16 - Введение в ES6 (let, const, шаблонные строки)

Недавно было введено новое ключевое слово let вместо var из JavaScript, с которым вы знакомы. Ключевое слово let - это просто новый способ задания переменной в JavaScript. Мы обсудим детали позже, сейчас же просто знайте, что много проблем в Javascript можно избежать, используя let. Поэтому вы должны использовать его вместо var, где это возможно.

1. Объявления переменных (variable declarations)

let и const - относительно новые типы объявления переменных в JavaScript. Как мы упомянули ранее, let похож на var в некотором смысле, но позволяет пользователям избежать некоторые из общих ошибок, с которыми сталкиваются в JavaScript. const это расширение let, которое предотвращает переопределение переменных.

Далее мы подробнее расскажем об этих новых объявлениях переменных и объясним, почему они более предпочтительны, чем var.

Правила области видимости (Scoping). Объявление var имеет несколько странных правил области видимости для тех, кто использует другие языки программирования. Посмотрите не следующий пример:

```
function f(shouldInitialize) {
      if (shouldInitialize) {
            var x = 10;
      }
      return x;
}
f(true);  // returns '10'
f(false); // returns 'undefined'
```

Переменная x была объявлена внутри блока if, и мы можем получить к ней доступ вне этого блока. Это потому что объявления var доступны где бы то ни было внутри содержащей их функции, модуля, пространства имен(namespace) или же глобальной области видимости несмотря на блок, в котором они содержатся.

Эти правила области видимости могут вызвать несколько типов ошибок. Одна из раздражающих проблем - это то, что не является ошибкой объявление переменной несколько раз:

```
function sumMatrix(matrix) {
      var sum = 0;
      for (var i = 0; i < matrix.length; i++) {
            var currentRow = matrix[i];
            for (var i = 0; i < currentRow.length; i++) {
                  sum += currentRow[i];
            }
      }
      return sum;
}
```

Скорее всего несложно заметить, что внутренний цикл for случайно перезапишет переменную i, потому что i имеет области видимости внутри функции sumMatrix. Опытные разработчики знают, что похожие ошибки проскальзывают при code review и могут быть причиной бесконечных циклов.

Variable capturing quirks. Попробуйте быстро догадаться, какой будет вывод у этого кода:

```
for (var i = 0; i < 10; i++) {
      setTimeout(function() {console.log(i); }, 100 * i);
}
```

Для тех, кто не знаком, setTimeout пытается выполнить функцию после указанного количества миллисекунд (при этом ожидая, пока какой-либо другой код прекратит выполняться)

Давайте рассмотрим это в контексте нашего примера. setTimeout запустит функцию через несколько миллисекунд, после завершения цикла for. К моменту, когда цикл for закончит выполнение, i будет равняться 10. Поэтому каждый раз, когда отложенная функция будет вызвана, она возвратит 10!

Самый простой способ решить проблему - использовать **немедленный запуск анонимной функции**, чтобы захватить i на каждой итерации:

```
for (var i = 0; i < 10; i++) {
      // capture the current state of 'i'
      // by invoking a function with its current value
      (function(i) {
            setTimeout(function() { console.log(i); }, 100 * i);
      })(i);
}
```

Этот странно выглядящий шаблон на самом деле не редок.

Объявления let. Сейчас мы уже понимаем, что var имеет некоторые проблемы, именно поэтому появился новый способ объявления переменных let. Они записываются точно также, как и объявления var.

```
let hello = "Hello!";
```

Ключевое различие не в синтаксисе, а в семантике, в которую мы сейчас погрузимся.

Блочная область видимости. Когда переменная объявляется с использованием **let**, она используется в режиме **блочной области видимости**. В отличие от переменных, объявленных с помощью var, чьи области видимости распространяются на всю функцию, в которой они находятся, переменные блочной области видимости невидимы вне их ближайшего блока или же цикла for.

```
function f(input) {
      let a = 100;
      if (input) {
            // Здесь мы видим переменную 'a'
            let b = a + 1;
            return b;
      }
// Ошибка: 'b' не существует в этом блоке
return b;
}
```

Здесь мы имеем две локальные переменные a и b. Область видимости a ограничена телом функции f, в то время как область b ограничена блоком условия if.

Другое свойство переменных блочной области видимости - к ним **нельзя обратиться перед тем, как они были объявлены**. При том, что переменные блочной области видимости представлены везде в своем блоке, в каждой точке до их объявления находится мертвая зона. Это просто такой способ сказать, что вы не можете получить к ним доступ до утверждения let.

Повторное объявление и экранирование. В случае объявлений var не имеет значения, как много раз вы объявляете одну и ту же переменную. Вы всегда получите одну.

```
function f(x) {
      var x;
      var x;

      if (true) {
            var x;
      }
}
```

В примере выше все объявления x на самом деле указывают на одну и ту же x, и это вполне допустимо. Это часто является источником багов. Поэтому хорошо, что объявления let этого не позволяют.

```
let x = 10;
let x = 20; // Ошибка: нельзя переопределить 'x' в одной области видимости
```

Переменные не обязательно должны обе быть с блочной областью видимости, чтобы компилятором была указана ошибка.

```
function f(x) {
      let x = 100; // ошибка: пересекается с параметром функции
}
function g() {
      let x = 100;
      var x = 100; // ошибка: нельзя два раза объявить 'x'
}
```

Это не значит, что переменная с блочной областью видимости не может быть объявлена с переменной с областью видимости в той же функции. Переменная с блочной областью просто должна быть объявлена в своем блоке

```
function f(condition, x) {
      if (condition) {
            let x = 100;
            return x;
      }
      return x;
}
f(false, 0); // returns 0
f(true, 0);  // returns 100
```

Способ введения нового имени во вложенной области называется сокрытием. Это своего рода меч с двумя лезвиями, т.к. он может ввести некоторые баги,

также как и избавить от других. Например, представьте, как мы могли бы переписать функцию sumMatrix, используя переменные let.

```
function sumMatrix(matrix) {
      let sum = 0;
      for (let i = 0; i < matrix.length; i++) {
            var currentRow = matrix[i];
            for (let i = 0; i < currentRow.length; i++) {
                  sum += currentRow[i];
            }
      }
return sum;
}
```

Эта версия цикла делает суммирование корректно, потому что i внутреннего цикла перекрывает i внешнего.

Такое **сокрытие нужно обычно избегать,** чтобы код был чище. Но в некоторых сценариях такой способ может идеально подходить для решения задачи. Вы должны использовать лучшее решение на ваше усмотрение.

Замыкание переменных с блочной областью видимости. Когда мы впервые коснулись замыкания переменных с объявлением var, мы коротко рассмотрели, как переменные ведут себя при замыкании. Чтобы лучше понимать суть, представьте себе, что каждый раз, когда появляется новая область видимости, она создает свою "среду" для переменных. Эта среда и ее захваченные извне переменные могут существовать даже после того, как все выражения внутри области видимости завершили свое выполнение.

```
function theCityThatAlwaysSleeps() {
      let getCity;

      if (true) {
            let city = "Seattle";
            getCity = function() {
                  return city;
            }
      }
      return getCity();
}
```

Из-за того, что мы захватили переменную city из ее среды, мы все еще можем получить к ней доступ, несмотря на тот факт, что блок if закончил выполнение. Вспомните наш предыдущий пример с setTimeout. Мы закончили на необходимости использовать IIFE, чтобы захватить состояние переменной для каждой итерации цикла for. В результате мы каждый раз создавали новую среду переменных для наших захваченных. Это доставляло немного боли, но, к счастью, нам не потребуется делать это снова.

Объявления let ведут себя совсем иначе, когда являются частью цикла. Вместо того, чтобы вводить новую среду для цикла, они вводят новую область видимости для каждой итерации. Так как это то, что мы делали с нашим IIFE, мы можем изменить наш старый пример setTimeout, используя объявления let.

```
for (let i = 0; i < 10 ; i++) {
      setTimeout(function() {console.log(i); }, 100 * i);
}
```

Объявления const. Объявления const - это еще один способ объявления переменных.

```
const numLivesForCat = 9;
```

Они такие же как и let, только, согласно их названию, их значение не может быть изменено после того, как им однажды уже присвоили значение. Другими словами, к ним применимы все правила области видимости let, но вы не можете их переназначить. Значение, с которым они связаны, является неизменным.

```
const numLivesForCat = 9;
const kitty = {
      name: "Aurora",
      numLives: numLivesForCat,
}

// О ш и б к а
kitty = {
      name: "Danielle",
      numLives: numLivesForCat
};

// В с е  х о р о ш о
kitty.name = "Rory";
kitty.name = "Kitty";
kitty.name = "Cat";
kitty.numLives--;
```

Несмотря на то, что переменная была объявлена как const, ее внутреннее состояние все еще может быть изменено.

let или const? У нас есть два способа объявления с похожими правилами их области видимости, поэтому сам собой напрашивается вопрос о том, какой использовать. Ответ будет таким же, как и на большинство широких вопросов: это зависит от обстоятельств.

Применяя **принцип наименьшего уровня привилегий**, все объявления переменных, которые вы в дальнейшем не планируете менять, должны использовать const. Объясняется это тем, что если переменная не должна изменять свое значение, другие разработчики, которые работают над тем же кодом, не должны иметь возможность записи в объект. Это должно быть позволено только в случае реальной необходимости переназначения переменной. Использование const делает код более предсказуемым и понятным при объяснении потока данных.