Оглавление

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте *www.dmkpress.com*, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу *dmkpress@gmail.com*; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу *http://dmkpress.com/authors/publish_book/* или напишите в издательство по адресу *dmkpress@gmail.com*.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу *dmkpress@gmail.com*, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу *dmkpress@gmail.com*.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Благодарности

Спасибо, что решились на это приключение вместе со мной. Книга никогда не пишется в одиночку, это результат поддержки, ободрения и руководства со стороны многих людей.

Я выражаю признательность всему коллективу издательства The Pragmatic Programmers, его опыт помог превратить задумку в реальность. Отдельное спасибо Маргарет Элдридж за тщательное редактирование и внимание к деталям, что положительно сказалось на качестве материала и ясности изложения.

Премного благодарен экспертам Майклу Фацио, Энди Лестеру и Дэниэлу Поси за бесценные рецензии на книгу до ее отправки в печать. Их опыт разработчиков значительно улучшил качество кода.

Я также выражаю благодарность Рэндаллу Кутнику, Лукасу Матису и Маркусу С. Зарра. Они не только нашли время просмотреть книгу и указать на ошибки, но также не поскупились на слова ободрения и положительной оценки.

Предисловие

Язык JavaScript давно известен своими причудами и странностями. Эти особенности зачастую объясняются в том числе эволюцией и необходимостью поддерживать обратную совместимость. Упомянем, например, знаменитую особенность – приведение типов: JavaScript многое прощает в отношении типов данных, что может стать причиной неожиданного поведения в случае смешения типов.

Разработчикам важно знать об этих чертах JavaScript, чтобы писать более надежный и предсказуемый код. По счастью, стандарт ECMAScript, определяющий JavaScript, развивается, и многие из этих странностей так или иначе разрешаются, что ведет к более предсказуемому поведению, а заодно привносит в язык дополнительные возможности.

В этой книге мы приглашаем совершить путешествие по тщательно отобранному собранию интригующих загадок JavaScript, призванному сорвать пелену тайны с тонких нюансов и особенностей, которые определяют самую сущность языка. На этих страницах найдет что-то полезное для себя каждый: и умудренный ветеран, желающий отточить свои навыки, и амбициозный новичок, жаждущий карьерного роста.

В начале каждой главы я выкатываю небольшую программку на JavaScript и предлагаю угадать, что она напечатает. Но не отчаивайтесь, я вас не брошу. Поразмыслив над задачей, запустите код и полюбуйтесь на чудо! А затем прочитайте объяснение и поднимите свои знания JavaScript на новый уровень.

Об авторе

Фараз К. Келхини – автор книг «Modern Asynchronous JavaScript» и «Text Processing with JavaScript». Прекрасно разбираясь в языке JavaScript и его запутанном API, Фараз напитал это путешествие страстью к новаторским идеям, которые помогут усовершенствовать практику кодирования, не забывая о решениях, в которых творчество органично сочетается с функциональностью.

О вас

Прежде чем пускаться в это увлекательное путешествие, отметим, что хотя бы поверхностное знакомство с JavaScript позволит гораздо лучше понять обсуждаемые в книге вещи. Если вы только начинаете изучать JavaScript и никогда раньше не имели случая познакомиться с его чудесами, то я рекомендую сначала потратить некоторое время на освоение основ – получите удовольствие!

О коде

Примеры кода намеренно краткие и призваны продемонстрировать основную суть каждой задачи. Не стесняйтесь выполнять их на консоли браузера, но предварительно обновите браузер до последней версии.

Для доступа к коду зайдите на сайт Pragmatic Bookshelf[1]. Там вы сможете оставить отзыв, сообщить о моих ошибках, получить актуальную информацию и присоединиться к обсуждению книги в форуме.

[1] *https://www.pragprog.com/titles/fkjsbrain*.

Часть 1
ЗАДАЧИ

Задача 1

Ваш код нуждается в подтяжке

your_code_deserves_a_lift/your_code_deserves_a_lift.js

```
let temp = 25;

function displayTemperature() {
  console.log(`Current temperature: ${temp} °C`);
}

function forecastTemperature() {
  console.log(`Expected temperature: ${temp} °C`);
  var temp = 28;
}

displayTemperature();
forecastTemperature();
```

Угадайте результат

Попробуйте угадать результат, не переворачивая страницу.

Вы, наверное, ожидали чего-то такого:

```
Current temperature: 25 °C
Expected temperature: 25 °C
```

Но на самом деле печатается:

```
Current temperature: 25 °C
Expected temperature: undefined °C
```

Обсуждение

В JavaScript все объявления подлежат поднятию. Это относится к объявлениям `var`, `let`, `const`, `function`, `function*` и `class`. Поднятие означает автоматическое перемещение объявлений в начало области видимости. Но инициализация откладывается до момента, когда поток выполнения дойдет до строки, в которой имело место поднятие. Это происходит еще до начала выполнения кода и помогает JavaScript обрабатывать ссылки на переменные и функции.

Рассмотрим следующий код:

```
// Создать переменную
var a = 10;
```

JavaScript обрабатывает этот код следующим образом:

```
var a;
// Создать переменную
a = 10;
```

Когда мы объявляем переменную с помощью ключевого слова `var`, объявление перемещается (или «поднимается») в начало объемлющей функции или глобальной области видимости. Но присваивание (инициализация) остается на прежнем месте. Это значит, что мы можем ссылаться на переменную еще до того, как она объявлена, и никакой ошибки не произойдет. Но значение будет не определено (равно `undefined`), пока не произойдет присваивания.

Такое поведение может приводить к неожиданным результатам при работе с функциями. Любая переменная, объявленная внутри функции, будет поднята наверх. Следовательно, если существует переменная с таким же именем в глобальной области видимости, внутри функции она будет скрыта. Например:

```
var a = 10;

function fn() {
  console.log(a); // → не определена
  var a = 20;

  console.log(a); // → 20
}

fn();
```

Таким образом, глядя на первую строку этой функции, можно подумать, что она напечатает `10` и `20`. Но JavaScript обрабатывает код иначе:

```
var a = 10;

function fn() {
  var a;
  console.log(a); // → не определена
  a = 20;
  console.log(a); // → 20
}

fn();
```

Чтобы избежать сюрпризов, в версии ES2015 придумали решение: ключевое слово `let` для объявления переменных. Переменные, объявленные с помощью `let`, по-прежнему поднимаются, но теперь они напоминают о себе. Попытавшись использовать необъявленную переменную, мы получим предостережение и тем самым избежим неожиданных проблем:

```
function fn() {
  console.log(a); // → не определена
  console.log(b); // → ReferenceError: b не определена

  var a = 20;
  let b = 20

  console.log(a);
}

fn();
```

Итак, если переменная объявлена с помощью `var`, то, попытавшись воспользоваться ею до момента объявления, мы увидим, что она равна `undefined`. С другой стороны, проделав такой трюк с переменной, объявленной с помощью `let`, мы получим сообщение об ошибке `ReferenceError`. И то же самое произойдет, если попытаться воспользоваться константой (`const`) или классом (`class`) до момента их объявления.

Сообщество JavaScript придумало термин *временная мертвая зона* (temporal dead zone – TDZ) для описания того, почему нельзя обращаться к величинам, объявленным с помощью `let` или `const`, до момента объявления. Такие объявления поднимаются точно так же, как объявления `var` и `function`, но имеется промежуток времени между входом в область видимости и точкой объявления, в течение которого доступ к ним запрещен. Этот промежуток и называется временной мертвой зоной.

Любая попытка обратиться к переменным, находящимся в TDZ, приводит к ошибке `ReferenceError`. Как только поток выполнения доходит до объявления, переменная удаляется из TDZ, и доступ к ней становится возможен. Цель TDZ – помочь вам находить ошибки в коде. Доступ к еще не объявленной переменной редко бывает преднамеренным.

Время жизни переменной

Переменная, объявленная внутри функции, существует на всем протяжении выполнения этой функции. После того как выполнение заканчивается, все локальные переменные автоматически удаляются. С другой стороны, глобальные переменные остаются в памяти до тех пор, пока не завершится программа или не будет закрыта веб-страница.

Важно помнить, что TDZ относится только к тому блоку кода, в котором переменная объявлена. Попытка доступа к переменной вне ее области видимости не приводит к ошибке, потому что переменная не находится в TDZ:

```
console.log(a); // → не определена

{
  console.log(a); // → ReferenceError
  let a = 10;
}

var a = 20;
```

Поднятие функций

Объявления функций, как и переменных, поднимаются в начало объемлющей области видимости. Но поднимаются целиком, включая имя и тело функции. Это значит, что функцию, объявленную с помощью `function`, можно вызывать еще до точки ее объявления в коде:

```
fn1(); // → Hello!

function fn1() {
 console.log( "Hello!" );
}
```

С другой стороны, функциональные выражения так не поднимаются. Поднимается только объявление переменной, а присваивание функции производится в точке объявления переменной:

```
fn2(); // → TypeError: fn2 не является функцией

var fn2 = function () {
 console.log( "Hello!" );
};
```

В этом случае переменной `fn2` первоначально присваивается значение `undefined`, поэтому невозможно вызвать ее как функцию до тех пор, пока не будет выполнено присваивание.

Итак, запомните, что объявлять переменные лучше всего в начале области видимости, которой они принадлежат. Тогда вы не столкнетесь по дороге ни с какими сюрпризами. И возьмите в привычку использовать `let` и `const` вместо `var`, потому что они помогают предотвратить потенциальные проблемы, например поднятие и непреднамеренное повторное объявление переменной.

Для дополнительного чтения

Поднятие в JavaScript
https://developer.mozilla.org/en-US/docs/Glossary/Hoisting.

Предложение var
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var.

Объявление let
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let.

Объявление const
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const.