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 № 8**

**Инструкционно-технологическая карта**

Тема: Функции. Локальные и внешние переменные.

Цель: Научиться разрабатывать скрипты с применением функций.

Время выполнения: 2 часа

# 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.

# Теоретические сведения

## Объявление функции

Для создания функций можно использовать объявление функции.

Пример объявления функции:

function showMessage() {

    alert( 'Всем привет!' );

  }

Вначале идёт ключевое слово **function**, после него имя функции, затем список параметров в круглых скобках через запятую (в вышеприведённом примере он пустой) и, наконец, код функции, также называемый «телом функции», внутри фигурных скобок.

function имя(параметры) {

    ...тело...

  }

Функция может быть вызвана по её имени: showMessage().

Например:

function showMessage() {

    alert( 'Всем привет!' );

  }

  showMessage();

  showMessage();

Вызов **showMessage()** выполняет код функции. Здесь сообщение выведется дважды.

Этот пример явно демонстрирует одно из главных предназначений функций: избавление от дублирования кода.

Если понадобится поменять сообщение или способ его вывода – достаточно изменить его в одном месте: в функции, которая его выводит.

## Локальные переменные

Переменные, объявленные внутри функции, видны только внутри этой функции.

Например:

function showMessage() {

    let message = "Привет, я JavaScript!"; // локальная переменная

   alert( message );

  }

  showMessage(); // Привет, я JavaScript!

  alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри функции

## Внешние переменные

У функции есть доступ к внешним переменным, например:

let userName = 'Вася';

function showMessage() {

  let message = 'Привет, ' + userName;

  alert(message);

}

showMessage(); // Привет, Вася

Функция обладает полным доступом к внешним переменным и может изменять их значение.

Например:

let userName = 'Вася';

function showMessage() {

  userName = "Петя"; // (1) изменяем значение внешней переменной

  let message = 'Привет, ' + userName;

  alert(message);

}

alert( userName ); // Вася перед вызовом функции

showMessage();

alert( userName ); // Петя, значение внешней переменной было изменено функцией

Внешняя переменная используется, только если внутри функции нет такой локальной.

Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю. Например, в коде ниже функция использует локальную переменную **userName**. Внешняя будет проигнорирована:

let userName = 'Вася';

function showMessage() {

  let userName = "Петя"; // объявляем локальную переменную

  let message = 'Привет, ' + userName; // Петя

  alert(message);

}

// функция создаст и будет использовать свою собственную локальную переменную userName

showMessage();

alert( userName ); // Вася, не изменилась, функция не трогала внешнюю переменную

## Параметры

Внутрь функции можно передать любую информацию, используя параметры (также называемые аргументами функции).

В нижеприведенном примере функции передаются два параметра: from и text.

function showMessage(from, text) { // аргументы: from, text

    alert(from + ': ' + text);

  }

  showMessage('Аня', 'Привет!'); // Аня: Привет! (\*)

  showMessage('Аня', "Как дела?"); // Аня: Как дела? (\*\*)

Когда функция вызывается в строках (\*) и (\*\*), переданные значения копируются в локальные переменные **from** и **text**. Затем они используются в теле функции.

Вот ещё один пример: есть переменная **from**, и она передаётся в функцию. Обратите внимание: функция изменяет значение **from**, но это изменение не видно снаружи. Функция всегда получает только копию значения:

function showMessage(from, text) {

    from = '\*' + from + '\*'; // немного украсим "from"

    alert( from + ': ' + text );

  }

  let from = "Аня";

  showMessage(from, "Привет"); // \*Аня\*: Привет

  // значение "from" осталось прежним, функция изменила значение локальной переменной

  alert( from ); // Аня

## Параметры по умолчанию

Если параметр не указан, то его значением становится **undefined**.

Например, вышеупомянутая функция **showMessage(from, text)** может быть вызвана с одним аргументом:

showMessage("Аня");

Это не приведёт к ошибке. Такой вызов выведет «Аня: undefined». В вызове не указан параметр text, поэтому предполагается, что **text === undefined**.

Если параметру text нужно установить значение по умолчанию, требуется указать его после =:

function showMessage(from, text = "текст не добавлен") {

    alert( from + ": " + text );

  }

  showMessage("Аня"); // Аня: текст не добавлен

Теперь, если параметр **text** не указан, его значением будет «текст не добавлен»

В данном случае «текст не добавлен» это строка, но на её месте могло бы быть и более сложное выражение, которое бы вычислялось и присваивалось при отсутствии параметра. Например:

function showMessage(from, text = anotherFunction()) {

    // anotherFunction() выполнится только если не передан text

    // результатом будет значение text

  }

## Возврат значения

Функция может вернуть результат, который будет передан в вызвавший её код.

Простейшим примером может служить функция сложения двух чисел:

function sum(a, b) {

    return a + b;

  }

  let result = sum(1, 2);

  alert( result ); // 3

Директива **return** может находиться в любом месте тела функции. Как только выполнение доходит до этого места, функция останавливается, и значение возвращается в вызвавший её код (присваивается переменной **result** выше).

Вызовов **return** может быть несколько, например:

function checkAge(age) {

    if (age > 18) {

      return true;

    } else {

      return confirm('А родители разрешили?');

    }

  }

  let age = prompt('Сколько вам лет?', 18);

  if ( checkAge(age) ) {

    alert( 'Доступ получен' );

  } else {

    alert( 'Доступ закрыт' );

  }

Возможно использовать **return** и без значения. Это приведёт к немедленному выходу из функции.

Например:

function showMovie(age) {

    if ( !checkAge(age) ) {

      return;

    }

    alert( "Вам показывается кино" ); // (\*)

    // ...

  }

В коде выше, если **checkAge(age)** вернёт **false**, **showMovie** не выполнит **alert**.

## Выбор имени функции

Функция – это действие. Поэтому имя функции обычно является глаголом. Оно должно быть простым, точным и описывать действие функции, чтобы программист, который будет читать код, получил верное представление о том, что делает функция.

Как правило, используются глагольные префиксы, обозначающие общий характер действия, после которых следует уточнение. Обычно в командах разработчиков действуют соглашения, касающиеся значений этих префиксов.

Например, функции, начинающиеся с «show» обычно что-то показывают.

Функции, начинающиеся с…

* «get…» – возвращают значение,
* «calc…» – что-то вычисляют,
* «create…» – что-то создают,
* «check…» – что-то проверяют и возвращают логическое значение, и т.д.

Примеры таких имён:

showMessage(..)     // показывает сообщение

getAge(..)          // возвращает возраст (в каком-либо значении)

calcSum(..)         // вычисляет сумму и возвращает результат

createForm(..)      // создаёт форму (и обычно возвращает её)

checkPermission(..) // проверяет доступ, возвращая true/false

Благодаря префиксам, при первом взгляде на имя функции становится понятным что делает её код, и какое значение она может возвращать.

## Функции == Комментарии

Функции должны быть короткими и делать только что-то одно. Если это что-то большое, имеет смысл разбить функцию на несколько меньших. Иногда следовать этому правилу непросто, но это определённо хорошее правило.

Небольшие функции не только облегчают тестирование и отладку – само существование таких функций выполняет роль хороших комментариев!

Например, сравним ниже две функции **showPrimes(n).** Каждая из них выводит простое число до n.

Первый вариант использует метку **nextPrime**:

function showPrimes(n) {

    nextPrime: for (let i = 2; i < n; i++) {

      for (let j = 2; j < i; j++) {

        if (i % j == 0) continue nextPrime;

      }

      alert( i ); // простое

    }

  }

Второй вариант использует дополнительную функцию **isPrime(n)** для проверки на простое:

function showPrimes(n) {

    for (let i = 2; i < n; i++) {

      if (!isPrime(i)) continue;

      alert(i);  // простое

    }

  }

  function isPrime(n) {

    for (let i = 2; i < n; i++) {

      if ( n % i == 0) return false;

    }

    return true;

  }

Второй вариант легче для понимания, не правда ли? Вместо куска кода мы видим название действия (**isPrime**). Иногда разработчики называют такой код самодокументируемым.

Таким образом, допустимо создавать функции, даже если они повторно не используются. Такие функции структурируют код и делают его более понятным.

**Итого**

Объявление функции имеет вид:

function имя(параметры, через, запятую) {

    /\* тело, код функции \*/

  }

* Передаваемые значения копируются в параметры функции и становятся локальными переменными.
* Функции имеют доступ к внешним переменным. Но это работает только изнутри наружу. Код вне функции не имеет доступа к её локальным переменным.
* Функция может возвращать значение. Если этого не происходит, тогда результат равен **undefined**.

Для того, чтобы сделать код более чистым и понятным, рекомендуется использовать локальные переменные и параметры функций, не пользоваться внешними переменными.

Функция, которая получает параметры, работает с ними и затем возвращает результат, гораздо понятнее функции, вызываемой без параметров, но изменяющей внешние переменные, что чревато побочными эффектами.

Именование функций:

* имя функции должно понятно и чётко отражать, что она делает. Увидев её вызов в коде, вы должны тут же понимать, что она делает, и что возвращает;
* функция – это действие, поэтому её имя обычно является глаголом;
* есть много общепринятых префиксов, таких как: create…, show…, get…, check… и т.д. пользуйтесь ими как подсказками, поясняющими, что делает функция.

Функции являются основными строительными блоками скриптов. Мы рассмотрели лишь основы функций в JavaScript, но уже сейчас можем создавать и использовать их. Это только начало пути. Мы будем неоднократно возвращаться к функциям и изучать их всё более и более глубоко.

# Контрольные вопросы

1. Опишите синтаксис функции.
2. Как можно передать информацию в функцию?
3. Что произойдёт, если внутри функции создать переменную с именем, которая уже объявлена в месте вызова функции?
4. При помощи какого оператора происходит возвращение значения из функции?
5. Какой код называется самодокументируемым?

# Задания

| **№ вар** | **Задача** |
| --- | --- |
|  | Перевести заданное в десятичной системе число a в двоичную систему, используя функцию перевода. |
|  | Даны два натуральных числа. Найти наименьшее общее кратное этих чисел, используя функцию, реализующую алгоритм Евклида. |
|  | Получить все четырехзначные счастливые номера. Счастливым называется номер, у которого сумма первых двух цифр номера равна сумме последних двух цифр. Использовать функцию для расчета суммы цифр двухзначного числа. |
|  | Написать программу определения максимального предельного заряда и потенциала для трех шаров с диаметром r1, r2, r3, используя функции расчета максимального предельного заряда и потенциала. Напряженность поля, при которой начинается ударная ионизация, равна 3 МВ/м. |
|  | Для заданного массива экспериментальных данных рассчитать дисперсию, используя функции определения среднего и дисперсии. |
|  | Получить все шестизначные счастливые номера. Счастливым называется номер, у которого сумма первых трех цифр номера равна сумме последних трех цифр. Использовать функцию для расчета суммы цифр трехзначного числа. |
|  | Даны два натуральных числа. Определить, является хотя бы одно из них палиндромом (число, которое читается одинаково слева направо и справа налево), используя функцию распознавания таких чисел. |
|  | Рассчитать рабочий диапазон длин волн приемника, если емкость конденсатора в его колебательном контуре плавно изменяется от 50 до 500 пФ, а индуктивность катушки постоянна и равна 20 мкГн. Использовать  функцию для определения длины волны |
|  | Отсортировать по убыванию заданный массив чисел, используя функцию сортировки. |
|  | Перевести заданное в двоичной системе число a в десятичную систему. |
|  | Отсортировать по возрастанию заданный массив чисел, исключив отрицательные числа. Использовать функцию сортировки. |
|  | Даны два натуральных числа a и b, обозначающие соответственно числитель и знаменатель дроби. Сократить дробь, используя функцию определения наибольшего общего делителя. |
|  | В заданном массиве целых чисел найти максимальное и минимальное число, использую функции определения максимального и минимального числа. |
|  | Задан одномерный массив из 16 элементов. Сформировать двухмерный массив 4×4, используя функцию преобразования любого одномерного массива в двухмерный массив размерностью n×n с добавлением нулевых элементов. |
|  | Задан одномерный массив из 16 элементов. Сформировать двухмерный массив 4×4, в котором элементы увеличиваются слева направо, используя функции сортировки и преобразования одномерного массива в двухмерный массив. |

# 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