Опциональная цепочка ?. — это безопасный способ доступа к свойствам вложенных объектов, в тех случаях когда какое-либо из промежуточных свойств не существует.

**[Проблема «несуществующего свойства»](https://learn.javascript.ru/optional-chaining" \l "problema-nesuschestvuyuschego-svoystva)**

Например, есть объекты user, которые содержат информацию о пользователях.

У большинства есть адреса в свойстве user.address с улицей user.address.street, но некоторые из них их не указали.

При попытке получить user.address.street, а пользователь окажется без адреса, вылетит ошибка:

let user = {}; // пользователь без свойства "address"

alert(user.address.street); // Ошибка!

Поскольку user.address имеет значение undefined, попытка получить user.address.street завершается ошибкой.

Во многих практических случаях предпочтительнее было бы получить undefined вместо ошибки (что означало бы «улицы нет»).

Например в веб-разработке мы можем получить объект, соответствующий элементу веб-страницы, с помощью специального вызова метода, такого как document.querySelector('.elem'), и он возвращает null, когда такого элемента нет. Если элемент не существует, мы получим сообщение об ошибке доступа к свойству .innerHTML у null. И в некоторых случаях, когда отсутствие элемента является нормальным, мы хотели бы избежать ошибки и просто принять html = null в качестве результата.

// document.querySelector('.elem') равен null, если элемента нет

let html = document.querySelector('.elem').innerHTML; // ошибка, если он равен null

Как можно это сделать?

1. Проверить значение с помощью if или условного оператора ?, прежде чем обращаться к его свойству, вот так:

let user = {};

alert(user.address ? user.address.street : undefined);

Это работает, тут нет ошибки… Но происхдит дублирование кода - "user.address" появляется дважды.

Для document.querySelector:

let html = document.querySelector('.elem') ? document.querySelector('.elem').innerHTML : null;

Как видно, поиск элемента document.querySelector('.elem') здесь вызывается дважды, что не очень хорошо.

Для более глубоко вложенных свойств это ещё ужаснее, поскольку потребуется больше повторений.

let user = {}; // у пользователя нет адреса

alert(user.address ? user.address.street ? user.address.street.name : null : null);

Использовать оператор &&:

let user = {}; // пользователь без адреса

alert( user.address && user.address.street && user.address.street.name ); // undefined (без ошибки)

Проход при помощи логического оператора И && через весь путь к свойству гарантирует, что все компоненты существуют (если нет, вычисление прекращается). Но имена свойств по-прежнему дублируются в коде.

Вот почему в язык была добавлена опциональная цепочка ?.. Чтобы решить эту проблему – раз и навсегда!

## [Опциональная цепочка](https://learn.javascript.ru/optional-chaining" \l "optsionalnaya-tsepochka)

Опциональная цепочка ?. останавливает вычисление и возвращает undefined, если значение перед ?. равно undefined или null.

Другими словами, value?.prop:

* работает как value.prop, если значение value существует,
* в противном случае (когда value равно undefined/null) он возвращает undefined.

вот пример с document.querySelector:

let html = document.querySelector('.elem')?.innerHTML; // будет undefined, если элемента нет

Считывание адреса с помощью user?.address работает, даже если объект user не существует:

let user = null;

alert( user?.address ); // undefined

?. делает необязательным значение перед ним, но не какое-либо последующее.

**Переменная перед ?. должна быть объявлена**

Переменная должна быть объявлена (к примеру, как let/const/var user или как параметр функции). Опциональная цепочка работает только с объявленными переменными.

Если переменной user вообще нет, то user?.anything приведёт к ошибке:

// ReferenceError: user is not defined

user?.address;

## [Сокращённое вычисление](https://learn.javascript.ru/optional-chaining" \l "sokraschyonnoe-vychislenie)

Как было сказано ранее, ?. немедленно останавливает вычисление, если левая часть не существует.

Так что если после ?. есть какие-то вызовы функций или операции, то они не произойдут.

Например:

let user = null;

let x = 0;

user?.sayHi(x++); // нет "user", поэтому выполнение не достигает вызова sayHi и x++

alert(x); // 0, значение не увеличилось

## [Другие варианты применения: ?.(), ?.[]](https://learn.javascript.ru/optional-chaining" \l "drugie-varianty-primeneniya)

Опциональная цепочка ?.  также работает с функциями и квадратными скобками.

?.()

Например, ?.() используется для вызова функции, которая может не существовать.

Например у некоторых пользователей есть метод admin, а у некоторых его нет:

let userAdmin = {

admin() {

alert("Я админ");

}

};

let userGuest = {};

userAdmin.admin?.(); // Я админ

userGuest.admin?.(); // ничего не произойдет (такого метода нет)

 В обеих строках сначала используем точку (userAdmin.admin), чтобы получить свойство admin, потому что предполагается, что объект userAdmin существует, так что читать из него безопасно.

Затем ?.() проверяет левую часть: если функция admin существует, то она запускается (это так для userAdmin). В противном случае (для userGuest) вычисление остановится без ошибок.

?.[]

Синтаксис ?.[] также работает, если мы хотим использовать скобки [] для доступа к свойствам вместо точки .. Как и в предыдущих случаях, он позволяет безопасно считывать свойство из объекта, который может не существовать.

let key = "firstName";

let user1 = {

firstName: "John"

};

let user2 = null;

alert( user1?.[key] ); // John

alert( user2?.[key] ); // undefined

Также мы можем использовать ?. с delete:

delete user?.name; // удаляет user.name если пользователь существует

**Мы можем использовать ?. для безопасного чтения и удаления, но не для записи**

Опциональная цепочка ?. не имеет смысла в левой части присваивания.

Например:

let user = null;

user?.name = "John"; // Ошибка, не работает

// то же самое что написать undefined = "John"

Синтаксис опциональной цепочки ?. имеет три формы:

1. obj?.prop – возвращает obj.prop если obj существует, в противном случае undefined.
2. obj?.[prop] – возвращает obj[prop] если obj существует, в противном случае undefined.
3. obj.method?.() – вызывает obj.method(), если obj.method существует, в противном случае возвращает undefined.

Как мы видим, все они просты и понятны в использовании. ?. проверяет левую часть на null/undefined и позволяет продолжить вычисление, если это не так.

Цепочка ?. позволяет безопасно получать доступ к вложенным свойствам.