```
// {}{} is undefined**
{}{}; // -> undefined
{}{}{}; // -> undefined
{}{}{}{}; // -> undefined
{foo: 'bar'}{}; // -> 'bar'
{}{foo: 'bar'}; // -> 'bar'
{}{foo: 'bar'}{}; // -> 'bar'
{a: 'b'}{c:' d'}{}; // -> 'd'
{a: 'b', c: 'd'}{}; // > SyntaxError: Unexpected token ':'
({}{}); // > SyntaxError: Unexpected token '{'

// Double dot**
27.toString() // > Uncaught SyntaxError: Invalid or unexpected token
27..toString(); // -> '27'

// [] is equal ![]**
[] == ![]; // -> true

// true is not equal ![], but not equal [] too**
true == []; // -> false
true == ![]; // -> false
false == []; // -> true
false == ![]; // -> true

// Array equality is a monster**
[] == ''   // -> true
[] == 0    // -> true
[''] == '' // -> true
[0] == 0   // -> true
```

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

// Magically increasing numbers**
999999999999999; // -> 999999999999999
9999999999999999; // -> 10000000000000000
10000000000000000; // -> 10000000000000000
10000000000000000 + 1; // -> 10000000000000000
10000000000000000 + 1.1; // -> 10000000000000002

// true is false**
!!"false" == !!"true"; // -> true
!!"false" === !!"true"; // -> true

// Funny math**
3  - 1  // -> 2
3  + 1  // -> 4
```

```
'3' - 1  // -> 2

'3' + 1  // -> '31'

'' + '' // -> ''

[] + [] // -> ''

{} + [] // -> 0

[] + {} // -> '[object Object]'

{} + {} // -> '[object Object][object Object]'

'222' - -'111' // -> 333

[4] * [4]       // -> 16

[] * []         // -> 0

[4, 4] * [4, 4] // NaN


// baNaNa

"b" + "a" + +"a" + "a"; // -> 'baNaNa'


// NaN is not a NaN**

NaN === NaN; // -> false


// Object.is() and === weird cases**

Object.is(NaN, NaN); // -> true

NaN === NaN; // -> false

Object.is(-0, 0); // -> false

-0 === 0; // -> true

Object.is(NaN, 0 / 0); // -> true

NaN === 0 / 0; // -> false


// [] is truthy, but not true**

!![]      // -> true

[] == true // -> false
```

```
// null is falsy, but not false**
!!null; // -> false
null == false; // -> false
0 == false; // -> true
"" == false; // -> true

// document.all is an object, but it is undefined**
document.all instanceof Object; // -> true
typeof document.all; // -> 'undefined'
document.all === undefined; // -> false
document.all === null; // -> false
document.all == null; // -> true

// Minimal value is greater than zero**
Number.MIN_VALUE > 0; // -> true

// Function is not a function**
// declare a class which extends null
class Foo extends null {}
// -> [Function: Foo]
new Foo() instanceof null;
// > TypeError: function is not a function
// >     at … … …

// Super constructor null of Foo is not a constructor**
class Foo extends null {}
new Foo() instanceof null;
// > TypeError: Super constructor null of Foo is not a constructor
```

```
// Adding arrays**
[1, 2, 3] + [4, 5, 6]; // -> '1,2,34,5,6'

// Trailing commas in array**
let a = [, , ,];
a.length; // -> 3
a.toString(); // -> ',,'

// undefined and Number**
Number(); // -> 0
Number(undefined); // -> NaN

// parseInt is a bad guy**
parseInt("f*ck"); // -> NaN
parseInt("f*ck", 16); // -> 15
parseInt(null, 24); // -> 23
parseInt("06"); // 6
parseInt("08"); // 8 if support ECMAScript 5
parseInt("08"); // 0 if not support ECMAScript 5
parseInt({ toString: () => 2, valueOf: () => 1 }); // -> 2
Number({ toString: () => 2, valueOf: () => 1 }); // -> 1
parseInt(0.000001); // -> 0
parseInt(0.0000001); // -> 1
parseInt(1 / 1999999); // -> 5

// Math with true and false**
true + true; // -> 2
(true + true) * (true + true) - true; // -> 3
```

```
// HTML comments are valid in JavaScript**
// valid comment
<!-- valid comment too

// NaN is not a number**
typeof NaN; // -> 'number'

// Precision of 0.1 + 0.2**
0.1 + 0.2; // -> 0.30000000000000004
0.1 + 0.2 === 0.3; // -> false

// Patching numbers**
Number.prototype.isOne = function() {
   return Number(this) === 1;
 };

 (1.0).isOne(); // -> true
 (1).isOne(); // -> true
 (2.0).isOne(); // -> false
 (7).isOne(); // -> false

// Comparison of three numbers**
1 < 2 < 3; // -> true
3 > 2 > 1; // -> false

// Addition of RegExps**
// Patch a toString method
RegExp.prototype.toString =
```

```
 function() {
   return this.source;
 } /
 7 /
 -/5/; // -> 2

// Strings aren't instances of String**
"str"; // -> 'str'
typeof "str"; // -> 'string'
"str" instanceof String; // -> false

// Calling functions with backticks**
function f(...args) {
   return args;
 }
 f(1, 2, 3); // -> [ 1, 2, 3 ]
 f`true is ${true}, false is ${false}, array is ${[1, 2, 3]}`;
// -> [ [ 'true is ', ', false is ', ', array is ', '' ],
// ->   true,
// ->   false,
// ->   [ 1, 2, 3 ] ]

// Call call call**
console.log.call.call.call.call.call.apply(a => a, [1, 2]);

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

```
// Object as a key of object's property**
{ [{}]: {} } // -> { '[object Object]': {} }

// Accessing prototypes with __proto__**
(1).__proto__.__proto__.__proto__; // -> null

// `${{Object}}`**
`${{ Object }}`;
// -> '[object Object]'

// Destructuring with default values**
let x,
  { x: y = 1 } = { x };
y;
// -> 1

// Dots and spreading**
[...[..."..."]].length; // -> 3

// Labels**
foo: {
   console.log("first");
   break foo;
   console.log("second");
  }

  // > first
  // -> undefined
```

```
// Nested labels**
a: b: c: d: e: f: g: 1, 2, 3, 4, 5; // -> 5

// Insidious try..catch**
(() => {
   try {
     return 2;
   } finally {
     return 3;
   }
 })();
// --> 3

// Is this multiple inheritance?**
new class F extends (String, Array) {}(); // -> F []

// A generator which yields itself**
(function* f() {
   yield f;
 })().next();
 // -> { value: [GeneratorFunction: f], done: false }

// A class of class**
typeof new class {
   class() {}
 }(); // -> 'object'
// Tricky arrow functions**
let f = () => 10;
f(); // -> 10
```

```
let f = () => {};
f(); // -> undefined

// Arrow functions can not be a constructor**
let f = function() {
   this.a = 1;
 };
 new f(); // -> f { 'a': 1 }
let f = () => {
   this.a = 1;
 };
 new f(); // -> TypeError: f is not a constructor

//   arguments and arrow functions**
let f = function() {
   return arguments;
 };
 f("a"); // -> { '0': 'a' }
let f = () => arguments;
 f("a"); // -> Uncaught ReferenceError: arguments is not defined
// Tricky return**
(function() {
   return
   {
     b: 10;
   }
 })(); // -> undefined

// Chaining assignments on object**
```

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

// Accessing object properties with arrays**
var obj = { property: 1 };
var array = ["property"];
obj[array]; // -> 1
// this also works with nested arrays
var nestedArray = [[[[[[[[[["property"]]]]]]]]]];
obj[nestedArray]; // -> 1

// Number.toFixed() display different numbers**
(0.7875).toFixed(3);
// Firefox: -> 0.787
// Chrome: -> 0.787
// IE11: -> 0.788
(0.7876).toFixed(3);
// Firefox: -> 0.788
// Chrome: -> 0.788
// IE11: -> 0.788

// Math.max() less than Math.min()**
Math.min() > Math.max(); // -> true
Math.min() < Math.max(); // -> false
```

```
// Comparing null to 0**
null == 0; // -> false
null > 0; // -> false
null >= 0; // -> true

// Same variable redeclaration**
a;
a;
// This is also valid
a, a;
var a, a, a;
var a;
var a;

// Default behavior Array.prototype.sort()**
[10, 1, 3].sort(); // -> [ 1, 10, 3 ]

// resolve() won't return Promise instance**
const theObject = {
   a: 7
 };
 const thePromise = new Promise((resolve, reject) => {
   resolve(theObject);
 }); // Promise instance object

 thePromise.then(value => {
   console.log(value === theObject); // > true
   console.log(value); // > { a: 7 }
 });
```

```
// arguments binding**
function a(x) {
   arguments[0] = "hello";
   console.log(x);
  }

  a(); // > undefined
  a(1); // > "hello"

// An alert from hell**
[666]["\155\141\160"]["\143\157\156\163\164\162\165\143\164\157\162"](
   "\141\154\145\162\164(666)"
  )(666); // alert(666)

// An infinite timeout**
setTimeout(() => console.log("called"), Infinity); // -> <timeoutId>
// > 'called'

// A setTimeout object**
setTimeout(123, 100); // -> <timeoutId>
// > 'called'
setTimeout('{a: 1}', 100); // -> <timeoutId>
// > 'called'

// Extra Newness**
class Foo extends Function {
   constructor(val) {
     super();
```

```
    this.prototype.val = val;
  }
 }

 new new Foo(":D")().val; // -> ':D'

// Why you should use semicolons**
class SomeClass {
   ["array"] = []
   ["string"] = "str"
 }

 new SomeClass().array; // -> 'str'

// Split a string by a space**
"".split(""); // -> []
// but…
"".split(" "); // -> [""]

// A stringified string**
JSON.stringify("production") === "production"; // -> false

// Non-strict comparison of a number to true**
1 == true; // -> true
// but…
Boolean(1.1); // -> true
1.1 == true; // -> false
```