՝ unary և binary

```
var a = 5;
var b = -a;
այստեղ "-" -unary operator է համարվում, իսկ եթե Լինի՝
var a = 5;
var b = 7;
var c = a + b;
ապա "+" կհամարվի binary operator.
```

unary "+"-ը string-ի type-ը սարքում է number.

binary "+"-ի օպերատորներից ցանկացածը եթե եղավ string, ապա կատարվելու է կցագրում (կոնկատանացիա).

```
var x = 5 + '5';
alert(x); // 55
```

Մյուս բոլոր օպերատորները աշխատում են թվերի հետ:

"%" օպերատորը բաժանում և վերադարձնում է մնացորդը՝

```
var a = 5 % 2;
var b = 8 % 3;
var c = 6 % 3;
alert(a); // 1
alert(b); // 2
alert(c); // 0
```

```
var i = 2;  
i = i + 1;  
alert(i); // 3  
'i = i + 1' կարող ենք գրել ավելի կարճ՝ i += 1:  
սա նույնական է i-ին վերագրում է i+1, և կարող ենք գրել ավելի կարճ՝  
++i (prefix inkrement) կամ i++(postfix inkrement); // ավելացնում է i-ի արժեքը 1-ով  
--i (prefix dekrement) կամ i--(postfix dekrement); // նվազեցնում է i-ի արժեքը 1-ով
```

Բայց postfix-ի և prefix-ի միջև կա տարբերություն

Postfix inkrement-ը ենթադրում է i-ի արժեքի ավելացում 1-ով, սակայն  
այն prioritet-ով (առաջնահերթություն) ավելի ցածր է քան մյուս թվաբանական  
գործողությունները, այսինքն մինչև i-ի արժեքը 1-ով ավելացնելը կատարվելու են  
մյուս թվաբանական գործողությունները (եթե առկա են այդպիսիք), հետո նոր  
փոխվելու է i-ի արժեքը, իսկ prefix-ի առաջնայնությունը բարձր է, այսպիսով՝ prefix  
inkrement-ը (կամ dekrement-ը) աշխատելու է հենց սկզբից, հետո նոր մյուս  
թվաբանական գործողությունները:

```
var i = 1;  
var a = ++i;// i = 2,  a = 2  
alert(a); // 2
```

```
var i = 1;  
var a = i++; // a = 1, i = 2  
alert(a); // 1  
alert(i); // 2
```

Ուշադրություն՝

inkrement և dekrement կարելի է կիրառել միայն փոփոխականների վրա (բնականաբար number type ունեցող), թվերի վրա կիրառելիս կստանանք error.

Համեմատության և տրամաբանական օպերատորներ

```
alert(2 > 1); //true  
alert(2 < 1); //false  
alert(2 >= 1); //true  
alert(2 <= 1); //false  
alert(2 == 1); //false  
alert(2 != 1); //true  
alert(1 == 1); //true  
alert(2 == '2'); //true
```

Կարող ենք նաև հենց փոփոխականին վերագրել տրամաբանական արժեք

```
var a = true;
var b = 3 > 4;
alert(b); //false
alert(a == b); //false
```

Օգտատիրոջից ստացված տվյալների համեմատում

```
alert("2" > "14"); //true //քանի որ '2'symbol-ի սունձը ավելի մեծ է քան '1'symbol-ինը  
alert(+ "2" > + "14"); //false //քանի որ stringner-ին սկզբից ստուգական փուլում առաջանակ է կատարվում և համեմատվում են արդեն որպես  
number  
alert("2" > +"14"); //false //քանի որ string-ը թվային արժեքի հետ համեմատելիս վերածվում է թվայինի ու հետո  
նոր համեմատվում:
```

String - ների համեմատում

```
var a = "A";
var b = "a";
alert(a > b); // "A" > "a" == false
alert(a < b); // "A" < "a" == true
(տես՝ աղյուսակ)
```

```
var unicodStr1 = "ABC";
var number1 = unicodStr1.charCodeAt(0); //65
var number2 = unicodStr1.charCodeAt(1); //66
var number3 = unicodStr1.charCodeAt(2); //67
document.write(number1 + ' ' + number2 + ' ' + number3); //65 66 67
```

Stringi վրա charCodeAt() ֆունկցիան կանչելիս այն վերադարձնում է իրեն որպես արգումենտ փոխանցված ինդեքսով symbol-ի unicode-ը.

```
var unicodStr1 = "ABC";
var unicodStr2 = "abc";
alert(unicodStr1 > unicodStr2); // false //քանի որ "A" < "a"
alert("Hello" > "Hell"); //true // օ-ն մեծ է քան symbol-ի բացակայությունը
alert("Hello" < "Hell"); //false
alert("Hello" == "Hell"); //false
```

code-ից -> symbole

```
var uniStr = String.fromCharCode(65,66,67);
document.write(uniStr); //ABC
```

Տարբեր type-երի փոփոխականներ թվայինի հետ համեմատելիս տեղի է ունենում փոփոխականների փոխակերպում թվայինի և համեմատվում են նրանց թվային արժեքները:

```
alert('2' > 1); //true
alert('01' == 1); //true
alert(false == 0); //true
alert(true == 1); //true
```

Խիստ հավասարման նշան "==="

```
alert(0 === false); // true
alert('' === false); // true
alert(0 ==== false); // false
alert('' ==== false); // false
alert("1" === 1); // false
alert(1 ==== 1); // true
```

Եթե դրվում է խիստ հավասարման նշան , որպեսզի կոդը վերադարձնի `true`' բացի նրանից, որ համեմատվողները պետք է ունենան նույն արժեքը, անհրաժեշտ է նաև, որ նրանք ունենան միևնույն `type`-ը:

Null-ն ու `undefined`-ը հավասար են իրար և հավասար չեն ուրիշ ոչնչի:

```
alert(null === undefined); // true
alert(null ==== undefined); // false
alert(+null); // 0
alert(+undefined); // NaN
```

Փոխակերպում string-ից թվայինի.

```
var str = "2017";
alert(typeof str);
var newStr = Number(str);
alert(typeof newStr); //number
```

Ինչպես արդեն նկատել ենք alert()-ը էկրանին ցույց է տալիս հաղորդագրություն (նոր պատուհանով) և կանգնեցնում է script-ը մինչև user-ը չսեղմի "ok", իսկ prompt() function-ը ընդունում է երկու արգումենտ. առաջինը՝ հարց, վերնագրի տեսքով և երկրորդը մուտքագրելու տողում տեղադրվող default արժեք:

```
var result = prompt('How Old Are You ?', '');
document.write(result);
var age = prompt('How Old Are You ?', '');
var name = prompt('What Is Your Name ?', '');
document.write(age + ' <br> ' + name);
```

Եթե prompt-ի պատուհանում չլրացնենք ոչինչ, ապակվերադարձնի դատարկ string, եթե սեղմենք cancel, Esc կամ x-ով փակենք, ապա կվերադարձնի null:

Prompt-ին ճիշտ է միշտ տալ երկրորդ արգումենտ, նույնիսկ եթե չենք լրացնելու ոչինչ (գոնե դատարկ չակերտ պետք է թողնել):

## Confirm()

Confirm()- ը վերադարձնում է true կամ false

```
var answer = confirm('Are You agree?');
document.write(answer);
```

## Պայմանի օպերատոր

Երբեմն անհրաժեշտ է լինում, ինչ-որ պայմանից կախված տարբեր գործողություններ կատարել, դրա համար օգտագործվում է if operator-ը:

```
var answer = confirm("?? ??????? ?");
if(answer) {
    result = 'Yes';
}
document.write(result);
```

if-ից հետո գրված () փակագծերում գրվում է պայմանը, որը true լինելու դեպքում կատարվում է if-ի մարմնում (ձևավոր փակագծերում՝ {}) գրված գործողությունը:

If-ից հետո կարելի է գործողությունը հանձնարարել նաև պայմանի `false` լինելու դեպքում՝

```
var answer = confirm("?? ??????? ?");
if(answer) {
    result = 'Yes';
} else {
    result = 'Cancel';
}
document.write(result);
```

Այս դեպքում `else`-ը կվերաբերվի `answer`-ի `false` լինելուն:

Վերևի գրված կոդը կարելի է կարդալ այսպես, եթե `answer` փոփոխականը `true` է՝ `result = 'Yes'`, մնացած բոլոր դեպքերում `result = 'Cancel'`:

```
var answer = prompt("Your age", 100);
if (answer < 18) {
    alert("You are not adult");
} else {
    alert("You are adult");
}
```



# JavaScript Ternary Operator

## Ternary operator

Վերևում գրված կոդը կարելի է գրել նաև այսպես (ternary):

```
(answer < 18) ? alert("You are not adult") : alert("You are adult");
```

## OR operator

Այժմ ծանոթանանք կամ օպերատորի հետ ('||'):

```
alert(true || true);//true  
alert(false || true);//true  
alert(true || false);//true  
alert(false || false);//false
```

Այն վերադառնում է true, եթե արժեքներից գոնե մեկը true է և վերադառնում է false, եթե արժեքներից ոչ մեկը true չէ:

Եթե արժեքների type-ը boolean չէ, ապա վերափոխվում են boolean-ի և աշխատում է OR-ը

```
if (1 || 0) {  
    alert('OK');//alert kani, qani vor 1-y true e  
}
```

Սովորաբար կամ օպերատորը օգտագործվում է if-ի հետ:

```
var hour = 9;  
if (hour < 10 || hour > 18) {  
    alert('closed');  
}
```

Կարող ենք փոխանցել նաև 2-ից շատ  
պայման

```
var hour = 12;
var isWeekend = true;
if (hour < 10 || hour > 18 || isWeekend) {
    alert( 'Closed' );
}
```

Կամ օպերատորի համար կարող ենք պատկերացնել այսպես, քանի որ  
արժեքներից գոնե մեկի true լինելու դեպքում այն վերադարձնում է true,  
հետևաբար առաջին true-ն գտնելուց հետո մյուս արժեքների վրայով անցնել պետք  
չէ իրեն, հենց գտավ առաջին true-ն մնացած OR-երի մեջ գրված  
արտահայտությունները չեն կատարվում:

```
var a;
true || (a = 5);
alert(a); //undefined, քանի որ առաջին արժեքը true է,
կամ օպերատորի մյուս կողմի արտահայտությունը անտեսվեց:
var a;
false || (a = 5);
alert(a); //5
```

Ամփոփելով ասենք, որ կամ օպերատորը վերադարձնում է առաջին հանդիպած true-ն:

```
alert( undefined || '' || 1 || 0 );//1
```

Եթե true չի գտնվում, ապա վերադարձնում է վերջին արժեքը, բնականաբար false արժեքը:

```
alert( undefined || '' || false || 0 );//0
```

## && (AND)

And օպերատորը վերադարձնում է true, եթե and-ով իրար հետ կապած բոլոր արտահայտությունների boolean արժեքները true են, այլապես վերադարձնում է false:

```
alert(true && true); //true  
alert(false && true); //false  
alert(true && false); //false  
alert(false && false); //false
```

```
var hour = 12;  
var minute = 30;  
if (hour === 12 && minute === 30) {  
    alert('Time 12:30');  
}
```

Alert-ը կաշխատի, քանի որ if-ի պայմանում գրված տրամաբանությունը true է:

Եթե &&-ի aperand-ները boolean type-ի չեն, ապա նրանք վերափոխվում են boolean-ի և հետո նոր &&-ը վերադարձնում է արժեք:

```
if (1 && 0) {  
    alert('?? ?????????, ?.?. ?????? ?????');  
}
```

And օպերատորը վերադարձնում է առաջին հանդիպած false-ը, իսկ եթե բոլոր aperand-ները true են , ապա այն վերադարձնում է վերջին արժեքը:

```
alert(1 && 2 && null && 3); //null  
alert(1 && 2 && 3); //3
```

Առաջնայնության տեսանկյունից && առավելություն ունի ||-ի նկատմամբ:

```
alert(5 || 1 && 0); //5
```

Սկզբում աշխատելու է կոդի 1 && 0 հատվածը, հետո նոր 5 || 0 (1 && 0 ից ստացված արժեք):

```
var num = 10;
if (num) {
    alert(num);
} else {
    alert("Variable is Empty");
}
```

## Կամ՝

```
if (!num) {
    alert("Variable is Empty");
} else {
    alert(num);
}
var str = '';
if (!str) {
    alert(str);
} else {
    alert("Variable is Empty");
}
```

# Switch

```
switch(արտահայտություն) {
  case x:
    // գործողություն
    break;
  case y:
    // գործողություն
    break;
  default:
    // գործողություն
}
```

switch-case-ի միջոցով կարող ենք ինչ-որ արտահայտության կոնկրետ արժեքների դեպքում կատարել որոշակի գործողություններ:

```
var today = new Date().getDay();

switch (today) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

Ամեն case-ից հետո break-ը պետք է, որպեսզի case-ի մարմին մտնելուց հետո switch-ից դուրս գցի, մյուս case-ի մարմնի գործողությունը չկատարվի: Եթե ուզում ենք case-երի դեպքում նույն գործողությունը կատարվի գրում ենք այսպես՝

```
var a = +prompt(' 0 < Enter Even Number <= 5','');
switch (a) {
    case 2:
    case 4:
        alert('true!');
        break;
    case 1:
    case 3:
    case 5:
        alert('you entered odd number!');
        break;
    default:
        alert('you entered number is bigger then 5, or smaller then 0');
}
```

Եթե ինտերպրետատորը չի մտնում ոչ մի case-ի մեջ, ապա մտնում է default - ի մեջ ու կատարում այնտեղ գրված գործողությունը:

## Loop/ for/

```
for (cikl-ի start-ը; պայման; cikl-ի քայլը) {  
    // cikl-ի մարմին  
}  
  
for (var i = 0; i < 3; i++) {  
    alert(i);  
}
```

cikl-ի մարմնում գրված գործողությունը կատարվում է այյնքան ժամանակ քանի դեռ cikl-ի պայմանում գրված է true արժեք ունեցող արտահայտություն: Այս օրինակում for-ի կլոր փակագծերի մեջ վերջում գրված է i++ , դա նշանակում է ամեն cikl-ի վերջում i-ի արժեքը մեկով ավելացնել, այսինքն բացի այն, որ պայմանի ճիշտ լինելու դեպքում կատարվելու է մարմնում գրված գործողությունը, կատարվելու է նաև cikl-ին որպես քայլ փոխանցված գործողությունը:

```
var name = "JavaScript";  
alert(name.length);  
alert(name[4]);  
for (var i = 0; i < name.length; i++) {  
    document.write(name[i] + "<br>");  
}
```

## While

while cikl-ը ունի հետևյալ տեսքը՝

```
while (պայման) {  
    // cikl-ի մարմին//կոդ  
}  
  
var i = 0;  
while (i < 3) {  
    alert(i); // 0, 1, 2  
    i++; // 1, 2, 3  
}  
alert(i); // 3
```

Եթե cikl-ի պայմանում գրենք մշտապես true պայման, ապա կստացվի անվերջ cikl՝

```
while (true) {  
    alert(12345);  
}
```

Օրինակ dekrement-ով՝

```
var i = 3;  
while (i) {  
    alert(i);  
    i--;  
}  
alert(i);
```

## Do while

```
var i = 0;
do {
    alert(i); // 0, 1, 2
    i++; // 1, 2, 3
} while (i < 3);
```

Do while-ով գրված cikl-ը սկզբից (առանց պայման ստուգելու) մեկ անգամ կատարում է cikl-ի մարմնում գրված գործողությունը հետո նոր ստուգում պայմանը:

## Break

Break-ը դադարեցնում է cikl-ը, դուրս է գալիս cikl-ից

```
var text = '';
for (i = 0; i < 10; i++) {
    if (i === 3) {
break;
}
    text += "The number is " + i + "<br>";
}
alert(text);
```

## Continue

```
var text = '';
for (i = 0; i < 10; i++) {
    if (i === 3) {
continue;
}
    text += "The number is " + i + "<br>";
}
alert(text);
```

# Function

Պրոյեկտների մեջ, շատ հաճախ, մեզ պետք է գալիս կեդի մի փոքր հատվածն օգտագործել մի քանի անգամ, տարբեր տեղերում, տարբեր իրավիճակներում: Կոդի այդ մի փոքր հատվածը ամեն անգամ copy paste չանելու և նմանատիպ այլ անհարմարություններից խուսափելու համար ստեղծվել է function-ը:

Այն իրենից ներկայացնում է գործողությունների համախումբ, որը կիրականանա կոդի այն հատվածում, որտեղ որ կկանչենք հայտարարված function-ը:

```
function showMessage() {  
    alert( 'Hello World' );  
}
```

ShowMessage-ը մեր հայտարարած function-ի անունն է: Թե ինչ կլինի մեր հայտարարած function-ի անունը, կվորոշենք մենք:

Ուշադրություն դարձնենք այն փաստին, որ չնայած function-ի մարմնում գրված է alert(), այնուամենայնիվ alert տեղի չի ունենում, ինչու՞, քանի որ function-ի մարմնում գրված կոդը չի իրականացվելու քանի դեռ այն կանչված չի:

Պատկերացնենք, որ function-ը կանչել նշանակում է իր անունը տալ, օրինակ նախորդ օրինակում գրված function-ը կկանչենք այսպես՝

```
showMessage(); // Hello World
```

Function-ի մարմնում հայտարարված փոփոխականը հայտարարվում է function-ի local տիրույթում և այն տեսանելի չէ դրսից(այս դեպքում գլոբալ տիրույթից):

```
function newMessage() {  
    var mess = "Hello World !!!";  
}  
newMessage();  
alert(mess); // chhitararvac popoxakan  
function newMessage() {  
    var mess = "Hello World !!!";  
    alert(mess);  
}  
newMessage(); //Hello World !!!
```

If/else, switch, for, while, do..while - այս պայմանի օպերատորներում և cikl - ներում մենք փոփոխականի տեսանելիության տիրույթից խնդիր չունենք:

```
function count() {  
    for (var i = 0; i < 3; i++) {  
        var j = i * 2;  
    }  
    alert(i);  
    alert(j);  
}  
count();
```

Այսօրինակում i, j տեսանելի են cikl-ից դուրս, բայց ոչ function-ից:

Function - ը կարող է դիմել դրսի տիրույթի փոփոխականի.

```
var userAge = 25;
function userInfo() {
    var message = 'You are ' + userAge + ' years old';
    alert(message);
}
userInfo();
```

Function-ի մարմնում կարելի է փոփոխել նույնիսկ դրսի տիրույթում հայտարարված փոփոխականի արժեքը՝

```
var userName = "Jon";
function renameUser() {
    userName = "Michael";
    alert("Your name is " + userName);
}
renameUser();
alert(userName);
```

## Argument

```
function letterCount(name, word) {
    var letCount = word.length;
    document.write("Dear <b>" + name + "</b> in the word " + word + " - <b>" + letCount + "</b> symbols.");
}
letterCount("Jon", "Javascript");
```

Function-ը կանչելուց կարող ենք որպես արգումենտ արժեք փոխանցել նաև  
հետևյալ կերպ՝

```
function letterCount(name, word) {  
    var letCount = word.length;  
    document.write("Dear <b>" + name + "</b> in the word " + word + " - <b>" + letCount + "</b> symbols."  
);  
}  
var userName = "Armen";  
var userWord = "Bootstrap";  
letterCount(userName, userWord);
```

Եթե function-ը հայտարարել ենք այնպես, որ կանչելիս պահանջում է argument,  
ապա argument չփոխանցելու դեպքում կփոխանցվի undefined: Հետևաբար  
գրագետ կլինի հետևյալ validation-ը՝

```
function letterCount(name, word) {  
    if (word !== undefined) {  
        var letCount = word.length;  
    } else {  
        word = "Not found";  
        letCount = 0;  
    }  
    document.write("Dear <b>" + name + "</b> in the word " + word + " - <b>" + letCount + "</b> symbols."  
);  
}  
letterCount("Jon");
```

## Արժեքի վերադարձ/return

Շատ հաճախ մեզ պետք է լինելու ոչ թե տպել function-ի միջոցով, այլ վերադարձած արժեքի հետ այլ գործողություններ կատարել, դրա համար կարող ենք օգտագործել return-ը: Function-ում return անելու դեպքում այն կանչելուց հետո մենք ենք որոշում տպել այդ արժեքը, թե այլ գործողություն կատարել հետո տպել.

```
function calc(num1,num2) {
    return num1 + num2;
}

document.write(calc(4,5));

kam`


function calc(num1,num2) {
    return num1 + num2;
}
var answer = 5 * calc(4,5);
document.write(answer);
```

Function-ը կարելի է կրկնօրինակել (copy)՝

```
function message() {  
    alert("Success");  
}  
var newFunc = message;  
newFunc();
```

## Function Declaration

Մինչ այժմ մենք օգտագործել ենք declaration տիպի functionner-ը, այն ենթադրում է function-ի հայտարարում հետևյալ ձևով՝

```
function calc(a, b) {  
    return a + b;  
}
```

Ուշադրություն.

Այս տիպի function-ները կարող են կանչվել իրենց հայտարարումից վերև:

```
alert(calc(5, 5));  
  
function calc(a, b) {  
    return a + b;  
}
```

Աշխատող կոդ:

Ինտերպրետատորը declaration տիպի function-ներին ծանոթանում է սկզբից, այդ է պատճառը, որ այս տիպի function-ը կարող է կանչվել կոդի ցանկացած հատվածում, անկախ հայտարարման տողի կոորդինանտից:

## Function Expression

```
var calc = function (a, b) {  
    return a + b;  
};  
alert(calc(5, 5));
```

Այս ձևով հայտարարված function-ները կոչվում են expression: Փաստորեն expression տիպի function-ը վերագրվում է փոփոխականի, հենց դա է պատճառը, որ այս տիպի function-ը չի կարող կանչվել իր հայտարարման տողից վերև.

```
alert(calc(5, 5));  
var calc = function (a, b) {  
    return a + b;  
};
```

Inspect-ում կստանանք 'calc is not a function':

Function-ի ևս մեկ տեսակ anonym.

```
function check(question, yes, no) {  
    if (confirm(question)) {  
        yes();  
    } else {  
        no();  
    }  
}
```

# Օգտագործման եղանակներ՝

1.

```
function showOk() {
    alert( "You agree!!" );
}
function showCancel() {
    alert( "Canceled" );
}
check("Are You agree?", showOk, showCancel);
```

2.

```
check(
    "Are You agree?",
    function() { alert("You agree!!"); },
    function() { alert("Canceled"); }
);
```

Հազվադեպ օգտագործվող

```
var myFunc = new Function('a, b', 'return a * b');
alert(myFunc(5,5));//25
```

Աշխատում ենք թվերի հետ:

Ենթադրենք ինչ-որ տեղից ստացել ենք '12px' արժեք և մեզ պետք է այս արժեքի դիմացի թվային մասը, այստեղ unary + նշանը մեզ չի կարող օգնել:

```
alert(+ "12px"); //NaN
```

Նման խնդրի հանդիպելիս ճիշտ է օգտվել parseInt-ից

```
var intNum = parseInt('12px');
document.write(intNum); //12
```

Այն վերադառնում է NaN, եթե իր արգումենտի արժեքի հենց սկզբից ոչ թվային արժեք է, հակառակ դեպքում վերադառնում է այն թվային արժեքը, որը գրված է արգումենտի արժեքի սկզբից սկսած (ամբողջ թիվ):

```
var floatNum = parseFloat('12.3.4');
document.write(floatNum); //12.3
```

ParseFloat-ը աշխատում է լրիվ նույն տրամաբանությամբ, մի տարբերությամբ, այն կարող է վերադառնել նաև մնացորդով թվերը:

Թվային արժեքի ստուգում.

```
var answer = prompt("Enter something", "");
if (isNaN(answer)) {
    alert("Not a number");
} else {
    alert("Number");
}
```

IsNaN-ը իր թվային արգումենտի արժեքը վերափոխում է թվայինի և վերադարձնում է true կամ false: Դատարկ string-ը, կամ пробел-ները (բացատները) վերափոխվում են զրոյի, այդ իսկ պատճառով բերադարձվում է true:

IsFinite(n)-ը վերափոխում է արգումենտը թվայինի և վերադարձնում է true, եթե թվայինի վերափոխված արգումենտը infinity/-infinity/NaN չի:

Փաստորեն թվային արժեքի validation-ի համար կարող ենք գրել հետևյալ function-ը:

```
var answer = prompt("Enter something", "11, abc");
function isNumeric(n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
}
alert(isNumeric(answer));
```

## Թվի կլորացում

Կլորացնում ենք դեպի ներքև՝

```
var f1 = Math.floor(3.5);
document.write(f1); //3
```

Կլորացնում ենք դեպի վերև՝

```
var c1 = Math.ceil(3.5);
document.write(c1); //4
```

Կլորացնում ենք դեպի մոտակա ամբողջ թիվը՝

```
var rn = Math.round(3.5);
document.write(rn); //4
```

Կլորացնում ենք տալով վերադարձվող արժեքի մնացորդային մասի ճշտությունը՝

```
var a = 12.34;
var num1 = a.toFixed(1);
document.write(num1); //12.3
```

Աշխատում ենք string-ի հետ:

Վերցնում ենք string-ի կոնկրետ index-ով symbol-ին.

```
var str = "JavaScript";
var str1 = str.charAt(0);
var str2 = str.charAt(4);
document.write(str1 + str2);
```

Վերափոխում ենք մեծատառերի և փոքրատառերի.

```
var str1 = "javascript";
var str2 = "JAVASCRIPT";
document.write(str1.toUpperCase()); //JAVASCRIPT
document.write(str2.toLowerCase()); //javascript
```

## Slice.

```
var str = 'JavaScript';
document.write(str.slice(0, 3) + "<br>");//Java
document.write(str.slice(0, -6) + "<br>");//Script
document.write(str.slice(4) + "<br>");//Script
```

Այս կոդով prompt-ում մուտք արվող տեքստի առաջին տառը վերափոխում ենք մեծատառի, իսկ մնացածը՝ փոքրատառի.

```
var name = prompt('enter name','');
document.write(name.charAt(0).toUpperCase() + name.slice(1).toLowerCase());
```

## Substr.

Սկիզբ և symbol-ների քանակ:

```
var str = "JavaScript";
document.write(str.substr(4,6) + "<br>");//Script
```

## Substring.

Սկիզբ և վերջ (վերջը ներառված չէ)

```
var str = "JavaScript";
document.write(str.substring(4,6) + "<br>");//Sc
document.write(str.substring(4) + "<br>");//Script
```

Փնտրում ենք str str-ի մեջ.

IndexOf(փնտրվող symbol կամ str, [սկզբնական index])

```
var str = "Lorem ipsum doLor sit amet.";
document.write(str.indexOf("Lorem") + "<br>");//0
document.write(str.indexOf("orem") + "<br>");//1
document.write(str.indexOf("Sit") + "<br>");//-1
document.write(str.indexOf("Lo", 2) + "<br>");//14
```

Վերադարձնում է փնտրվողի սկզբի index կամ -1, եթե չի գտնում:

Array.

Շատ հաճախ կարիք է լինելու մեկ փոփոխականի մեջ պահել ոչ թե մեկ արժեք, այլ մի քանիսը: Այդ պարագայում օգնության է հասնելու օրինակ array-ը:

```
var fruits = ["Apple", "Orange", "Plum"];
console.log(fruits);
alert(typeof fruits);//Object
document.write(fruits[0] + "<br>");//Apple
document.write(fruits[1] + "<br>");//Orange
document.write(fruits[2] + "<br>");//Plum
```

Array-ի անդամի արժեքը կարելի է փոփոխել:

```
fruits[2] = "Strawberry";
console.log(fruits);
```

Կամ ավելացնել՝

```
fruits[5] = "Pear";
document.write(fruits[5] + "<br>");//Pear
console.log(fruits);
```

Array-ի անդամների քանակը՝

```
document.write("Length of fruits is - " + fruits.length);
```

Զանգվածում (array) կարելի է պահել ցանկացած քանակի և type-ի արժեքներ:

```
var arr = [1, 'JavaScript', {name: 'Jon', age: 30}, true];
console.log(arr);
document.write(arr[1] + "<br>");//JavaScript
document.write(arr[2].name + "<br>");//Jon
document.write(arr[2]['age'] + "<br>");//30
```

Array-ի հետ աշխատելու մի քանի մեթոդ՝ ( pop / push / shift / unshift )

Pop() կտրում հանում է array-ի վերջին էլեմենտը և վերադարձնում այն:

```
var lastElements = fruits.pop();
console.log(fruits);
console.log(lastElements); //Melon
```

Push() ավելացնում է array-ում էլեմենտ վերջից:

```
fruits.push("Pear");
console.log(fruits);
```

Shift() կտրում է array-ի առաջին էլեմենտը և վերադարձնում այն:

```
var firstElements = fruits.shift();
console.log(fruits);
console.log(firstElements); //Apple
```

Unshift() ավելացնում է array-ում էլեմենտ սկզբից:

```
fruits.unshift("Strawberry");
console.log(fruits);
```

Push և unshift մեթոդները կարող են միանգամից մի քանի էլեմենտ ավելացնել array-ում:

```
fruits.unshift("Orange", "Orange", "Orange");
fruits.push("Orange", "Orange");
console.log(fruits);

var arr = ["BMW", "KIA", "NISSAN", "HONDA"];
var newArr = arr;
```

Այսպիսի վերագրումից հետո փոփոխականները կապվում են միմյանց հետ հղմամբ, այսինքն մի array-ում փոփոխությունից հետո մյուսն էլ է փոխվելու:

```
var myArr = [];
myArr[200] = 5;
```

Այսպիսի վերագրումից հետո myArr փոփոխականում ստեղծվում են 200 հատ դատարկ անդամ և 201-րդի (index-ով 200) արժեքը դառնում է 5:

```
var arr = ["BMW", "KIA", "NISSAN", "HONDA"];
console.log(arr);
arr.length = 2;
console.log(arr);
```

Իսկ այս տարբերակով եթե array-ի length-ը փոքրացնենք, ապա անդամների կորուստ կունենանք array-ի վերջից:

## Բազմաչափ array

Բազմաչափի են համարվում այն array-ները , որոնք իրենց մեջ պարունակում են array:

```
var carInfo = [ "Nissan", ["Japan", "X_Trail", "50.000$"], "BMW",
["Germany", "X-5", "100.000$"], "KIA", ["Korea", "RIO", "40.000$"] ];
```

Սա երկշափ array է, այս array-ը բացելու համար անհրաժեշտ է՝

```
for (var i = 0; i < carInfo.length; i++) {
  if (Array.isArray(carInfo[i])) {
    for (var j = 0; j < carInfo[i].length; j++) {
      document.write("&ampnbsp&ampnbsp&ampnbsp<span style='color: coral; font-size: 14px;'>" +
        carInfo[i][j] + "</span><br>");
    }
  } else {
    document.write("<span style='color: dodgerblue;'>" + carInfo[i] + "</span><br>");
  }
}
```

Array.isArray()-ը վերադարձնում է true, եթե իրեն որպես  
արգումենտ  
փոխանցած փոփոխականը array է և false հակառակ դեպքում:

Եթե ուզում ենք եռաչափ array բացել, ապա՝

```
var numbers = [1, 2, 3, 4, [5, 6, 7, 8, [9, 10, 11, 12, 13] ] ];  
  
function myArray() {  
    for (var i = 0; i < numbers.length; i++) {  
        if (Array.isArray(numbers[i])) {  
            for (var j = 0; j < numbers[i].length; j++) {  
                if (Array.isArray(numbers[i][j])) {  
                    for (var k = 0; k < numbers[i][j].length; k++) {  
                        if (!Array.isArray(numbers[i][j][k])) {  
                            document.write(numbers[i][j][k]);  
                        }  
                    }  
                } else {  
                    document.write(numbers[i][j]);  
                }  
            }  
        } else {  
            document.write(numbers[i]);  
        }  
    }  
}  
  
myArray();
```

Փաստորեն քանի չափանի մեր array-ն էմ այդքան էլ for ենք օգտագործում, իսկ եթե չգիտենք քանի չափանի է այն array-ը, որը պիտի բացենք:

## Recursia

Եթե function-ը կանչվում է ինքն իր մարմնում, այդ երևույթը կոչվում է recursia`

```
var numbers = [1, 2, 3, 4, [5, 6, 7, 8, [9, 10, [11, 12, [13]]]]];  
  
function rec(arr) {  
    for (var i = 0; i < arr.length; i++) {  
        if (Array.isArray(arr[i])) {  
            rec(arr[i]);  
        } else {  
            document.write(arr[i] + ' ');  
        }  
    }  
}  
  
rec(numbers);
```

## Split (բաժանիչ symbol [ էլեմենտների սահմանափակում ] )

Split մեթոդը string-ը ինչ-որ symbol-ով «կտրում» է ու վերադրձնում է array, որի անդամները string-ի այդ կտորներն են, որպես երկրորդ արգումենտ թիվ փոխանցելով կարող ենք սահմանափակել array-ի անդամների քանակը՝

```
var namesStr = "Armen, Gor, Aram, Tigran";
var namesArr = namesStr.split(',');
console.log(namesArr);
var namesArr = namesStr.split(', ', 2);
console.log(namesArr);
for (var i = 0; i < namesArr.length; i++) {
    document.write('You Have a message ' + namesArr[i] + "<br>");
}
```

Եթե split-ին տանք որպես արգումենտ դատարկ str, ապա այն

լուրջ պատճեն ունի որևէ անուղղակիոր ։ string-ի տարրերն են

```
var str = "JavaScript";
var arr = str.split('');
console.log(arr);
```

## Join (բաժանիչ symbol )

Join-ը կանչվում է array-ի վրա և վերադարձնում է string, որը կազմվում է array-ի անդամներից, միացվում են իրար join-ին որպես արգումենտ տրված symbol-ով:

```
var namesArr = ['Armen', 'Gor', 'Aram', 'Tigran'];
var namesStr = namesArr.join(" / ");
console.log(namesStr);
```

## Splice

```
var carsArr = ['NISSAN', 'BMW', 'AUDI', 'KIA', 'HYUNDAI', 'HONDA'];
var car = carsArr.splice(3,1);
console.log(carsArr);
console.log(car);
```

Splice-ը array-ից անդամներ ջնջելու (նաև ավելացնելու) մեթոդ է, այն array-ի տեսքով վերադարձնում է իր կտրած, հանած կտորը:

```
var carsArr = ['NISSAN', 'BMW', 'AUDI', 'KIA', 'HYUNDAI', 'HONDA'];
var car = carsArr.splice(4,2,"LADA", "VOLGA");
console.log(carsArr);
console.log(car);
```

Կարող ենք անդամներ ավելացնել նաև առանց ջնջելու՝

```
var carsArr = ['NISSAN', 'BMW', 'AUDI', 'KIA', 'HYUNDAI', 'HONDA'];
carsArr.splice(6,0,"MERCEDES", "VOLVO");
console.log(carsArr);
```

## Slice

slice(begin, end)

Copy ենք անում array-ը begin-ից մինչև end՝

```
var carsArr = ['NISSAN', 'BMW', 'AUDI', 'KIA', 'HYUNDAI', 'HONDA'];
var newCarsArr = carsArr.slice(3,5);
console.log(newCarsArr);
```

Եթե արգումենտ չենք տալիս slice-ին, ապա copy է լինում ամբողջ array-ը:

```
var carsArr = ['NISSAN', 'BMW', 'AUDI', 'KIA', 'HYUNDAI', 'HONDA'];
var newCarsArr = carsArr.slice();
console.log(newCarsArr);
```

## Reverse

arr.reverse()- փոխում ենք հերթականությունը.

```
var numArr = [1,2,3,4,5,6,7];
var newNumArr = numArr.reverse();
console.log(newNumArr);
```

## Concat

```
var numArr = [1,2,3];
var newNumArr = numArr.concat(4,5,6);
console.log(newNumArr);
```

Կամ՝

```
var numArr1 = [1,2,3,4,5,6,7];
var numArr2 = [10,20,30];
var newNumArr = numArr1.concat(numArr2);
console.log(newNumArr);
```

## IndexOf

Փնտրում ենք էլեմենտ array-ում, վերադարձնում է գտնելու դեպքում index-ը, չգտնելու դեպքում -1:

```
var carsArr = ['NISSAN', 'BMW', 'AUDI', 'KIA', 'HYUNDAI', 'HONDA'];
if (carsArr.indexOf('AUDI') !== -1) {
  document.write("we found it!!!")
}
```

## ForEach

```
var cars = ["Audi", "Mercedes", "BMW", "Nissan", "Toyota", "Hyundai"];
cars.forEach(function(element, key, cars) {
  document.write('Key: <b>' + key + "</b>
 | Element: <b>" + element + "</b> &nbsp;&nbsp;&nbsp; <b>( " + cars + " )</b><br>");
});
```

Կարելի է պատկերացնել այսպես՝ `forEach`-ը մեր փոխարեն աշխատացնում է `for` ցիկլ `array`-ի և իր արգումենտ ֆունկցիայի վրա՝ որպես առաջին արգումենտ (այս դեպքում էլեմենտ) փոխանցում է էլեմենտները, որպես երկրորդ՝ վերադարձնում է `key`-երը (ինդեքսավորված `array`-ի դեպքում ինդեքսները) և որպես երրորդ՝ ամբողջ `array`-ը:

## Filter

`filter`-ը ստեղծում է նոր `array`, որը ներառում է միայն էլեմենտները, որոնց համար `callback`-ը կվերադանի `true`:

```
var numbers = [1,2,3,4,5,6,7,8,9,10];
var newNumArr = numbers.filter(function (element) {
  if ((element % 2) == 0) {
    return element;
  }
});
document.write(newNumArr);

console.log(newNumArr);
```

## Map

map-ը վերադարձնում է նոր array, որը բաղկացած է callback-ի արդյունքներից:

```
var brands = ["Audi", "Mercedes", "BMW", "Nissan", "Toyota", "Hyundai"];
var brandLetterCount = brands.map(function (element) {
    return element.length;
});
for (var j = 0; j < brandLetterCount.length; j++) {
    document.write(brands[j] + " - " + brandLetterCount[j] + "<br>");
}
```

## Arguments

JS-ում ցանկացած ֆունկցիա կարելի է կանչել՝ տալով նրան ցանկացած քանակի արգումենտ:

```
function go(a, b) {
    alert("a = " + a + ", b = " + b);
}
go(1, 2);

go(1, 2, 3);
```

Ինչպե՞ս ստանալ արգումենտները:

```
function myNum(a, b) {  
    console.log(arguments);  
    for (var i = 0; i < arguments.length; i++) {  
        document.write(arguments[i] + "<br>");  
    }  
}  
myNum(1, 2, 3, 4, 5);
```

arguments-ը array չէ, չենք կարող կիրառել array-ի հետ աշխատելու մեթոդները, այն object է, ուղղակի key-երը թվերն են, և կիրառելի է length-ը: Եթե պետք լինի array-ում ստանալ արգումենտները, ապա՝

```
function myFunc() {  
    var args = [];  
    for (var i = 0; i < arguments.length; i++) {  
        args[i] = arguments[i];  
        // args.push(arguments[i]);  
    }  
    return args;  
}  
var numArray = myFunc(1,2,3,4,5);  
console.log(numArray);
```

Եթե function-ը կանչվել է ավելի քիչ քանակի արգումենտներով քան հայտարարված է, ապա պակասող արգումենտները համարվում են undefined.

```
function showMessage(width, height, title) {  
    if (width === undefined) {  
        width = 200;  
    }  
    if (height === undefined) {  
        height = 100;  
    }  
    if (title === undefined) {  
        title = "Warning";  
    }  
}  
  
showMessage();
```

## Object

Հայտարարում ենք դատարկ object'

ՏԱՐԲԵՐԱԿ 1.

```
var person = Object();
```

ՏԱՐԲԵՐԱԿ 2.

```
var person = {};
```

Դատարկ object - ի մեջ նոր ինֆորմացիա ենք մուտքագրում.

```
person.name = 'Jon';
person.age = 30;
console.log(person);
```

Վերցնում ենք object-ից

```
alert(person.name);
alert(person.age);
document.write("Name: " + person.name + "<br> Age: " + person.age);
```

Զնշում ենք object-ից.

```
delete person.age;
console.log(person);
```

Ստուգում ենք object-ի մեջ տրված key-ի առկայությունը (1)

```
var person = {};
person.age = 30;
console.log(person);
delete person.age;
console.log(person);
if (!("age" in person)) {
    document.write("<br> there is no age key in object <br>");
}
```

Ստուգում ենք object-ի մեջ տրված key-ի առկայությունը (2).

```
var person = {};
person.age = 30;
alert(person.name);
if (person.age !== undefined) {
    alert('Age YES');
}
if (person.name !== undefined) {
    alert('Name YES');
}
```

Երկրորդ տարբերակը այդքան էլ հարմար չէ ստուգելու համար object-ում key-ի գոյությունը, քանի որ՝

```
var person = {};
person.name = undefined;
alert(person.name);
alert(person.age);
if (person.age === undefined) {
    alert('AGE undefined');
}
if (person.name === undefined) {
    alert('NAME undefined');
}
```

Բայց name կա objecti մեջ, ուղղակի նրա արժեքը undefined է: Իսկ (in) օպերատորը երաշխավորում է ճիշտ արդյունք.

```
var obj = {};
obj.test = undefined;
alert("test" in obj);
alert("age" in obj);
```

Մինչ այս մենք object-ի key կանչել ենք կետով՝ obj.test կամ obj.name: Կարող ենք կանչել նաև այսպես՝ **obj['test'] կամ obj['name']:**

Այս գրելաձևերի տարբերությունը հիմնականում կայանում է նրանում, որ կետով գրելիս փոփոխական չենք կարող կանչել, իսկ այս դեպքում կարող ենք՝

```
var obj = {};
var x = 'test';
obj[x] = 'it is testing text';
alert(obj[x]); //it is testing text
alert(obj['test']); //it is testing text
alert(obj.test); //it is testing text
```

Մյուս կարևոր տարբերությունը այն է, որերկու բառից կազմված key կարող ենք տեղադրել կամ կանչել Ո փակագծերով գրելու դեպքում՝

```
var person = {};
person['beautiful color'] = "coral";
document.write(person['beautiful color']); //coral
```

## Ներկառուցված object 1

```
var cars = {  
    brand: "KIA",  
    model: "Optima",  
    "my car": "Nissan",  
    price: {  
        min: "50.000$",  
        max: "80.000$",  
        "new price": "100.000$"  
    }  
};  
console.log(cars);  
document.write(cars.brand + "<br>");//KIA  
document.write(cars["my car"] + "<br>");//Nissan  
document.write(cars.price.min + "<br>");//50.000$  
document.write(cars.price["new price"] + "<br>");//100.000$
```

## Ներկառուցված object 2

```
var cars = {};  
cars.japan = {};  
cars.japan.Nissan = {1: 'X-Trail', 2: 'TIDA', 3: 'SANNY'};  
cars.germany = {};  
cars.germany.brand = 'Mercedes';  
cars.germany.model = 'Brabus';  
console.log(cars);
```

## For in

```
var price = {
    bentley: "200.000$",
    mercedes: "150.000$",
    bmw: "100.000$",
    nissan: "50.000$"
};
for (var key in price) {
    document.write(key + ' : ' + price[key] + "<br>");
}
```

Եթե object-ում key կա, ապա այս դեպքում key փոփոխականի մեջ for-ի ամեն ինտերացիայի ժամանակ հերթով ընկնում են object-ի key-երը: Արդյունքում կարող ենք վերցնել ինչպես key-ը առանձին, այնպես էլ obj[key]-ը, այսինքն այդ key-ի արժեքը:

JS-ում փոփոխականին object վերագրել նշանակում է կապել իրար հղմամբ:

```
var user = {
    name: "Jon",
    age: 40
};
console.log(user);
var newUser = user;
console.log(newUser);
newUser.name = "Armen";
newUser.age = 25;
console.log(user);
console.log(newUser);
```

Նմանատիա վերագրումից հետո մի object-ում փոփոխությունից հետո մյուսն էլ է փոփոխվում համապատասխան կերպով:

Նման խնդրից խուսափելու համար object կրկնօրինակել կարող ենք կլոնավորման տարբերակով:

```
var user = {  
    name: "Jon",  
    age: 30  
};  
console.log(user);  
var userClone = {};  
for (var name in user) {  
    userClone[name] = user[name];  
}  
console.log(user);  
console.log(userClone);  
userClone.country = "USA";  
userClone.age = 50;  
console.log(user);  
console.log(userClone);
```

This

```
var user = {  
    name: "Admin",  
    sayHi: function () {  
        alert("Hello " + user.name);  
    }  
};  
user.sayHi(); // Hello Admin
```

User-ի փոխարեն կարող ենք գրել this, this-ը այս դեպքում լինելու է հենց user object-ը:

```
var user = {
    name: "Admin",
    sayHi: function () {
        alert("Hello " + this.name);
    }
};
user.sayHi(); // Hello Admin
```

Ցանկացած function կարող է ունենալ this, անկախ նրանից թե այդ function-ը object-ում է հայտարարված թե ոչ:

```
var user = {
    firstName: "User"
};
var admin = {
    firstName: "Admin"
};
console.log(user);
console.log(admin);
function getFirstName() {
    alert("Hello " + this.firstName);
}
user.sayHi = getFirstName;
admin.sayHi = getFirstName;
console.log(user);
console.log(admin);
user.sayHi(); // Hello User
user["sayHi"](); // Hello User
admin.sayHi(); // Hello Admin
admin["sayHi"](); // Hello Admin
```

## Proto

Եթե մի object proto հատուկ հղում ունի մեկ այլ object-ի վրա, և իր սեփական key-երի մեջ չի գտնվում մեզ հետաքրքրածը, ապա փնտրում ենք մյուս object-ում:

```
var animal = {  
    eats: true  
};  
var rabbit = {  
    jumps: true  
};  
rabbit.__proto__ = animal;  
// ? rabbit ????? ?????? ??? ???????  
alert( rabbit.jumps ); // true  
alert( rabbit.eats ); // true  
alert( animal.jumps ); // undefined
```

Այն object-ը, որի վրա հղում է կատարվում proto-ի միջոցով կոչվում է prototype մյուս object-ի համար: Այս դեպքում animal-ը հանդիսացավ rabbit-prototype:

```
var animal = {  
    eats: true  
};  
var rabbit = {  
    jumps: true,  
    eats: false  
};  
rabbit.__proto__ = animal;  
alert(rabbit.eats); // false,  
rabbit-ը ունի իր սեփական eats-ը, որը false է:
```

```
var obj1 ={  
    name: 'John'  
};  
var obj2 ={  
    'last name': 'Lennon'  
};  
var obj3 ={  
    age: 40  
};  
obj1.__proto__ = obj2;  
obj2.__proto__ = obj3;  
document.write(obj1.name + ' ' + obj1['last name'] + ' ' + obj1.age);
```

## HasOwnProperty

Սովորական for..in շարբերություն չի դնում object սեփական և proto-ով եկած key-երի միջև

```
var animal = {  
    eats: true  
};  
var rabbit = {  
    jumps: true,  
    __proto__: animal  
};  
for (var key in rabbit) {  
    alert(key);  
}  
alert( rabbit.hasOwnProperty('jumps') ); // true: jumps-ը պատկանում է  
// rabbit-in.  
alert( rabbit.hasOwnProperty('eats') ); // false: eats-ը չի պատկանում  
// rabbit-in.
```

Ստեղծում ենք լրիվ դատարկ object.

```
var data = Object.create(null);
console.log(data);
data.text = "Hello";
alert(data.text); // Hello
alert(data.toString()); // undefined
```

Վերադարձնում է prototype-ը:

```
var obj = {
  data: 'name'
};
var obj2 = {
  data2: 'name2'
};
obj.__proto__ = obj2;
console.log(Object.getPrototypeOf(obj));
```

Կապեցինք իրար proto-նվ:

```
var obj = {
  data: 'name'
};
var obj2 = {
  data2: 'name2'
};
Object.setPrototypeOf(obj, obj2);
console.log(obj.data2);
```

Գլոբալ տիրույթում հայտարարված փոփոխականը դառնում է window object-ի key:

```
var a = 5;
function f() {
    var a = 5;
}
alert(window.a); // 5
```

Function-ի local տիրույթում հայտարարված փոփոխականը չի դառնում window-ի key:

```
function f() {
    var a = 5;
}
alert(window.a); // undefined
```

Բայց մենք կարող ենք function-ի ներսում կանել գլոբալում հայտարարված փոփոխականին և նույնիսկ փոփոխել արժեքը:

```
var a = 5;
function f() {
    a = 8;
}
alert(a); // 8
```

Սա տեղի է ունենում [Scope]-ի միջոցով, այն local-ում չգտնելով փոփոխականը, մի տիրույթ դուրս է գալիս function-ից և փնտրում այնտեղ:

```
function makeCounter() {
    var currentCount = 1;
    return function() {
        return currentCount++;
    };
}
var counter = makeCounter(); // [[Scope]] -> {currentCount: 1}
alert( counter() ); // 1, [[Scope]] -> {currentCount: 1}
alert( counter() ); // 2, [[Scope]] -> {currentCount: 2}
alert( counter() ); // 3, [[Scope]] -> {currentCount: 3}
```

### With Comment

```
var a = 5;
(function() {
    alert(a)
})();
```

Այս գրելածնը նշանակում է, որ function-ը կանչվում է հենց  
հայտարարելիս:

New-ով կանչվող function-ները դառնում են constructor:

Ստեղծվում է նոր, դատարկ object, this-ը ստանում է հղում դեպի այդ object-ը, function-ը կանչվում է, վերադարձվում է this-ը:

```
function Animal(name) {  
    this.name = name;  
    this.canWalk = true;  
}  
let animal = new Animal("dog");  
console.log(animal);
```

New-ով կանչելիս տեղի է ունենում հետևյալը՝

```
//this = {};  
այսուհետև՝  
// this-ում ստեղծվում են key-եր  
this.name = name;  
this.canWalk = true;  
// return this;//վերադարձվում է this object-ը:
```

New-ով կանչելիս եթե արգումենտ չենք փոխանցում, ապա փակագծեր չենք դնում:

```
let animal = new BigAnimal;
```

```
function User(name) {
    this.name = name;
    this.sayHi = function() {
        alert( "My Name Is: " + this.name);
    };
}
let obj = new User("John");
obj.sayHi(); // My Name Is: john
console.log(obj);
```

Constructor-ները ոչինչ չեն վերադարձնում, նրանց խնդիրն է this object-ում գրանցել այն ինչ պետք է:

function-ում եթե return է արվում object, ապա կվերադարձվի այդ object-ը, այլ ոչ թե this-ը, եթե return է արվում պրիմիտիվ արժեք, ապա միևնույն է return է լինելու this-ը՝

```
function BigAnimal() {
    this.name = "Mouse";
    return { name: "Dog" }; // <-- return է լինելու այս object-ը
}
alert( new BigAnimal().name ); // Dog
function BigAnimal() {
    this.name = "Mouse";
    return "Dog"; // <-- return ենք անում հասարակ արժեք
}
alert( new BigAnimal().name ); // Mouse
```

constructor function-ում շատ հաճախ հարմար է լինում օգտագործել օգնական փոփոխականներ կամ function-ներ, որոնք տեսանելի կլինեն միայն constructor-ների ներսում

```
function User(firstName, lastName) {
    // local, օգնական փոփոխական
    let phrase = "Hello";
    // local, որ կարուցված է
    function getFullName() {
        return firstName + " " + lastName;
    }
    this.sayHi = function() {
        alert(phrase + " " + getFullName()); // կիրառություն
    };
}
let obj = new User("John", "Lennon");
obj.sayHi(); // Hello John Lennon
console.log(obj);
```

## Class

class-constructor-ի և prototype-ի տակու համար գրելածն.

```
class ??????? [extends ծնող]  {
    constructor
    մեթոդներ
}
class User {
    constructor(name) {
        this.name = name;
    }
    sayHi() {
        alert(this.name);
    }
}
let user = new User("John");
console.log(user);
console.log(user.name);
user.sayHi(); // John
```

constructor-ը կանչվում է new-ով object սարքելուց, իսկ մյուս մեթոդները գրվում են այս դեպքում User.prototype-ում:

class-ի ներսում հայտարարված function-ները ունեն որոշակի առանձնահատկություններ՝

1. sayHi- համարվում է հենց մեթոդ, այսինքն ունի super-ի հասանելիություն:

2. class-i բոլոր մեթոդները աշխատում են use strict ռեժիմում, նույնիսկ եթե այն միացված չէ:

3. class-ի մեթոդները հաշվելի չեն, այսինքն object սարքելուց հետո for in-ով հասանելի չեն:

Class expression

```
let User = class {
  sayHi() {
    alert('Hello!');
  }
};
new User().sayHi();
```

Այս դեպքում էլ կարող ենք անուն տալ class-ին, բայց այն հասանելի կլինի միայն class-ի ներսում:

```
let SiteGuest = class User {
  sayHi() {
    alert('Hello!');
  }
};
new SiteGuest().sayHi(); // Hello!
new User(); // error
```

## Getter, setter

```
let user = {
    firstName: "John",
    surname: "Lennon",
    get fullName() {
        return this.firstName + ' ' + this.surname;
    },
    set fullName(value) {
        let split = value.split(' ');
        this.firstName = split[0];
        this.surname = split[1];
    }
};
alert(user.fullName); // John Lennon (get-ic)
user.fullName = "Ray Charles";
alert( user.firstName ); // Ray (set-ic)
alert( user.surname ); // Charles (set-ic)
```

Փաստորեն երբ կանչում ենք `fullName`-ը, ապա աշխատում է `get`-ը, հենց նոր արժեք ենք տալիս `fullName`-ին: Այդ արժեքը ընկնում է `set`-ի մեջ, աշխատում է `set`-ը:

## Ժառանգում / extends

```
class Animal {
    constructor(name) {
        this.name = name;
    }
    walk() {
        alert("I walk: " + this.name);
    }
}
class Rabbit extends Animal {
    walk() {
        super.walk();          // super-ը օգտագործվում է ծնողից function-ը
        // կանչելու համար
        alert("...and jump!");
    }
}
new Rabbit("John").walk();
let anim = new Rabbit('John');
anim.walk();
```

New Rabbit-ում հասանելի են ինչպես իր մեթոդները, այնպես էլ ծնողինը:

```
class Animal {
    constructor(name) {
        this.name = name;
    }
}
class Rabbit extends Animal {
    constructor() {
        // alert(this); // error, this- undefined է
        // պարտավոր ենք կանչել super(), ինու նոր դիմել this-ին
        super();
        // alert(this)
        // այստեղ կարելի է օգտագործել  this
    }
}
new Rabbit();
```

Browser-ի պատուհանի չափսերը, առանց գործիքների վահանակի և scroll-ի ժապավենի:

```
var w = window.innerWidth;
var h = window.innerHeight;
alert(w + " x " + h);
```

Browser-ի չափսերը՝

```
var w = window.outerWidth;
var h = window.outerHeight;
alert(w + " x " + h);
```

Navigator object-ը պարունակում է info browser-ի և օպերացիոն համակարգի մասին.

```
function get_browser_name() {
    var ua = navigator.userAgent;
    if (ua.search(/Chrome/) !== -1) return 'Google Chrome';
    if (ua.search(/Firefox/) !== -1) return 'Firefox';
    return '?? ??????????';
}
alert(get_browser_name());
```

Location object-ը պարունակում է info տվյալ URL-ի մասին:՝

```
document.write(location.href);
// Եցի թարմեցում
location.reload();
```

User-ի էկրանի չափսերը.

```
var w = screen.width;
var h = screen.height;
alert(w + " x " + h);
```

Document object-ը հենց մեր \$այլն է.

```
console.log(document);
document.title = "New Title";
//<HTML> = document.documentElement
document.documentElement.style.height = '50px';
document.documentElement.style.border = '2px solid coral';
//<BODY> = document.body
document.body.style.backgroundColor = 'coral';
```

## Selectors

```
document.getElementById()- select ելք անում document-ից id-ով  
document.getElementsByClassName()- select ելք անում document-ից class-ով  
document.getElementsByName()- select ելք անում document-ից name-ով  
document.getElementsByTagName()- select ելք անում document-ից tag-ով  
document.querySelector()- select ելք անում document-ից css-ի selector-ի  
պես, վերադառնում է առաջին հանդիպածը,  
document.querySelectorAll()- select ողջ անում document-ից css-ի selector-ի  
պես, վերադառնում է բոլորը:
```

## PreviousSibling & NextSibling

HTML-ում ունեն՝

```
<p>  
  <span id="span1">  
    Span 1  
  </span>  
  <span id="span2">  
    Span 2  
  </span>  
</p>  
<script>  
var span2 = document.getElementById('span2');  
console.log(span2);  
span2.style.color = 'blue';  
console.log(span2.previousSibling); // Այսորդ Ելեմենտ  
span2.previousSibling.style.color = 'red';  
//parent  
var span1 = document.getElementById('span1');  
console.log(span1.parentNode);  
span1.parentNode.style.backgroundColor = 'blue';  
span1.parentNode.style.color = '#fff';  
</script>  
<ul id="myList" class="myList">  
  <li>Category 1</li>  
  <li>Category 2</li>  
  <li>Category 3</li>  
  <li>Category 4</li>  
  <li>Category 5</li>  
</ul>
```

```
var parentBlock = document.querySelector('ul.myList').children;
console.log(parentBlock);
```

ParentBlock փոփոխականը array է և կարելի է for-ի օգնությամբ հասնել անդամներից յուրաքանչյուրին.

```
for (var i = 0; i < parentBlock.length; i++) {
    parentBlock[i].style.color = 'red';
}
//firstElementChild, lastElementChild -
var parentBlock = document.querySelector('ul');
console.log(parentBlock);
parentBlock.firstElementChild.style.color = 'coral';
parentBlock.lastElementChild.style.color = 'blue';
```

```
var baseBlock2 = document.getElementById('base-block2');
Նախորդ թեզ՝
baseBlock2.previousElementSibling.style.color = 'red';
հաջորդ թեզ՝
baseBlock2.nextElementSibling.style.color = 'blue';
ծնող՝
baseBlock2.parentElement.style.border = '2px solid red';
baseBlock2.parentElement.style.width = '100px';
baseBlock2.parentElement.style.textAlign = 'center';
```

Փնտրում ենք էլեմենտի մեջ.

```
var allElements = document.getElementById('element-block');
console.log(allElements);
var elements = allElements.getElementsByTagName('div');
console.log(elements);
for (var i = 0; i < elements.length; i++) {
    elements[i].style.color = 'red';
}
```

getElementsByName-շատ հազվադեպ օգտագործվող selektor.

```
var elem = document.getElementsByName('fname');
console.log(elem);
for (var i = 0; i < elem.length; i++) {
    elem[i].style.backgroundColor = "coral";
    elem[i].style.color = "white";
    elem[i].style.border = "2px solid coral";
    elem[i].style.padding = "5px 10px";
}
```

elem.matches(css)- մեթոդը ոչինչ չի փնտրում, այլ ստուգում է տվյալ էլեմենտը համապատասխանում է արդյոք իր argument css-ին, վերադարձնում է true կամ false.

```
var elems = document.getElementsByTagName('a');

// cankali e misht stugel inch stacanq
console.log(elems);

for (var i = 0; i < elems.length; i++) {
    if (elems[i].matches('a[href$=".txt"]')) {
        document.write(elems[i].href + '<br>');
    }
}
```

elem.closest(css)- փնտրում է css selektor-ին համապատասխան էլեմենտ դեպի վերև, ի դեպ ըստ-ը նույնպես ներառվում է որոնման մեջ՝

```
<ul>
  <li class="chapter">????? I
    <ul>
      <li class="subchapter">?????
        <span class="num">1.1</span>
      </li>
      <li class="subchapter">?????
        <span class="num">1.2</span>
      </li>
    </ul>
  </li>
</ul>
```

```
<script>
var numberSpan = document.querySelector('.num');
console.log(numberSpan);
// վերևից ամենամու լի էլեմենտը՝
alert(numberSpan.closest('li').className);
// .chapter առ ունեցող ամենամու էլեմենտը՝
alert(numberSpan.closest('.chapter').tagName);
// վերևից ամենամու span-ը
// հենց նույն numberSpan էլեմենտն է
alert(numberSpan.closest('span') === numberSpan);
</script>
```

tagName-պարունակում է tag-ի

անունը՝

```
alert(document.body.tagName); // BODY
alert(document.querySelector('.chapter').tagName); // LI
```

innerHTML- թույլ է տալիս string-ի տեսքով ստանալ էլեմենտի ներսի HTML-ը՝

```
<p id="hi">Hello JQuery !!!</p>
alert(document.getElementById('hi').innerHTML);
document.getElementById('hi').innerHTML = '';
```

outerHTML - string-ի տեսքով վերադարձնում է էլեմենտի HTML-ը ամբողջությամբ, ներառված նաև հենց ինքը՝ էլեմենտը:

```
<div id="text-block-2">some text</div>
<script>
console.log(document.getElementById('text-block-2').innerHTML);
console.log(document.getElementById('text-block-2').outerHTML);
console.log(typeof document.getElementById('text-block-2').outerHTML);
</script>
```

```
<input type="text" name="city" id="city" value="Yerevan"><br>
<script>
var input = document.getElementById('city');
console.log(input);
document.write(input.type + "<br>");
document.write(input.name + "<br>");
document.write(input.id + "<br>");
document.write(input.value + "<br>");
</script>
```

Ասրիբուտից արժեքի ստացում.

```
<div id="elem" href="http://ya.ru" about="Elephant"></div>
<script>
var elem = document.getElementById('elem');
alert(elem.id);
alert(elem.about);
var elem = document.getElementById('fname').getAttribute('myattr');
console.log(elem);
var elem = document.getElementById('fname').setAttribute('myattr', 'kuku');
console.log(document.getElementById('fname'));
</script>
```

- \* elem.getAttribute(name) - ստուգում է attribute-ի առկայությունը
- \* elem.getAttribute(name) - ստանում է attribute-ի արժեքը
- \* elem.setAttribute(name, value) - տեղադրում է attribut-ը
- \* elem.removeAttribute(name) - ջնջում է attribute-ը

```
var element = document.getElementById('myInput');
if (element.getAttribute('name')) {
    alert("Yes");
} else {
    alert("No");
}
var value = element.getAttribute('value');
alert(value);
```

className- class-ները string-ի տեսքով:

classList- class-ները object-ի տեսքով:

- \* elem.classList.contains("class") - վերադարձնում է true եթե elem-ը ունի 'class' klas, և false` հակառակ դեպքում.
- \* elem.classList.add/remove("class") - ավելացնում/ջնջում ենք 'class'-ը:
- \* elem.classList.toggle("class") - եթե կա "class", ապա ջնջում է, եթե չկա՝ ավելացնում է.

```
var div = document.getElementById('myBlock');
div.onclick = function () {
    var hasClass = div.classList.contains('close');
    if (hasClass) {
        div.classList.remove('close');
        div.classList.add('open');
    } else {
        div.classList.remove('open');
        div.classList.add('close');
    }
};
```

Կամ՝

```
document.getElementById('myBlock').onclick = function () {
    this.classList.toggle('open');
};
```

Որտեղ open և close class-ներով համապատասխան style է  
տրված:

Ստեղծում ենք նոր էլեմենտ.

```
var div = document.createElement('div');
console.log(div);
```

createElement-ով ստեղծում ենք նոր էլեմենտ, բայց կարելի է «կոպիտ» պատկերացնել թե վիրտուալ ենք ստեղծում, այսինքն դեռ չի երևում document-ում մեր ստեղծածը:

Ստեղծում ենք text-node.

```
var textElem = document.createTextNode('JavaScript message');

// appendChild(elem) - avelacnum enq element` vorpes verjin child:
var newLi = document.createElement('li');
var parentElement = document.getElementById('list');
newLi.innerHTML = 'JavaScript';
parentElement.appendChild(newLi);
```

form-ի հետ աշխատանք.

```
document.getElementById('numForm').onsubmit = function () {
    var resultList = document.getElementById('resultList');
    var inputVal = document.getElementById('num').value;
    var resultContainer = document.createElement('li');
    resultContainer.innerHTML = "<b><i class='fa fa-angle-right fa-lg fa-fw' aria-hidden='true'></i> " + inputVal + "</b>";
    resultList.appendChild(resultContainer);
    document.getElementById('numForm').reset();
};
```

## Էլեմենտի կլոնավորում՝

```
var messageBlock = document.createElement('div');
messageBlock.className = 'alert alert-success';
messageBlock.innerHTML = "Welcome to the <b>JavaScript <i class='fa fa-code fa-lg' aria-hidden='true'></i></b> course !!!";
document.body.appendChild(messageBlock);
// stexcum enq klony
var messageBlockClone = messageBlock.cloneNode(true);
// փոփոխում ենք նոր ստեղծած էլեմենտը՝
messageBlockClone.querySelector('b').innerHTML = "jQuery <i class='fa fa-code fa-lg' aria-hidden='true'></i>";
// տեղադրում ենք body-ի մեջ
document.body.appendChild(messageBlockClone);
document.body.insertBefore(messageBlockClone, document.body.firstChild);
```

## Էլեմենտի ջնջում՝

```
document.getElementById('delete').onclick = function () {
    var parent = document.getElementById('img-block');
    var img = document.getElementById('sport');
    parent.removeChild(img);
};
```

## Տրված ժամանակամիջոցից հետո ջնջում՝

```
setTimeout(function () {
var div = document.createElement('div');
div.className = 'alert alert-success';
div.innerHTML = '<b>Welcome to the JavaScript!!!</b> <i>message will be
removed after 3 seconds</i>';
document.body.appendChild(div);
    setTimeout(function () {
        div.remove();
    }, 3000);
}, 3000);
```

Եթե պետք է ուղղակի տեքստ տեղադրել կարող ենք օգտագործել՝  
createTextNode(text)

```
createTextNode(text)
var div = document.createElement('div');
div.className = "alert alert-success";
document.body.appendChild(div);
var text = prompt("Enter text for message", "Message!");
div.appendChild(document.createTextNode(text));
// append(), prepend(), after(), before(), replaceWith()
var newP = document.createElement('p');
newP.innerHTML = 'New text';
var parentBlock = document.getElementById('text-block');
parentBlock.prepend(newP);
parentBlock.before(newP);
parentBlock.append(newP);
parentBlock.after(newP);
parentBlock.replaceWith(newP);
```

Որպեսզի առանձին տողերով style չգրենք, կարող ենք օգտվել հետևյալ, ոչ այդքան էլ հարմար գրելածնից:

```
var myBtn = document.getElementById('btn');
myBtn.style.cssText = "\n    outline: none; \n    background-color: coral; \n    border: 1px solid coral; \n    padding: 5px 15px; \n    color: #fff; \n    width: 100px; \n    text-align: center; \n";
```

Style-ը css-ից այսպես չենք կարող ստանալ՝

```
console.log(document.body.style.color);
```

Atributi style- ը կարելի է ստանալ՝

```
console.log(document.body.style.height);
```

Եթե ուզում ենք տվյալ պահին style-ից ինչ-որ արժեքներ ստանալ, ապա կարող ենք օգտագործել.

```
window.getComputedStyle  
var blueBtn = document.getElementById('blue-btn');  
var blueBtnStyle = getComputedStyle(blueBtn);  
alert(blueBtnStyle.color);  
alert(blueBtnStyle.fontSize);  
alert(blueBtnStyle.padding);  
alert(blueBtnStyle.transition);  
/* elementi chapter, scroll */  
var element = document.getElementById('example');
```

```
// offsetLeft/Top ահմանել տեղակայումը ծնողի նկատմամբ:  
document.write("element.offsetLeft: " + element.offsetLeft + "<br>");  
document.write("element.offsetTop: " + element.offsetTop + "<br>");  
// պարունակում է ներքին երկարություն/բարձրություն:  
document.write("element.offsetWidth: " + element.offsetWidth + "<br>");  
document.write("element.offsetHeight: " + element.offsetHeight + "<br>");  
document.write("element.scrollTop: " + element.scrollTop + "<br>");  
document.write("element.scrollHeight: " + element.scrollHeight + "<br>");  
document.write("element.clientTop: " + element.clientTop + "<br>");  
document.write("element.clientLeft: " + element.clientLeft + "<br>");
```

Էլեմենտի չափսերը շրջանակից ներս, ներառված նաև padding-ները՝

```
document.write("element.clientWidth: " + element.clientWidth + "<br>");  
document.write("element.clientHeight: " + element.clientHeight + "<br>");
```

Որպեսզի բացենք էլեմենտը՝

```
element.style.height = element.scrollHeight + 'px';
```

Պատուհանի տեսանելի մասի երկարություն/բարձրություն՝

```
document.write("Width: " + document.documentElement.clientWidth + "<br>");  
document.write("Height: " + document.documentElement.clientHeight + "<br>");
```

Scroll-ի երկարություն/բարձրություն՝

```
document.write("Scroll width: " + document.documentElement.scrollWidth +  
"<br>");  
document.write("Scroll height: " + document.documentElement.scrollHeight +  
"<br>");
```

Էջի scroll-ի չափը տվյալ պահին՝

```
setInterval(function () {  
    console.log(window.pageYOffset);  
    // console.log('???????? ?????: ' + window.pageXOffset);  
}, 1000);
```

Էջի scroll, դեպի տրված կոորդինատներ՝

```
document.getElementById('scrollLink').onclick = function () {
    window.scrollTo(0, 556);
    return false;
```

Արգելում ենք scroll՝

```
document.body.style.overflow = "hidden";
function question() {
    alert('Start ?');
}
function start() {
    this.classList.add('circle-right');
}
var circleBtn = document.getElementById('circle');
circleBtn.onclick = question;
circleBtn.onclick = start;
```

Նույն էլեմենտի վրա, նույն event-ը գրելու դեպքում կկատարվի միայն վերջինը: Եթե ուզում ենք հերթականություն, ապա՝

```
circleBtn.addEventListener("click", question);
circleBtn.addEventListener("click", start);
```

Event object-ից ստանում ենք տեղեկություն event-ի մասին՝

```
document.getElementById('btn').onclick = function(event) {
    console.log(event);
    // event-ի տեսակ, ելեմենտ որի վրա կատարվեց event, կոորդինատներ
    // alert(event.type + " ?? " + event.currentTarget);
    // alert(event.clientX + ":" + event.clientY);
};
```

```
//form.elements psevdomassive
document.forms.my -- 'my' անունով ֆորմաներ
document.forms[0] -- document-ի առաջին ֆորման
var firstForm = document.forms.login;
firstForm.elements.login.style.backgroundColor = 'coral';
var secondForm = document.forms[1];
secondForm.elements.search.style.backgroundColor = 'blue';
```

Ցանկացած form-ի էլեմենտներ կարելի է վերցնել form.elements-ի օգնությամբ՝

```
document.getElementById('login').onsubmit = function () {
    alert(this.elements.login.value);
    alert(this.elements.password.value);
};
```