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 № 9**

**Инструкционно-технологическая карта**

Тема: Виды функций: Expression, Declaration, Callback

Цель: Научится разрабатывать скрипты с различными видами функций: Expression, Declaration, Callback.

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

**Function Expression**

Функция в JavaScript – это не магическая языковая структура, а особого типа значение.

Синтаксис, который использовался до этого, называется **Function Declaration** (Объявление Функции):

function sayHi() {

    alert( "Привет" );

  }

Существует ещё один синтаксис создания функций, который называется **Function Expression** (Функциональное Выражение).

Оно выглядит вот так:

let sayHi = function() {

    alert( "Привет" );};

В коде выше функция создаётся и явно присваивается переменной, как любое другое значение. По сути, без разницы, как определяется функция, это просто значение, хранимое в переменной sayHi.

Смысл обоих примеров кода одинаков: создать функцию и поместить её значение в переменную «sayHi».

Можно даже вывести это значение с помощью **alert**:

function sayHi() {

    alert( "Привет" );

  }

  alert( sayHi ); // выведет код функции

Нужно обратить внимание, что последняя строка не вызывает функцию sayHi, после её имени нет круглых скобок. Существуют языки программирования, в которых любое упоминание имени функции совершает её вызов. JavaScript – не один из них.

В JavaScript функции – это значения, поэтому и обращаются с ними, как со значениями. Код выше выведет строковое представление функции, которое является её исходным кодом.

Конечно, функция – не обычное значение, в том смысле, что её можно вызвать его при помощи скобок: sayHi().

Но всё же это значение. Поэтому можно делать с ним то же самое, что и с любым другим значением.

Можно скопировать функцию в другую переменную:

function sayHi() {   // (1) создаём

    alert( "Привет" );

  }

  let func = sayHi;    // (2) копируем

  func(); // Привет    // (3) вызываем копию (работает)!

  sayHi(); // Привет   //     прежняя тоже работает (почему бы нет)

Подробно разберём всё, что произошло в примере выше:

1. Объявление **Function Declaration** (1) создало функцию и присвоило её значение переменной с именем **sayHi**.
2. В строке (2) скопировали её значение в переменную **func**. Нужно обратить внимание (ещё раз): нет круглых скобок после **sayHi**. Если бы они были, то выражение **func** = **sayHi**() записало бы результат вызова **sayHi**() в переменную **func**, а не саму функцию **sayHi**.
3. Теперь функция может быть вызвана с помощью обеих переменных **sayHi**() и **func**().

Заметим, что можно было бы использовать и **Function** **Expression** для того, чтобы создать **sayHi** в первой строке:

let sayHi = function() {

    alert( "Привет" );

  };

  let func = sayHi;

Результат был бы таким же.

**Функции-«колбэки»**

Нужно рассмотреть ещё примеры функциональных выражений и передачи функции как значения.

Напишем функцию **ask(question, yes, no)** с тремя параметрами:

**Question –** текст вопроса

**Yes** – функция, которая будет вызываться, если ответ будет «**Yes**»

**No** – функция, которая будет вызываться, если ответ будет «**No**»

Функция должна задать вопрос **question** и, в зависимости от того, как ответит пользователь, вызвать **yes**() или **no**():

function ask(question, yes, no) {

    if (confirm(question)) yes()

    else no();

  }

  function showOk() {

    alert( "Вы согласны." );

  }

  function showCancel() {

    alert( "Вы отменили выполнение." );

  }

  // использование: функции showOk, showCancel передаются в качестве аргументов ask

  ask("Вы согласны?", showOk, showCancel);

На практике подобные функции очень полезны. Основное отличие «реальной» функции **ask** от примера выше будет в том, что она использует более сложные способы взаимодействия с пользователем, чем простой вызов **confirm**. В браузерах такие функции обычно отображают красивые диалоговые окна. Но это уже другая история.

Аргументы функции **ask** ещё называют функциями-колбэками или просто колбэками.

Ключевая идея в том, что передаём функция передаётся и ожидается, что она вызовется обратно (от англ. «call back» – обратный вызов) когда-нибудь позже, если это будет необходимо. В нашем случае, **showOk** становится **колбэком’** для ответа «**yes**», а **showCancel** – для ответа «**no**».

Можно переписать этот пример значительно короче, используя **Function** **Expression**:

function ask(question, yes, no) {

    if (confirm(question)) yes()

    else no();

  }

  ask(

    "Вы согласны?",

    function() { alert("Вы согласились."); },

    function() { alert("Вы отменили выполнение."); }

  );

Здесь функции объявляются прямо внутри вызова ask(...). У них нет имён, поэтому они называются анонимными. Такие функции недоступны снаружи ask (потому что они не присвоены переменным), но это как раз то, что нам нужно.

Подобный код, появившийся в скрипте, выглядит очень естественно, в духе JavaScript.

**Function Expression в сравнении с Function Declaration**

Нужно разобрать ключевые отличия Function Declaration от Function Expression.

Во-первых, синтаксис: как определить, что есть что в коде.

* **Function Declaration**: функция объявляется отдельной конструкцией «function…» в основном потоке кода.

// Function Declaration

function sum(a, b) {

    return a + b;

  }

* **Function Expression**: функция, созданная внутри другого выражения или синтаксической конструкции. В данном случае функция создаётся в правой части «выражения присваивания» =:

// Function Expression

let sum = function(a, b) {

    return a + b;

  };

Более тонкое отличие состоит, в том, когда создаётся функция движком JavaScript.

**Function Expression создаётся, когда выполнение доходит до него, и затем уже может использоваться.**

После того, как поток выполнения достигнет правой части выражения присваивания **let sum = function**… – с этого момента, функция считается созданной и может быть использована (присвоена переменной, вызвана и т.д. ).

С **Function Declaration** всё иначе.

**Function Declaration можно использовать во всем скрипте (или блоке кода, если функция объявлена в блоке).**

Другими словами, когда движок JavaScript готовится выполнять скрипт или блок кода, прежде всего он ищет в нём **Function Declaration** и создаёт все такие функции. Можно считать этот процесс «стадией инициализации».

И только после того, как все объявления **Function Declaration** будут обработаны, продолжится выполнение.

В результате, функции, созданные, как **Function Declaration** могут быть вызваны раньше своих определений.

Например, так будет работать:

sayHi("Вася"); // Привет, Вася

function sayHi(name) {

  alert( `Привет, ${name}` );

}

Функция sayHi была создана, когда движок JavaScript подготавливал скрипт к выполнению, и такая функция видна повсюду в этом скрипте.

Если бы это было **Function Expression**, то такой код вызвал бы ошибку:

sayHi("Вася"); // ошибка!

let sayHi = function(name) {  // (\*) магии больше нет

  alert( `Привет, ${name}` );

};

Функции, объявленные при помощи **Function Expression**, создаются тогда, когда выполнение доходит до них. Это случится только на строке, помеченной звёздочкой (\*). Слишком поздно.

Ещё одна важная особенность **Function Declaration** заключается в их блочной области видимости.

В строгом режиме, когда **Function Declaration** находится в блоке {...}, функция доступна везде внутри блока. Но не снаружи него.

Для примера представим, что нужно создать функцию welcome() в зависимости от значения переменной age, которое получаем во время выполнения кода. И затем запланируем использовать её когда-нибудь в будущем.

Такой код, использующий **Function Declaration**, работать не будет:

let age = prompt("Сколько Вам лет?", 18);

// в зависимости от условия объявляем функцию

if (age < 18) {

  function welcome() {

    alert("Привет!");

  }

} else {

  function welcome() {

    alert("Здравствуйте!");

  }

}

// ...не работает

welcome(); // Error: welcome is not defined

Это произошло, так как объявление **Function Declaration** видимо только внутри блока кода, в котором располагается.

Вот ещё один пример:

let age = 16; // присвоим для примера 16

if (age < 18) {

  welcome();               // \   (выполнится)

  function welcome() {     //  |

    alert("Привет!");      //  |  Function Declaration доступно

  }                        //  |  во всём блоке кода, в котором объявлено

  welcome();               // /   (выполнится)

} else {

  function welcome() {

    alert("Здравствуйте!");

  }

}

// здесь фигурная скобка закрывается,

// поэтому Function Declaration, созданные внутри блока кода выше -- недоступны отсюда.

welcome(); // Ошибка: welcome is not defined

Что можно сделать, чтобы welcome была видима снаружи **if**?

Верным подходом будет воспользоваться функцией, объявленной при помощи **Function Expression**, и присвоить значение **welcome** переменной, объявленной снаружи **if**, что обеспечит нужную видимость.

Такой код работает, как ожидалось:

let age = prompt("Сколько Вам лет?", 18);

let welcome;

if (age < 18) {

  welcome = function() {

    alert("Привет!");

  };

} else {

  welcome = function() {

    alert("Здравствуйте!");

  };

}

welcome(); // теперь всё в порядке

Можно упростить этот код ещё сильнее, используя условный оператор **?**:

let age = prompt("Сколько Вам лет?", 18);

let welcome = (age < 18) ?

  function() { alert("Привет!"); } :

  function() { alert("Здравствуйте!"); };

welcome(); // теперь всё в порядке

**Итого**

* Функции – это значения. Они могут быть присвоены, скопированы или объявлены в любом месте кода.
* Если функция объявлена как отдельная инструкция в основном потоке кода, то это Function Declaration.
* Если функция была создана как часть выражения, то считается, что эта функция объявлена при помощи Function Expression.
* Function Declaration обрабатываются перед выполнением блока кода. Они видны во всём блоке.
* Функции, объявленные при помощи Function Expression, создаются, только когда поток выполнения достигает их.

В большинстве случаев, когда нужно создать функцию, предпочтительно использовать Function Declaration, т.к. функция будет видима до своего объявления в коде. Это позволяет более гибко организовывать код и улучшает его читаемость.

Таким образом, нужно прибегать к объявлению функций при помощи

Function Expression в случае, когда синтаксис Function Declaration не подходит для решения задачи.

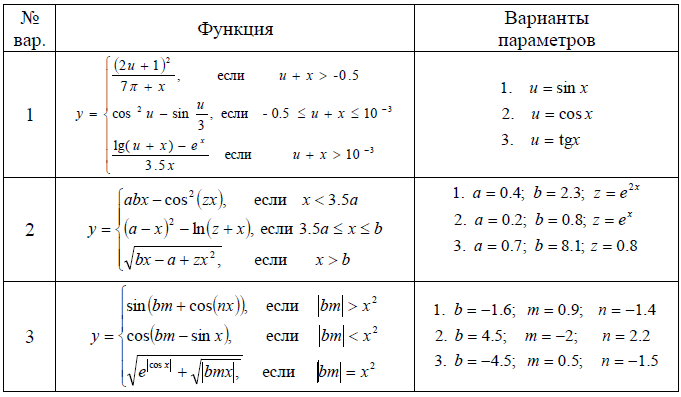
# Контрольные вопросы

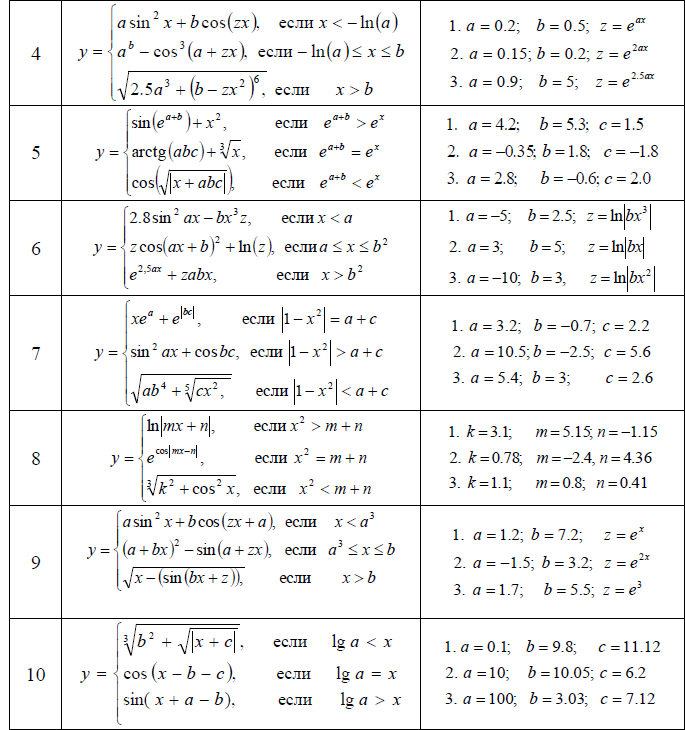
1. Какие существуют три вида функций?
2. В чём отличие между разными видами функций?
3. Чем являются функции в JavaScript?
4. В какой момент создаются функции при разных видах объявления?
5. Приведите пример функций, написанных разными методами объявления.
6. Что такое анонимная функция?

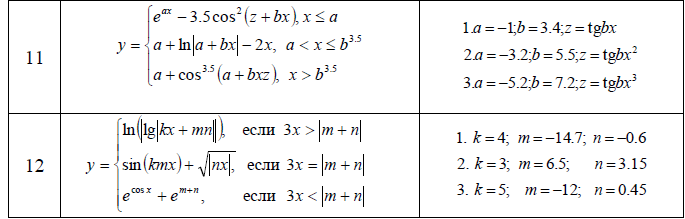
# Задания

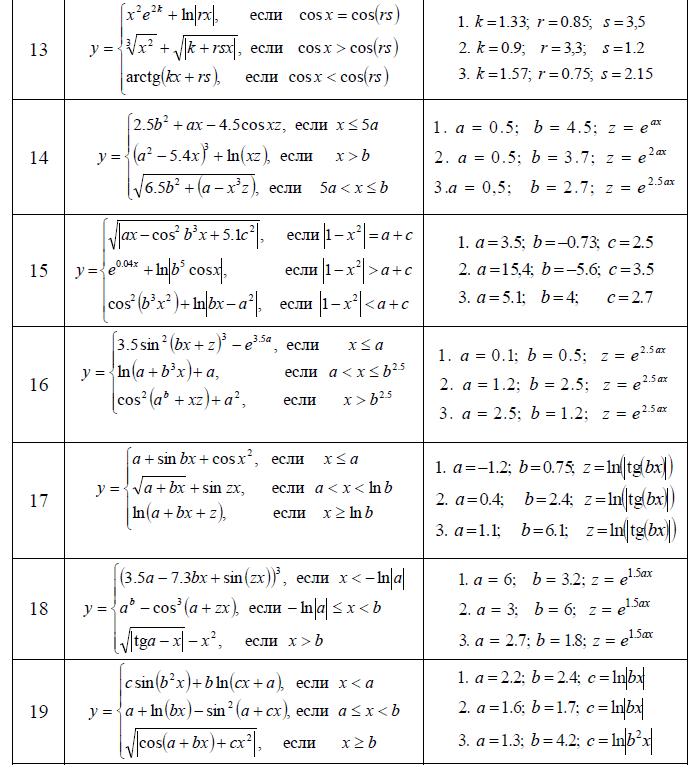
Реализуйте функции для решения уравнений как при помощи **Function Expression,** так и при помощи **коллбэков.**

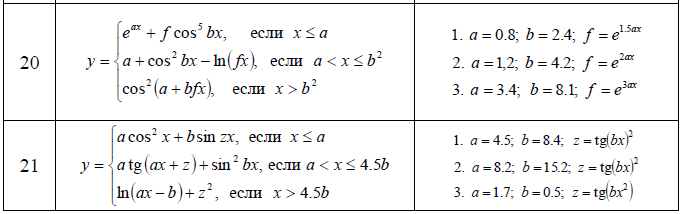
В данном задании пользователь сначала выбирает, по какой формуле вычислять исходные значения (правый столбец), после чего на основании этих значений высчитывается значение по формулам в левом столбце в зависимости от выполнения условий.

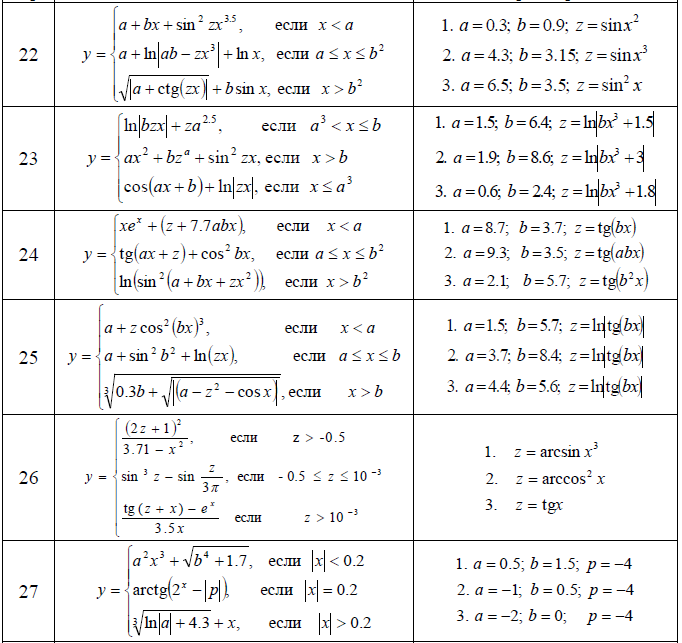


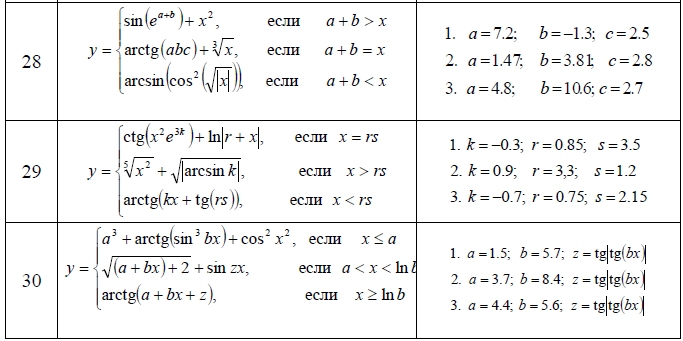












# Литература

**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