null undefined 0 NaN "" false (13)

1 2 3 4 5 6

① ② ③ ④ ⑤ ⑥

```
1. // falsy values // null undefined 0 NaN "" false
var foobar;
console.log(foobar); //undefined
2.
var foobar=null;
console.log(typeof foobar); //object
3.
var foobar=undefined-2;
console.log(foobar); // NaN
4.
var arr=[];
console.log(typeof arr); // object
5.
var foobar=+true;
console.log(foobar); // 1
var foobar=+false;
console.log(foobar); // 0
6.
var foo = 5;
console.log(!foo); // false
var foo = "0";
console.log(!foo); // false
7.
var fn = function(){};
var obj = new fn;
console.log(obj instanceof fn); // true
8.
var foobar = function(){};
var b = " " + foobar;
console.log(typeof b); // string
9.
var foo = false;
var bar = false;
var barfoo = 5;
var foobar = foo || bar || barfoo;
console.log(foobar); // 5
10.
var foo = true;
var bar = function(){
        console.log("Hello World");
}

foo && bar(); // hello world
11.
var foo=5;
console.log(!!foo); // true
12.
var f = function(){
        console.log("Hello World");
}
var foobar = function(fn)
{
        fn();
}
foobar(f); // Hello World
13.
var fn = function(){};
console.log(typeof fn()); // undefined
14.
var falsyValue = NaN;
if(!falsyValue)
{
        console.log("falsy value");
}
15.
var falsyValue = "";
if(falsyValue==0)
{
        console.log("falsy value");
}
16.
var falsyValue;
if(falsyValue==null)
{
        console.log("falsy value");
}
17.
var foo=null;
var bar;
console.log(foo==bar); // true
console.log(0==null); // false
18.
(function(i){
        delete i;
        //return i;
        console.log(i); // 10
})(10);
19.
var fn = function(){
        var i = 10;
}
fn();
console.log(typeof i); // undefined
20.
var fn = function(){
        i = 10;
}
fn();
console.log(i); // 10
21.
var i = 10;
var fn = function(){
        console.log(i); // undefined
        var i = 1000;
        console.log(i); // 1000
}
fn();
```

```
22.
var fn = function(){
        var i = 100;
        var foobar = function(){
                console.log(i); // 100
                i=300;
        };
        foobar();
        console.log(i); // 300
}
fn();
```

```
23.
var obj = {
        property:"foo",
        method:function(){
                return this.property;
        }
};
console.log(obj.method()); // foo
```

```
24.
var foo = {
        bar:function(){
                return this.f;
        },
        f:1
};
var f =foo.bar;
console.log(typeof f); // function
```

```
25.
var fn = function(){};
fn.method = function(){
        console.log(this===fn); // true
};  // function anonymous
fn.method();
```

```
26.
var fn = function(){};
fn.method = function(){
        console.log("foobar");
};
var obj = new fn;
obj.method(); // error
```

can skip

```
27.
var fn = function(a,b)
{
        console.log(this); // [Number: 5]
        console.log('rrrr');
  console.log(a);
  console.log('ssss');
        console.log(a*this+b); // 7
}
fn.call(5,1,2);
```

```
28.
var fn = function(a,b)
{
        return this;
}
var obj = {foobar:"foobar"};
console.log(obj ===fn.call(obj)); // true
```

```
29.
var fn = function(a,b,c)
{
        console.log(a+b+c); // 6
}
fn.apply(this,[1,2,3]);
```

```
30.// puzzle
var fn = function()
{
        console.log(this); // [object window]
}
fn.call(null);
```

can skip

```
31.
var obj = (function(){
        var priv = 200;
        return {
                method:function(){},
                method2:function(){},
                property:1000
        };
})();

console.log(obj.priv); // undefined
```

```
32.///////
var base = function(){
        return {
                method:function(){
                        console.log(100);
                }
        }
}
var child = function(){
        var that = base();
        that.method2 = function(){
                console.log(200);
        }
        return that;
}

var obj = new child;
obj.method(); // 100
```

() [optional]

```
33.
var com = {};
com.testApp = {};
com.testApp.utils = {
        trim:function(){
                console.log("Hello World");
        }
};
com.testApp.utils.trim();
```

ok

```
34. // ES6 for-of statement
let colors = ['red', 'green', 'blue'];
for (const color of colors){
   console.log(color);
}
```

```
35. // remove empty or extra values
console.log([1, false, "", undefined,
2].filter(Boolean)); // [1, 2]
console.log(['a','b','',,,'w','b'].filter(v => v));
```

```
36. // Validate decimal numbers in JavaScript - IsNumeric()
const IsNumeric = (num) => /^-{0,1}\d*\.{0,1}\d+
$/.test(num);
console.log(IsNumeric("do"));
```

```
37. // remove duplicates
var arr = ["a","b","c","d","a","e","f"];
let unique = [...new Set(arr)];
console.log(unique);
```

```
38. // range
console.log([...Array(5).keys()]);
console.log(Array.from(Array(20).keys()))
```

```
39. // es6 spread operator
const codeburst = 'ABCDEF';
const characters = [ ...codeburst ];
console.log(characters);
```

```
40. // spread operator
const obj1 = { firstName: 'Foo', age: 22 };
const obj2 = { lastName: 'Bar', gender: 'M' };
const newObj = { ...obj1, ...obj2, planet: 'Earth' };
console.log(newObj)
```

```
41. // Sort Numbers in Ascending Order
function fAsc(arr) {
  return (arr || []).sort((a,b)=>a-b)
}
var result = fAsc([1, 2, 10, 50, 5]);
console.log(result);
```

```
42. // to sum of cubes
function cube(nums) {
  return nums.reduce((p,c) => p + Math.pow(c,3),0);
}
console.log(cube([1,5,9]));
```

```
43. // How Many Decimal Places
const getDecimalPlaces = s => (s.split(".")[1] ||
[]).length;
console.log(getDecimalPlaces("43.20"));
```

```
44. // How Many Vowels
function countVowels(str) {
  return str.match(/[aeiou]/g).length;
}
console.log(countVowels("the quick brown fox
jumped over the lazy dog."))
```

```
45. // Repeating Letters
function doubleChar(str) {
  return str.split('').map(x => x + x).join('');
}
console.log(doubleChar("Hello World!")); //
"HHeellllloo  WWoorrlldd!!"
```

```
46. // Converting Objects to Arrays
var obj = { a: 1, b: 2 };
console.log(Object.entries(obj));
```

```
47.
const sum = arr => arr.reduce((a, b) => a + b, 0);
console.log(sum([1,2,3,4,5,6]));
```

```
48. // es6 for-of
let colors = ['red', 'green', 'blue'];
for (const color of colors){
   console.log(color);
}
```

```
49.    // es6 Currying
const concat = (str1) => (str2) => (str3) => {
   console.log(`${str1} ${str2} ${str3}`);
}
concat("JavaScript")("For")("Life");
```

```
50. // asc sort
var users = [
  { firstname: "Anna", id: 318},
  { firstname: "Cnna", id: 319},
  { firstname: "bnna", id: 320},
  { firstname: "dnna", id: 321},
  { firstname: "bnna", id: 322},
];
console.log(users.sort((a, b) =>
a.firstname.localeCompare(b.firstname)))
```

```
51. // simplest way to merge ES6 Sets

var set1 = new Set([1,2,3]); // {1,2,3}
var set2 = new Set([4,5,6]); // {4,5,6}
var set3 = new Set([7,8,9]); // {7,8,9}

//var arr = Array.from(set);//[1,2,3]

//For sets:
var merged = new Set([...set1, ...set2, ...set3]);
console.log(merged);
```

```
52. // simplest way to merge ES6 Maps
var map1 = new Map([[1,2],[2,3]]); // map = {1=>2, 2=>3}
var map2 = new Map([[4,5],[5,6]]); // map = {1=>2, 2=>3}
var map3 = new Map([[6,7],[7,8]]); // map = {1=>2, 2=>3}

var merged = new Map([...map1, ...map2, ...map3]);
console.log(merged);
```

```
// Filter object properties by key in ES6

const data = {
 item1: { key: '1sdfd', value:'1sdfd' },
 item2: { key: '2sdfd', value:'2sdfd' },
 item3: { key: '3sdfd', value:'3sdfd' }
};

const { item2, ...newData } = data;
console.log(item2);
console.log(newData);
```

```
// Sorting an array of objects by property values
var homes = [
{"h_id": "3", "city": "Dallas", "state": "TX", "zip": "75201",
"price": "162500"},
{"h_id": "4", "city": "Bevery Hills", "state": "CA", "zip": "90210",
"price": "319250"},
{"h_id": "5", "city": "New York", "state": "NY", "zip": "00010",
"price": "962500"}
];
homes.sort((a, b) => parseFloat(a.price) - parseFloat(b.price));
console.log(homes);
```

```
// dynamic variable
const dynamic = 'color'
var item = {
 brand: 'Ford',
 [dynamic]:'Blue'
}
console.log(item);
```

```
// es6 object properties by key in ES6

const data = {
 item1: { key: 'sdfd', value:'sdfd' },
 item2: { key: 'sdfd', value:'sdfd' },
 item3: { key: 'sdfd', value:'sdfd' }
};

const { item2, ...newData } = data;
console.log(item2);
console.log(newData);
```

```
// closure
var globalVariable = "I am a Global variable";
const greetWith = (greet) => {
    return (name) => {
      console.log(globalVariable); // "I am a Global variable"
      console.log(`${greet} ${name}`); // Hello Anto
    }
}
const greet = greetWith("Hello"); // returns a
Function
greet("Anto");
```

```
////// ES6
const isType = type => target =>
Object.prototype.toString.call(target) === `[object ${type}]`;
const isObject = isType('Object');
const isNumber = isType('Number');
const isArray = isType('Array');

console.log(isObject({foo: 'bar'})) //true
console.log(isNumber(5)); // true
console.log(isArray([])); // true
```

```
// nested Destructing

const user = {
 id: 339,
 name: 'Fred',
 age: 42,
 education: {
  degree: 'Masters'
 }
};
const { education: { degree } } = user;
console.log(degree); //prints: Masters
```

```
//
var fn = function(){};
console.log(typeof fn.prototype); // object
```

```
//
var fn = function(){
};
fn.prototype.foo = "bar";
var obj = new fn;
fn.prototype.foo = "foo";
console.log(obj.foo); // foo
```

```
//
var fn = function(){
      this.hello = "Hello World";
};
fn.prototype = {
      say:function(){
            console.log(this.hello);
      }
};
var obj = new fn;
obj.say(); // Hello World
```

```
// js substring
const string = "foo";
const substring = "oo";
console.log((new RegExp(substring)
      .test(string));
console.log(/oo/.test(string));
console.log(string.indexOf(substring) !== -1);
console.log(string.includes(substring));
```

51

63. // using reduce to count items in array

```
var cars = ['BMW', 'Benz', 'Benz', 'Tesla', 'BMW', 'Toyota'];
var carsObj = cars.reduce(function (obj, name) {
 obj[name] = obj[name] ? ++obj[name] : 1;
 return obj;
}, {});

console.log(carsObj); // => { BMW: 2, Benz: 2,
Tesla: 1, Toyota: 1 }
```

64.

```
Function.prototype.say = function(){
        console.log("Hello World");
}
var fn = function(){};
fn.say(); // Hello World
```

65. ////////////////////////////////

```
var obj = {
        property:"foobar",
        method:function(){}
};

for(var i in obj)
{
        console.log("The value of key "+i+" is " +
obj[i]);
}

for(var i in obj)
{
        if(obj.hasOwnProperty(i))
        {
                console.log("The value of key "+i+" is
" + obj[i]);
        }
}
```

http methods

ACL BIND CHECKOUT
CONNECT COPY DELETE
GET HEAD LINK LOCK
M-SEARCH MERGE
MKACTIVITY MKCALENDAR
MKCOL MOVE NOTIFY
OPTIONS PATCH POST
PROPFIND PROPPATCH PURGE
PUT REBIND REPORT
SEARCH SUBSCRIBE TRACE
UNBIND UNLINK UNLOCK
UNSUBSCRIBE

```
// difference
var a1= ['a', 'b'];
var a2 = ['a','b','c','d'];
console.log (a2.filter ( d =>
    !a1.includes(d)));
```

~~// intersection~~

~~Console.log(a2.filter(d => a1.includes(d)));~~

~~d => !includes~~

```
// intersection
console.log (
a2.filter ( d =>
a1.includes(d)));
```

has

union

```
// first / last element
let array =[1,2,3,4,5,6,7,8];
let {0:a,[array.length-1]:b}=array;
console.log(a,b);

// count duplicates
var counts = {};
['a','b','c','a','a'].forEach(function(x)
    { counts[x] = (counts[x] || 0)+1; });
console.log(counts);

// union
console.log([...new Set([...a1,...a2])]);
```

8570
47000

① const all = (arr, fn = Boolean) => arr.every(fn);
console.log(all([4, 2, 3], x => x > 1));
console.log(all([1, 2, 3]));

---

② const allEqual = arr => arr.every
(val => val === arr[0]);

console.log(allEqual([1, 2, 3...]));
console.log(allEqual([1, 1, 1, 1]));

---

③
const arrayToCSV = (arr, delimiter = ',') =>
arr.map(v => v.map(x => `"${x}"`)
.join(delimiter)).join('\n');

console.log(arrayToCSV([['a', 'b'], ['c', 'd']
]));

console.log(arrayToCSV([['a', 'b'],
['c', 'd']], ';'));

---

④
const capitalizeEveryword = str =>
str.replace(/\b[a-z]/g,
char => char.toUpperCase());
console.log(capitalizeEveryword('hello world'));

5
```
const countOccurence = (arr, val) =>
    arr.reduce((a,v) => (v===val ? a+1 : a), 0);

console.log(countOccurence([1,1,2,1,3], 1));
```

6
```
const digitize = n => [...`${n3}`].map(
                    i => parseInt(i));

digitize(431);
```

---

7 indexOfAll
```
const indexOfAll = (arr, val) =>
    arr.reduce((acc, el, i) =>
        (el === val ? [...acc, i] : acc),
                    [ ]);

console.log(
indexOfAll([1,2,3,1,2,3], 1));
```

8
```
const isBrowser = () =>
    ![typeof window, typeof document].
      includes('undefined');

isBrowser();
```

9
```
const isBrowserTabFocused = () => !document.hidden;
isBrowserTabFocused();
```

10
```
const isValidJSON = str => {
  try {
    JSON.parse(str);
    return true;
  } catch (e) {
    return false;
  }
};
isValidJSON('{"name":"...","age":203'); // valid
isValidJSON('{"name":"...";"age":203'); // invalid
```

11
```
const negate = func => (...args) => !func(...args);
[1, 2, 3, 4, 5, 6].filter(negate(n => n % 2 === 0));
// [1, 3, 5]
```

12
```
const nodeListToArray = nodeList => [...nodeList];
nodeListToArray(document.childNodes);
// [ <!DOCTYPE html>, html ]
```

```
consale.log([] == ![]);      // -> true
+[] == +![]);                // -> true
(0 == +false);               // -> true
(false == []);               // true
(false == ![]);              // true
```

toNumber(true); // 1

toNumber([]); // 0

(![]) // false

toNumber([]); // 0

!!"false" == !!"true" // true

!!"false" === !!"true" // true

true == "true"

false == "false"

"foo" + + "bar" : foo NaN

NaN NaN === NaN // false

-0 === 0 // true

Object.is(NaN, 0/0) // true

1 == true // true

Boolean(1.1) // true

1.1 == true // false

(1 == Number(true))

```
+![]        // 0
+!![]       // 1
+{}3        // NaN
!![]        // true
!!null      // false
0 == false  // true
"" == false // true
```

```
null > 0        // false
null == 0       // false
null >= 0       // true

+null == +0

null >= 0
!(null < 0)
!(+null < +0)   true
!(0 < 0)        true
!false          true
```

```
document.all instanceof Object;   // true
typeof document.all;              undefined
document.all == null              // true
```

```
[] == ''        // true
[] == 0         // true
[''] == ''      // true
[0] == 0        // true
[0] == ''       // false
['' ] == 0      // true
[null] == ''    // true
[null] == 0     // true
[undefined] == ''   // true
[undefined] == 0    // true
```

```
[[]] == 0       // true
[[]] == ''      // true
```

node and hence protect the network from the Sybil attack. At last, the QOS parameters are evaluated to determine the efficiency of the network. Some of them are listed below

1) Throughput

3) Error Rate

4) Energy Consumption

5) Delay

```
true + true          // 2
(true + true) * (true + true) - true;   // 3
Number (true);       // 1
+true                // 1
typeof NaN           // number
typeof []            // object
typeof null          // object
null instanceof Object   // false
1 < 2 < 3            // true
3 > 2 > 1            // false
{3} + []             [object object] // 0
[4, 4] * [4, 4]      // NaN
"str" instanceof String   // false
String ("str") == "str";  // true
new String ("str") == "str"; // true
typeof new String ("str");
```

```
const c = "constructor";
c[c][c]('console.log("WTF?")')(); // WTF
```

String → [Function: String]

[Function: Function]

```
console.log( '${{ Object }}' );
```

[Object Object]

```
consale.log( [...[... "..."]].length); // 3
```

---

arguments with arrow functions.

```
let f = () => arguments;
console.log(f("a")); // -> Uncaught ReferenceError:
                        arguments is not defined

let f2 = (...args) => args;
console.log(f2("a")); // [ 'a' ]
```

"a", "b", "c" → returns array

tricky return

```
console.log(
  (function() {
    return
    {
      b:10
    }
  })();
);
```

in next line return undefined

can eliminate

// -> undefined

```
(function() {
  return {
    b:10
  };
})(); // -> {b:10}
```

return object in same line

optional

Chaining assignment

```
var foo = {n:1};
var bar = foo;
foo.x = foo = {n:2};
foo.x; // -> undefined
foo; // -> {n:2}
bar; // -> {n:1, x:{n:2}}
```

```
var obj = { property: 1 };
var array = ["property"];
obj[array];  // -> 1
```

```
// works in strict mode
var a, a, a;
var a;
var a;
```

```
{}{}; -> undefined
{foo: 'bar'}{}; -> bar
{}{foo: 'bar'}; -> bar
{}{foo: 'bar'}{}; -> bar
{a: 'b'}{c: 'd'}{}; -> 'd'
{a: 'b', c: 'd'}{}; -> Syntax error: Unexpected
                       token ':'
({}{}); -> Syntax error: Unexpected
           token '{'
```

```
if (true) {
  foo: "bar";
}
```

An infinite timeout

```
setTimeout ( () => console.log ("called"), Infinity);
                    // -> <timeoutId>

setTimeout ('{a:13', 100);  // -> <timeoutId>

setTimeout ({a:13, 100);
    // -> 'Uncaught SyntaxError: Unexpected identifier
```

```
27.toString()  // -> Uncaught SyntaxError:
                    Invalid or unexpected token

27..toString();   // -> 27
```

. represent numeric literal syntax of Ecma script

```
(27).toString();   // -> 27

27..toString();    // -> 27
```

```
class SomeClass {
    ["array"] = []
    ["string"] = "str"
}
console.log(
  new SomeClass().array;      // -> str
);
```

ES6 (shallow copy)

```
var A1 = {a: "2"};
var A2 = Object.assign({}, A1);
var A3 = {...A1};
```

ES6 scopes global (1), module (2), function (3), block (4)

ES6 // dynamically add functions to to classes

```
['GET', 'PUT', 'POST', 'DELL'].forEach(method => {
    Executor.prototype[method] = function (body) {
        return this.request(method, body)
    }
})

const objToArray = object => Object.keys(object)
  .map( el => [el, object[el]] );

objToArray({a:4, b:53});
          ({x:1, y:2, z:3})
```

```
const sumNumbers = (...array) =>
        [...array].reduce((accumulator,
    currentValue) => accumulator + currentValue, 0);

sumNumbers(5,6,7,8,9,10);
sumNumbers(...[1,2,3,4,5,6,7,8,9,10]);
```

IIFE (Immediately Function Expressions) Blocks

```
(function () {
    var food = "meo mix";
})();
console.log(food); // Reference Error
```

ES6 blocks

```
{ let food = "meo mix";
};
console.log(food); // Reference Error.
```

ES6 repeat

```
'meo'.repeat(3);
```

818
915