JavaScript не позволяет настраивать, как операторы работают с объектами. Мы не можем реализовать специальный объектный метод для обработки сложения (или других операторов). В случае таких операций, объекты автоматически преобразуются в примитивы, затем выполняется сама операция над этими примитивами, и на выходе получим примитивное значение.

Результатом obj1 + obj2 (или другой математической операции) не может быть другой объект!

**[Правила преобразования](https://learn.javascript.ru/object-toprimitive" \l "pravila-preobrazovaniya)**

1. Не существует преобразования к логическому значению. В логическом контексте все объекты являются true, всё просто. Существует лишь их числовое и строковое преобразование.

2. Числовое преобразование происходит, когда мы вычитаем объекты или применяем математические функции.

3. Преобразования к строке обычно происходит, привыводе на экран объектов при помощи alert(obj) и в подобных контекстах.

Можно реализовать свои преобразования к строкам и числам, используя специальные объектные методы.

## [Хинты](https://learn.javascript.ru/object-toprimitive" \l "hinty)

Существует три варианта преобразования типов, которые происходят в различных ситуациях. Они называются «хинтами», как описано в [спецификации](https://tc39.github.io/ecma262/#sec-toprimitive):

**"string"**

Для преобразования объекта к строке, когда мы выполняем операцию над объектом, которая ожидает строку, например alert:

// вывод

alert(obj);

// используем объект в качестве ключа

anotherObj[obj] = 123;

**"number"**

Для преобразования объекта к числу, в случае математических операций:

// явное преобразование

let num = Number(obj);

// математические (не считая бинарного плюса)

let n = +obj; // унарный плюс

let delta = date1 - date2;

// сравнения больше/меньше

let greater = user1 > user2;

Например, бинарный плюс + может работать как со строками (объединяя их в одну), так и с числами (складывая их). Поэтому, если бинарный плюс получает объект в качестве аргумента, он использует хинт "default" для его преобразования.

Также, если объект сравнивается с помощью == со строкой, числом или символом, тоже неясно, какое преобразование следует выполнить, поэтому используется хинт "default".

// бинарный плюс использует хинт "default"

let total = obj1 + obj2;

// obj == number использует хинт "default"

if (user == 1) { ... };

Операторы сравнения больше/меньше, такие как < >, также могут работать как со строками, так и с числами. Тем не менее, по историческим причинам, они используют хинт "number", а не "default".

Все встроенные объекты, за исключением одного (объект Date, который мы рассмотрим позже), реализуют "default" преобразование тем же способом, что и "number". И нам следует поступать так же.

**Чтобы выполнить преобразование, JavaScript пытается найти и вызвать три следующих метода объекта:**

1. Вызвать obj[Symbol.toPrimitive](hint) – метод с символьным ключом Symbol.toPrimitive (системный символ), если такой метод существует,
2. Иначе, если хинт равен "string"
   * попробовать вызвать obj.toString() или obj.valueOf(), смотря какой из них существует.
3. Иначе, если хинт равен "number" или "default"
   * попробовать вызвать obj.valueOf() или obj.toString(), смотря какой из них существует.

## [Symbol.toPrimitive](https://learn.javascript.ru/object-toprimitive" \l "symbol-toprimitive)

Есть встроенный символ с именем Symbol.toPrimitive, который используется для обозначения метода преобразования, вот так:

obj[Symbol.toPrimitive] = function(hint) {

// вот код для преобразования этого объекта в примитив

// он должен вернуть примитивное значение

// hint = чему-то из "string", "number", "default"

};

Если метод Symbol.toPrimitive существует, он используется для всех хинтов, и больше никаких методов не требуется.

Пример реализации:

let user = {

name: "John",

money: 1000,

[Symbol.toPrimitive](hint) {

alert(`hint: ${hint}`);

return hint == "string" ? `{name: "${this.name}"}` : this.money;

}

};

// демонстрация результатов преобразований:

alert(user); // hint: string -> {name: "John"}

alert(+user); // hint: number -> 1000

alert(user + 500); // hint: default -> 1500

## [toString/valueOf](https://learn.javascript.ru/object-toprimitive" \l "tostring-valueof)

Если нет Symbol.toPrimitive, тогда JavaScript пытается найти методы toString и valueOf:

* Для хинта "string": вызвать метод toString, а если он не существует или возвращает объект вместо примитивного значения, то valueOf (таким образом, toString имеет приоритет при строковом преобразовании).
* Для других хинтов: вызвать метод valueOf, а если он не существует или возвращает объект вместо примитивного значения, то toString (таким образом, valueOf имеет приоритет для математических операций).

Методы toString и valueOf берут своё начало с древних времён. Это не символы (символов тогда ещё не было), а скорее просто «обычные» методы со строковыми именами. Они предоставляют альтернативный «старомодный» способ реализации преобразования.

Эти методы должны возвращать примитивное значение. Если toString или valueOf возвращает объект, то он игнорируется (так же, как если бы метода не было).

По умолчанию обычный объект имеет следующие методы toString и valueOf:

* Метод toString возвращает строку "[object Object]".
* Метод valueOf возвращает сам объект. Т.к., он возвращает сам объект поэтому игнорируется. Почему возвращает сам объект? это по историческим причинам. Так что можно предположить, что его не существует.

let user = {name: "John"};

alert(user); // [object Object]

alert(user.valueOf() === user); // true

Пример, реализации объекта user, используя комбинацию toString и valueOf вместо Symbol.toPrimitive:

let user = {

name: "John",

money: 1000,

// для хинта равного "string"

toString() {

return `{name: "${this.name}"}`;

},

// для хинта равного "number" или "default"

valueOf() {

return this.money;

}

};

alert(user); // toString -> {name: "John"}

alert(+user); // valueOf -> 1000

alert(user + 500); // valueOf -> 1500

Получается то же поведение, что и в примере с Symbol.toPrimitive.

Довольно часто нужно единое «универсальное» место для обработки всех примитивных преобразований. В этом случае мы можем реализовать только toString:

let user = {

name: "John",

toString() {

return this.name;

}

};

alert(user); // toString -> John

alert(user + 500); // toString -> John500

В отсутствие Symbol.toPrimitive и valueOf, toString обработает все примитивные преобразования.

### [Преобразование может вернуть любой примитивный тип](https://learn.javascript.ru/object-toprimitive" \l "preobrazovanie-mozhet-vernut-lyuboy-primitivnyy-tip)

Важная вещь, которую следует знать обо всех методах преобразования примитивов, заключается в том, что они не обязательно возвращают подсказанный хинтом примитив.

Нет никакого контроля над тем, вернёт ли toString именно строку, или чтобы метод Symbol.toPrimitive возвращал именно число для хинта "number".

Единственное обязательное условие: эти методы должны возвращать примитив, а не объект.

**Историческая справка**

По историческим причинам, если toString или valueOf вернёт объект, то ошибки не будет, но такое значение будет проигнорировано (как если бы метода вообще не существовало). Это всё потому, что в древние времена в JavaScript не было хорошей концепции «ошибки».

А вот Symbol.toPrimitive уже «четче», этот метод *обязан* возвращать примитив, иначе будет ошибка.

## [Дальнейшие преобразования](https://learn.javascript.ru/object-toprimitive" \l "dalneyshie-preobrazovaniya)

Как мы уже знаем, многие операторы и функции выполняют преобразования типов, например, умножение \* преобразует операнды в числа.

Если мы передаём объект в качестве аргумента, то в вычислениях будут две стадии:

1. Объект преобразуется в примитив (с использованием правил, описанных выше).
2. Если необходимо для дальнейших вычислений, этот примитив преобразуется дальше.

Например:

let obj = {

// toString обрабатывает все преобразования в случае отсутствия других методов

toString() {

return "2";

}

};

alert(obj \* 2); // 4, объект был преобразован к примитиву "2", затем умножение сделало его числом

1. Умножение obj \* 2 сначала преобразует объект в примитив (это строка "2").
2. Затем "2" \* 2 становится 2 \* 2 (строка преобразуется в число).

А вот, к примеру, бинарный плюс в подобной ситуации соединил бы строки, так как он совсем не брезгует строк:

let obj = {

toString() {

return "2";

}

};

alert(obj + 2); // 22 ("2" + 2), преобразование к примитиву вернуло строку => конкатенация

На практике часто бывает достаточно реализовать только obj.toString() в качестве универсального метода для преобразований к строке, который должен возвращать удобочитаемое представление объекта для целей логирования или отладки.