ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ

JavaScript
без угадывания

12 разборов
неожиданных результатов

НАБЛЮДАЕМ
```
'4' + '7' → '47'
```
↓
ОБЪЯСНЯЕМ
```
typeof '4' → 'string'
```
↓
ПРОВЕРЯЕМ ИСПРАВЛЕНИЕ
```
Number('4') + Number('7') → 11
```

24 задания с полными решениями

Локальные примеры • Массивы • Объекты • Promise

ГЛЕБ ЗАЙЦЕВ

Глеб Зайцев

JavaScript без угадывания. 12 разборов неожиданных результатов

«Автор»
2026

Зайцев Г.

JavaScript без угадывания. 12 разборов неожиданных результатов / Г. Зайцев — «Автор», 2026

Почему число превратилось в строку, копия изменила оригинал, а ошибка прошла мимо catch? Практикум для тех, кто уже знаком с переменными, условиями, функциями, массивами и объектами JavaScript. В каждом из 12 разборов — конкретная задача, полный пример неожиданного поведения, пошаговое объяснение, исправление и два самостоятельных упражнения с решениями. Всего 48 автономных программ: ошибочные версии, исправления и ответы. Примеры проверены в Node.js 24.19.0 и работают локально, без сети и дополнительных пакетов. Разбираем преобразования, значения по умолчанию, массивы, ссылки, замыкания, this, JSON и Promise. Это не полный курс языка с нуля и не пособие по браузерной вёрстке.

Содержание

Глеб Зайцев JavaScript без угадывания. 12 разборов неожиданных результатов

Прежде чем менять код

Кому пригодится этот практикум

Вы написали короткую программу, она запускается, но результат не совпадает с ожиданием. В такой момент легко начать менять операторы наугад. Эта книга предлагает другой рабочий ритм: сначала назвать обещание программы, затем увидеть промежуточные значения, объяснить расхождение и только после этого исправить код. Результат каждого разбора — не набор запретов, а маленькая проверяемая модель поведения.

Предполагается, что вы уже умеете объявить переменную, написать if и функцию, обратиться к элементу массива и полю объекта. Новые для кейса конструкции объясняются рядом с примером. Если эти основы пока незнакомы, лучше сначала пройти вводный материал, а сюда вернуться для практики. Мы не строим сайт и не настраиваем сервер: Node.js нужен лишь как воспроизводимая среда выполнения JavaScript.

Двенадцать случаев разделены по смыслу ошибки. В первых четырёх прослеживаем значения и возвращаемые результаты. В следующих четырёх — поиск, общие ссылки, дубликаты и переменные замыканий. В последних четырёх — контекст вызова, JSON и асинхронные границы. Можно выбрать конкретный случай, но соседние главы иногда дают необходимое понятие; например, ссылки полезно понять до замыканий на объекты.

Как запустить один пример

Все программы проверены в Node.js v24.19.0. Для собственной работы выбирайте поддерживаемую LTS-ветку Node.js с официального сайта; закреплённая здесь версия описывает проверку книги, а не требование установить старый патч. Список официальных источников помещён в конце. Не нужно устанавливать npm-пакеты, запускать сервер или подключать рабочие учётные записи.

Создайте отдельную учебную папку и обычный текстовый файл с именем, указанным над листингом, например c05-fixed.mjs. Расширение .mjs выбирает режим ES-модуля. Перенесите только строки программы, без заголовка и ожидаемого вывода. Откройте терминал в этой папке и выполните node --unhandled-rejections=strict c05-fixed.mjs. Для другого листинга замените только имя файла. Каждый файл автономен: не склеивайте ошибочную и исправленную версии в одну.

Листинги используют встроенный модуль node:assert/strict. assert.equal сравнивает значения строго, assert.deepEqual проверяет структуру, assert.throws проверяет синхронное исключение, а assert.rejects — отклонение Promise, которое нужно дождаться. Успешная проверка ничего не печатает. Если она не выполнена, программа сообщает об AssertionError; это не тот ожидаемый учебный TypeError, который некоторые ошибочные версии сами перехватывают.

В книге обычные прямые кавычки, а не типографские. Сохраняйте окончания строк и не превращайте код в текстовый документ Word. Начальные отступы в электронной версии могут содержать неразрывные пробелы; для JavaScript вне строковых литералов это пробельные символы, но редактором их удобно заменить обычными пробелами. Не заменяйте содержимое строк и не добавляйте переносы внутри кавычек. При копировании из приложения сначала проверьте короткий листинг: конкретная читалка может переносить видимую строку иначе, чем исходный файл.

Программы работают с данными, прямо записанными в тексте. Они не читают личные файлы, не отправляют запросы, не выполняют платежи и не требуют паролей. Асинхронные случаи тоже локальные: Promise моделируют готовность результата без таймеров и сети. Не переносите учебную обработку ошибок в реальную систему без пересмотра её требований.

Сначала прогноз, затем запуск

Перед каждой ошибочной программой сформулируйте ожидаемый вывод своими словами. Затем просмотрите код и запишите прогноз по строкам. После запуска сравните не только итог, но и типы, тождество объектов, выбранную запись или место перехвата ошибки. Несовпадение прогноза — полезный материал: оно показывает конкретное предположение, которое нужно пересмотреть.

В исправлениях даны дополнительные проверки. Они не доказывают правильность на всех возможных входах, зато делают часть контракта видимой. Если в условии разрешены только плотные массивы обычных объектов, тестировать функцию как универсальный обработчик любых значений бессмысленно без новой постановки задачи. И наоборот: если функция обещает отвергать некорректный id, проверка этого отказа принадлежит её контракту.

После разбора решите два задания, не открывая ответы сразу. Первое может расширять структуру, второе — менять требование или способ представления результата. Ответы расположены отдельно и являются полными программами. Сравнивайте не названия переменных, а решения: что именно сохраняется, что заменяется, что признаётся отсутствием и кому принадлежит обработка ошибки.

Короткая входная проверка

Перед началом ответьте без запуска: чем строка с цифрой отличается от числа; когда функция начинает выполнять тело; означает ли новый массив, что все вложенные объекты тоже новые; является ли undefined объектом с пустыми полями? Если ответ не уверен, оставьте пометку и вернитесь к нему после соответствующего разбора.

Ориентир для проверки: тип влияет на смысл операции; тело обычной сохранённой функции выполняется при вызове; новый контейнер может хранить прежние ссылки; у undefined нельзя читать свойства как у обычного объекта. Это лишь исходные различия. Конкретные следствия и границы мы проверим в программах, а не примем как универсальные рецепты.

Навигация в электронной книге

В оглавлении есть отдельные разборы, условия, подсказки и решения с номерами вида 5.1. Встроенные ссылки помогают переходить между ними, если читалка поддерживает такие переходы. Если нажатие не переместило текст, используйте оглавление и тот же номер: оно является запасным способом навигации. Видимый номер страницы зависит от устройства и размера шрифта.

На узком экране увеличенный шрифт может переносить длинные строки кода. Отличайте визуальный перенос от новой строки исходника. Полные листинги намеренно короткие по ширине; для набора или проверки символов удобно читать их рядом с редактором на компьютере. Скриншот не заменяет текст программы.

Выбрать разбор

Двенадцать разборов

Разбор 1. Сумма двух текстовых ячеек

1. Задача и понятия

Отделить проверку числового импорта от сложения и получить количество, а не новую строку.

Предпосылки

Строковые и числовые литералы, вызов функции, условие и return; простой цикл for перебирает значения последовательно.

В мастерской настольных игр считают заготовки жетонов. В утренней ведомости записано '27', в дополнительной — '5'. Данные уже представлены литералами: никакой файл читать не нужно. Обе ячейки содержат текст, хотя человек видит в них количества. Итог должен быть числом 32, пригодным для дальнейшего расчёта, а не подписью '275'.

Договоримся о формате заранее: принимаем строки с десятичными цифрами 0–9; внешние пробельные символы и ведущие нули разрешены. После преобразования каждое количество лежит от 0 до 10000 включительно. Пустота, знак, дробь, экспонента, другая система счисления и нестроковый вход означают ошибку. Функция суммы возвращает null, если хотя бы одна ячейка непригодна. Частичного итога нет.

У оператора + здесь два возможных назначения. Для двух чисел это сложение. Если один из двух наших простых операндов — строка, получается соединение текстов. Кавычки в исходнике не декоративны: они выбирают строковое значение. Выражение '27' + 5 тоже даст строку. Поэтому преобразовать только одну ячейку недостаточно; проверять нужно обе стороны операции.

Number(text), вызванный без new, превращает допустимую числовую запись в число. Но он не знает правил ведомости: пустая строка превращается в ноль, а экспоненциальные и шестнадцатеричные записи тоже могут быть числовыми. Проверка диапазона не доказывает, что исходная запись была десятичной. Сначала проверяем написание, затем значение, и лишь потом выполняем предметное действие.

Метод trim() убирает внешние пробельные символы. Шаблон /^[0-9]+$/ после этой очистки требует непустую последовательность цифр: [0-9] обозначает цифру, + — одно или несколько повторений, ^ и $ задают края. Метод test сообщает true или false. Number.isSafeInteger проверяет, что получено безопасное целое число; отдельное сравнение устанавливает наш значительно меньший предел. null служит отдельным сигналом отказа, не совпадающим с допустимым нулём.

1. Ошибочная версия — c01-bug.mjs

```
const morning = '27';
const extra = '5';
const total = morning + extra;
```

```
console.log(total);
console.log(typeof total);
```

1. Что произошло

Ожидаемый вывод

```
275
string
```

Сначала переменные получают строки '27' и '5'. При вычислении + содержимое не распознаётся как количество жетонов: строка справа присоединяется к строке слева. Поэтому первая строка вывода похожа на число, но вторая прямо показывает тип string.

Ошибки выполнения нет: соединение строк допустимо. Расхождение обнаруживает контракт, требующий числовую сумму. Диагностический typeof полезен именно потому, что один внешний вид результата в консоли может ввести в заблуждение.

1. Исправленная версия — c01-fixed.mjs

```
import assert from 'node:assert/strict';

function readCount(raw) {
if (typeof raw !== 'string') return null;
const text = raw.trim();
if (!/^[0-9]+$/.test(text)) return null;
const value = Number(text);
if (!Number.isSafeInteger(value)) return null;
if (value > 10000) return null;
return value;
}

function sumCounts(left, right) {
const a = readCount(left);
const b = readCount(right);
if (a === null || b === null) return null;
return a + b;
}

const total = sumCounts('27', '5');
console.log(total);
console.log(typeof total);

assert.equal(total, 32);
assert.equal(sumCounts(' 004 ', '6'), 10);
assert.equal(sumCounts('0', '0'), 0);
assert.equal(sumCounts('10000', '10000'), 20000);
for (const bad of ['', ' ', '-1', '2.5', '1e2']) {
assert.equal(sumCounts(bad, '1'), null);
}
```

```
for (const bad of ['0x10', '10001', '2x', 7]) {
assert.equal(sumCounts('1', bad), null);
}
```

1. Почему работает исправление

Ожидаемый вывод

```
32
number
```

Сложение находится после двух успешных преобразований; в эту строку попадают только числа. Проверка === null не отбрасывает ноль. Верхняя граница суммы равна 20000, так что при заявленных входах вопрос огромных целых не возникает. Сохранять исходное написание в числовом результате мы не обещаем: '004' становится 4.

Проверки берут не только рабочую пару: они защищают разрешённые нули, внешние пробелы, верхний предел и разные причины отказа. При смене формата импорта сначала меняйте договорённость о записи. Механическое удаление всех нецифровых символов опасно: ошибочная ячейка '2x' не должна незаметно становиться двумя жетонами.

У количества есть формат записи и допустимый диапазон; проверьте оба условия до числового сложения.

Задание 1.1

Сверьте пакет заявок на бумажные конверты: ['18', '0', '7']. Верните число 25. Функция получает плотный массив — с элементом в каждой позиции — максимум из 100 строк. После удаления внешних пробельных символов каждая строка должна содержать одну или несколько цифр 0–9 и задавать целое число 0–10000. Ведущие нули разрешены; пустая запись, знаки, дроби, экспонента и другая система счисления запрещены. Пустой пакет даёт 0; одна плохая ячейка или превышение размера пакета отменяет весь результат: возвращается null. Не пропускайте ошибочные заявки.

Подсказка 1.1
Решение 1.1

Задание 1.2

Подготовьте номера карточек очереди из начального номера '008' и количества '3': Q-008, Q-009, Q-010, каждый на своей строке. Оба входа — проверяемые десятичные строки с внешними пробелами. Начало допускается от 0 до 999, количество — от 0 до 20. Нулевое количество даёт пустой массив. Плохая запись или выход последнего номера за 999 даёт null.

Подсказка 1.2
Решение 1.2
К списку разборов

Разбор 2. Ноль тоже настройка

2. Задача и понятия

Различить отсутствующее значение, допустимый ноль и ошибочную настройку перед применением умолчания.

Предпосылки

Числа, логические значения, поля объекта, условие и проверка строгого равенства.

Организатор кружка готовит текстовые карточки с заданиями. Настройка blankLines задаёт число пустых строк между заданиями: от 0 до 6, только целое число. Ноль выбран сознательно для компактной версии. Если настройку не передали или передали null, используются две строки. Сейчас программа печатает 2 даже для запроса 0; нужно сохранить выбор организатора.

Мы вычисляем только настройку, не создаём документ и не отправляем его в печать. В исправленном варианте функция возвращает выбранное число либо null при недопустимом значении. Строка '0', false, дробь, NaN и число вне диапазона не считаются отсутствием. Их нельзя молча заменять рабочим умолчанием.

Оператор || удобно читать как выбор по логической истинности, но не как проверку наличия. Он возвращает левое значение, если оно truthy, иначе правое. Среди falsy есть 0, false, пустая строка, NaN, null и undefined. В настройках первые три нередко означают полноценный пользовательский выбор. Один оператор не определяет их предметный смысл: его задаёт контракт поля.

Оператор ?? выполняет более узкий выбор: справа берётся запасное значение только при null или undefined слева. Для остальных значений возвращается левое. В частности, ноль остаётся нулём. Это оператор подстановки при отсутствии, а не очиститель неправильных данных. Он сохранит и отрицательное число, и NaN; поэтому после подстановки нужна отдельная проверка выбранного значения.

В функции ниже Number.isInteger принимает именно целые числа, без преобразования строк. Затем два сравнения ограничивают диапазон. Так получаются три разные ситуации: отсутствие получает умолчание, допустимое значение сохраняется, неправильное вызывает отказ. Операторы ?? и || нельзя напрямую смешивать в одном выражении без скобок; здесь они для одного решения вместе не нужны.

2. Ошибочная версия — c02-bug.mjs

```
function blankLines(value) {
return value || 2;
}

console.log(blankLines(0));
console.log(blankLines(undefined));
```

2. Что произошло

Ожидаемый вывод

```
2
2
```

В первом вызове значение 0 действительно передано, но для || оно falsy. Выражение возвращает правую часть 2. Во втором вызове undefined тоже falsy, и получается та же двойка. Разные пользовательские намерения превратились в одинаковый результат.

При ручном тесте только с отсутствующей настройкой дефект не виден: второй результат как раз правильный. Нужен пример, где значение присутствует, допустимо и при этом логически ложно. Здесь это компактный режим с нулём пустых строк.

2. Исправленная версия — c02-fixed.mjs

```
import assert from 'node:assert/strict';

function blankLines(value) {
const chosen = value ?? 2;
if (!Number.isInteger(chosen)) return null;
if (chosen < 0 || chosen > 6) return null;
return chosen;
}

console.log(blankLines(0));
console.log(blankLines(undefined));

assert.equal(blankLines(0), 0);
assert.equal(blankLines(5), 5);
assert.equal(blankLines(6), 6);
assert.equal(blankLines(null), 2);
assert.equal(blankLines(), 2);
for (const bad of [-1, 7, 0.5, '0', false, NaN]) {
assert.equal(blankLines(bad), null);
}
assert.equal(blankLines(Infinity), null);
```

2. Почему работает исправление

Ожидаемый вывод

```
0
2
```

Оператор выбирает кандидата, а проверки решают, можно ли использовать его как число строк. Это разные обязанности. Отрицательное значение не становится двойкой и не просачи-

вается дальше. Успешный assert ничего не печатает, поэтому строки вывода показывают только два интересующих нас режима.

Исправление подходит именно для договора, где null и undefined равнозначны отсутствию. Если null означал бы отдельную команду, например сброс сохранённых данных, эту ветку пришлось бы обработать явно. Нет общего правила «всегда заменять || на ??»: сначала перечислите допустимые значения конкретного поля.

Подстановка по отсутствию должна сохранять валидные falsy-значения и не маскировать ошибки диапазона.

Задание 2.1

Подготовьте настройки предпросмотра: {captions: false, title: ''}. Функция получает обычный объект; поля могут отсутствовать. Поле captions принимает только boolean; умолчание true. Поле title принимает любую строку, включая пустую; умолчание 'Preview'. Умолчание применяется только к null и undefined. Получив неподходящий тип хотя бы одного выбранного поля, функция возвращает null. Иначе она возвращает новый объект с двумя полями. Выведите false и затем пустой заголовок внутри квадратных скобок.

Подсказка 2.1
Решение 2.1

Задание 2.2

Для учебного стенда выберите число повторных попыток: сначала личное значение, затем групповое, затем 4. Отсутствуют только null и undefined. Первый присутствующий кандидат должен быть целым числом от 0 до 8; если он неправильный, верните null и не переходите к следующему. Ноль означает отсутствие повторов. Для личного undefined и группового 0 выведите 0.

Подсказка 2.2
Решение 2.2
К списку разборов

Разбор 3. Порядок чисел и порядок записи

3. Задача и понятия

Построить числовую сортировку для отчёта, не меняя массив в порядке поступления.

Предпосылки

Плотный массив — массив с элементом в каждой позиции; знакомы индексы, длина и вызов функции.

В клубе моделирования фиксируют длительности четырёх этапов: [8, 35, 120, 4] минут. Массив хранит порядок выполнения; рядом нужен список от короткого этапа к длинному. Правильный отчёт — 4, 8, 35, 120. Исходная последовательность должна остаться прежней, иначе исчезнет информация о ходе работы.

Принимаем плотные массивы целых чисел от 0 до 600 включительно. Ноль, повторы и пустой массив допустимы. Строки, NaN, Infinity, объекты и пропуски позиций находятся вне входного контракта. Здесь исследуем порядок и изменение массива, а не строим универсальный валидатор импортированных данных.

Вызов sort() без функции сравнения выбирает порядок по строковым представлениям элементов, сравнивая кодовые единицы UTF-16. Для наших положительных целых это видно уже по первым цифрам: запись '120' начинается с '1' и оказывается перед '35', '4' и '8'. При этом сами элементы не становятся строками в массиве: меняется порядок, а не числовой тип.

Чтобы задать другой порядок, sort получает функцию сравнения, или компаратор. В записи (a, b) => a - b два значения называются a и b, а результат выражения возвращается вызывающему методу. Отрицательный результат требует поставить a раньше b, положительный — позже, ноль означает равенство по выбранному критерию. Для ограниченных конечных чисел вычитание задаёт нужный числовой порядок.

Вторая независимая особенность sort — изменение того массива, на котором его вызвали. Возвращается ссылка на него же. Присваивание результата другой переменной не создаёт копию. Запись [...values] сначала создаёт новый массив из элементов values; сортируя эту копию, мы сохраняем исходную последовательность. Для данной задачи с числами этого достаточно; обещания о копировании вложенных объектов здесь нет.

3. Ошибочная версия — c03-bug.mjs

```
const minutes = [8, 35, 120, 4];
const report = minutes.sort();

console.log(report.join(', '));
console.log(minutes.join(', '));
console.log(report === minutes);
```

3. Что произошло

Ожидаемый вывод

```
120, 35, 4, 8
120, 35, 4, 8
true
```

Метод join(', ') соединяет элементы для компактного вывода; запятые относятся к представлению списка. Первая строка обнаруживает строковый порядок. Вторая показывает, что массив этапов тоже переставлен. Третья подтверждает: report и minutes обозначают один массив, а не два одинаковых по содержимому.

Если добавить только компаратор, первая строка исправится, но порядок записи этапов всё равно потеряется. Если сделать только копию, исходник сохранится, однако числовой отчёт останется неправильным. Требование состоит из двух частей, и каждую нужно проверить отдельно.

3. Исправленная версия — c03-fixed.mjs

```
import assert from 'node:assert/strict';

function orderedMinutes(values) {
const copy = [...values];
copy.sort((a, b) => a - b);
return copy;
}

const minutes = [8, 35, 120, 4];
const report = orderedMinutes(minutes);
console.log(report.join(', '));
console.log(minutes.join(', '));
console.log(report === minutes);

assert.deepEqual(report, [4, 8, 35, 120]);
assert.deepEqual(minutes, [8, 35, 120, 4]);
assert.notEqual(report, minutes);
assert.deepEqual(orderedMinutes([]), []);
assert.deepEqual(orderedMinutes([7]), [7]);
const bounds = [600, 0, 600, 0];
assert.deepEqual(orderedMinutes(bounds), [0, 0, 600, 600]);
assert.deepEqual(bounds, [600, 0, 600, 0]);
const empty = [];
assert.notEqual(orderedMinutes(empty), empty);
```

3. Почему работает исправление

Ожидаемый вывод

```
4, 8, 35, 120
8, 35, 120, 4
false
```

Копия отделяет порядок отчёта от порядка записи, а компаратор отделяет величину числа от его написания. Повторяющиеся длительности остаются отдельными наблюдениями. Для проверки сохранения исходника используется заранее известный массив, а не другая ссылка на уже изменённый объект.

Тест пустого массива дополнительно требует новую ссылку: даже отсутствие этапов не отменяет обещания вернуть отдельный список. Не привязывайте проверки к числу или последовательности вызовов компаратора: программа задаёт итоговый порядок, но не выбирает внутренний алгоритм движка.

У сортировки отчёта два решения: критерий сравнения и массив, который разрешено менять.

Задание 3.1

Из замеров [28, 6, 103, 6] выберите два самых коротких времени. Верните [6, 6]: одинаковые результаты принадлежат разным попыткам и не удаляются. Вход — плотный массив целых минут 0–600. При одном элементе верните один, при пустом входе — пустой массив. Исходную последовательность сохраните. Выведите выбранные значения и затем исходник через запятую с пробелом.

Подсказка 3.1
Решение 3.1

Задание 3.2

Посчитайте медиану длительностей [9, 130, 7, 16], сохранив исходный порядок. После числовой сортировки при нечётной длине берётся центральное значение, при чётной — среднее двух центральных. Для этих данных ответ 12.5. Плотный вход содержит целые минуты 0–600; пустой массив даёт null. Выведите медиану и затем исходные значения через запятую с пробелом.

Подсказка 3.2
Решение 3.2
К списку разборов

Разбор 4. Вычислено ещё не значит возвращено

4. Задача и понятия

Получить новый массив из результатов callback и проверить возврат на каждом пути выполнения.

Предпосылки

Массивы, функции и return; стрелочная запись функции знакома по компаратору предыдущего разбора.

В шкафу кружка роботов нужны короткие этикетки для отделений. Исходные подписи [' drill ', 'spare cord'] должны превратиться в ['DRILL', 'SPARE CORD']: внешние пробелы убираются, латинские буквы становятся прописными. Количество и порядок подписей сохраняются. Исходный массив нужен для сверки с черновиком и не изменяется.

Вход — плотный массив строк с обычной латиницей, цифрами и пробелами. Пустой массив, пустая строка и строка из пробелов разрешены; последние две после очистки дают пустую подпись, но остаются на своих местах. Мы не измеряем длину сложных символов Unicode, не подставляем объекты вместо строк и не рассматриваем массивы с пропусками позиций.

Метод map вызывает переданную функцию для каждого элемента нашего плотного массива и собирает именно возвращённые ею значения в новый массив. Переданную функцию называют callback: её вызывает метод, а не отдельная строка нашего кода. Достаточно назвать первый параметр name, чтобы работать с текущей подписью. Сам факт вызова trim или toUpperCase ещё не сообщает map, какое значение записать в результат.

У стрелочной функции два полезных для этой задачи вида тела. После стрелки можно поставить одно выражение: его значение возвращается автоматически. Если поставить фигурные скобки, получится блок команд. Из такого блока нужное значение следует передать через return. Дойти до конца блока без return — значит вернуть undefined. JavaScript не выбирает последнее вычисленное выражение вместо автора.

Конец ознакомительного фрагмента.