Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 20**

**Инструкционно-технологическая карта**

Тема: Ключевое слово «var». Блочная область видимости.

Цель: научиться использовать ключевое слово var, изучить блочную область видимости

Время выполнения: 2 часа

# Краткие теоретические сведения

Существуют ознакомились с тремя способами объявления объектов:

* let;
* const;
* var;

**let** и **const** ведут себя одинаково по отношению к лексическому окружению, области видимости.

Но **var** – это совершенно другой зверь, берущий своё начало с давних времён. Обычно var не используется в современных скриптах, но всё ещё может скрываться в старых.

На первый взгляд, поведение **var** похоже на **let**. Например, объявление переменной:

function sayHi() {

    var phrase = "Привет"; // локальная переменная, "var" вместо "let"

    alert(phrase); // Привет

}

sayHi();

alert(phrase); // Ошибка: phrase не определена

## Для «var» не существует блочной области видимости

Область видимости переменных var ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока.

Например:

if (true) {

    var test = true; // используем var вместо let

}

alert(test); // true, переменная существует вне блока if

Так как **var** игнорирует блоки, получена глобальная переменную **test**.

А если бы использовался **let** **test** вместо **var** **test**, тогда переменная была бы видна только внутри **if**:

if (true) {

    let test = true; // используем let

}

alert(test); // Error: test is not defined

Аналогично для циклов: var не может быть блочной или локальной внутри цикла:

for (var i = 0; i < 10; i++) {

    // ...

}

alert(i); // 10, переменная i доступна вне цикла, т.к. является глобальной переменной

Если блок кода находится внутри функции, то **var** становится локальной переменной в этой функции:

function sayHi() {

    if (true) {

        var phrase = "Привет";

    }

    alert(phrase); // срабатывает и выводит "Привет"

}

sayHi();

alert(phrase); // Ошибка: phrase не определена (видна в консоли разработчика)

Как можно увидеть, **var** выходит за пределы блоков **if**, **for** и подобных. Это происходит потому, что на заре развития JavaScript блоки кода не имели лексического окружения. Поэтому можно сказать, что **var** – это пережиток прошлого.

## «var» допускает повторное объявление

Если в блоке кода дважды объявить одну и ту же переменную let, будет ошибка:

let user;

let user; // SyntaxError: 'user' has already been declared

Используя var, можно переобъявлять переменную сколько угодно раз. Повторные **var** игнорируются:

var user = "Pete";

var user; // ничего не делает, переменная объявлена раньше

// ...нет ошибки

alert(user); // Pete

Если дополнительно присвоить значение, то переменная примет новое значение:

var user = "Pete";

var user = "John";

alert(user); // John

## «var» обрабатываются в начале запуска функции

Объявления переменных **var** обрабатываются в начале выполнения функции (или запуска скрипта, если переменная является глобальной).

Другими словами, переменные **var** считаются объявленными с самого начала исполнения функции вне зависимости от того, в каком месте функции реально находятся их объявления (при условии, что они не находятся во вложенной функции).

Т.е. этот код:

function sayHi() {

    phrase = "Привет";

    alert(phrase);

    var phrase;

}

sayHi();

Технически полностью эквивалентен следующему (объявление переменной **var** **phrase** перемещено в начало функции):

function sayHi() {

    var phrase;

    phrase = "Привет";

    alert(phrase);

}

sayHi();

…И даже коду ниже (блочная область видимости игнорируется):

function sayHi() {

    phrase = "Привет"; // (\*)

    if (false) {

        var phrase;

    }

    alert(phrase);

}

sayHi();

Это поведение называется «**hoisting**» (всплытие, поднятие), потому что все объявления переменных var «всплывают» в самый верх функции.

В примере выше **if (false)** условие никогда не выполнится. Но это никаким образом не препятствует созданию переменной var phrase, которая находится внутри него, поскольку объявления var «всплывают» в начало функции. Т.е. в момент присвоения значения (\*) переменная уже существует.

#### Объявления переменных «всплывают», но присваивания значений – нет.

Это проще всего продемонстрировать на примере:

function sayHi() {

    alert(phrase);

    var phrase = "Привет";

}

sayHi();

Строка **var phrase = "Привет"** состоит из двух действий:

1. Объявление переменной var
2. Присвоение значения в переменную =.

Объявление переменной обрабатывается в начале выполнения функции («всплывает»), однако присвоение значения всегда происходит в той строке кода, где оно указано. Т.е. код выполняется по следующему сценарию:

function sayHi() {

    var phrase; // объявление переменной срабатывает вначале...

    alert(phrase); // undefined

    phrase = "Привет"; // ...присвоение - в момент, когда исполнится данная строка кода.

}

sayHi();

Поскольку все объявления переменных **var** обрабатываются в начале функции, можно ссылаться на них в любом месте. Однако, переменные имеют значение **undefined** до строки с присвоением значения.

В обоих примерах выше вызов **alert** происходил без ошибки, потому что переменная **phrase** уже существовала. Но её значение ещё не было присвоено, поэтому возвращался **undefined**.

## Области видимости.

В JavaScript существуют три области видимости:

1. Лексическая.
2. Функциональная.
3. Блочная.

**Лексическая** область видимости заключается в том, что область видимости определяется после компиляции, и когда интерпретатору надо найти объявление переменной или функции, сначала он смотрит в текущей области видимости, но, если найти то, что ему нужно, не удаётся, он переходит в родительскую область видимости, продолжая поиск по тому же принципу. Самый высокий уровень, на который он может перейти, называется глобальной областью видимости.

**Функциональная** область видимости заключается в том, что переменная видна только внутри функции, в которой она создана.

**Блочная** область видимости похожа на функциональную, но она ограничена не функцией, а блоком кода, в котором была объявлена переменная, например, внутри цикла.

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Контрольные вопросы

1. Какие основные отличия между **var** и **let/const.**
2. Опишите области видимости переменных.

# Задания

Выполнить примеры, описанные выше, объяснить особенности работы **var.**

# Литература

**Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.

**Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.

**Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А.В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич