**Лабораторная работа № 4**

1. **Встроенные Javascript функции**

В JavaScript достаточно много функций, встроенных в синтаксис языка. Рассмотрим одну из них.

**eval(строка)**

Функция eval(строка) вычисляет выражение в указанной строке (в качестве параметра); выражение должно быть составлено по правилам языка JavaScript и не содержать тегов HTML

Рассмотрим примеры использования функции *eval*:

|  |  |
| --- | --- |
| 1  2  3 | **var** y = 5; *// значение у равно 5*  **var** x = "if (y==5) y\*2-3"; *// значение х равно строке символов*  **var** rezult = eval(x); *// rezult равно 7* |

**Задание 1.** В скрипте запрашивать у пользователя ввести в диалоговое окно ввода JavaScript математическое выражение, затем это выражение должно вычисляться и результат выдаваться в диалоговом окне.

1. **Пользовательские Javascript функции**

**Синтаксис создания функции:**

1. **Функция в роли процедуры. Если функция выполняет какие-либо действия и не возвращает значение:**

|  |
| --- |
| **function** имя\_функции(аргументы){  код;  } |

**Важно:** Если аргументы у функции отсутствуют, то после имени функции ставятся пустые скобки ().

*В javascript****вызов функции в роли процедуры*** *происходит следующим образом:*

|  |
| --- |
| имя\_функции (аргументы); *// с агрументами*  имя\_функции (); *// без аргументов* |

**Пример:**

|  |
| --- |
| **function** sayHello()  {  alert("hello");  }  *//...*  sayHello(); |

1. **Если функция возвращает значение**

|  |
| --- |
| **function** имя\_функции(аргументы){  код;  **return** результат или выражение  } |

**Важно:** В javascript return — оператор, который завершает выполнение функции и возвращает значение

***В javaScript вызов функции, возвращающей значение, происходит следующим образом:***

|  |
| --- |
| **var** a = имя\_функции (аргументы);  alert(a); |

**Пример:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **function** sayHello()  {  alert("1");*// выполнится*  **return** "привет!";*// завершаем выполнение функции*  alert("2");*// не выполнится, т.к. после return*  }    **var** a=sayHello();  alert(a); |

Как видно из примера, после return функция действительно завершает свою работу. Поэтому оператор alert("2") выполняться уже не будет.  
Вызов же функции происходит через присваивание. Вызов в данном примере можно также организовать так: alert(sayHello()); , но это не рекомендуемый вариант.

**Важно:** Таким образом, оператор return указывает на возвращаемое значение функции, которое будет передано переменной при следующей форме вызова:  
переменная = имя\_функции();

**Задание 2.** Дополните код согласно заданию: В переменную povtor присвоить результат выполнения функции confirm, которая принимает фразу *«Пройти заполнение еще раз?»*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **function** say() {  document.write("Вы здесь?" + "<br/>");  }  **do** {  say();  povtor = ...  } while (povtor); |

**Задание 3.** Создать функцию, которая вычитает два вводимых пользователем в диалоговое окно числа и выводит сообщение с результатом (функция без параметров)

1. **Javascript функции с параметрами (аргументами) и возврат значений**

Рассмотрим подробнее использование инструкции return в функции javascript на двух примерах. Кроме того, рассмотрим использование в Javascript функции с параметрами (аргументами).

1. **функция возвращает значение undefined, но выполняет какие-то действия:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **function** sayHello(userName)  {  alert("Привет " + userName);  }  sayHello("Вася");  sayHello("Петя"); |

В примере функция имеет один аргумент, указанный в скобках. Переменная userName примет то значение, которое указано в скобках при вызове функции "Вася" и "Петя". Значит при первом вызове функция выведет в диалоговое окно Привет Вася, при втором — Привет Петя.

1. **функция возвращает конкретное значение:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **function** sayHello(userName)  {  **return** "Привет " + userName;  }  **var** a = sayHello("Вася");  alert(a); |

**Важно:** Обратите внимание на разницу между вызовом функции без инструкции return и с инструкцией: в первом случае функция вызывается как оператор, возвращая значение undefined; во втором случаем функция вызывается через присваивание, либо участвуя в выражениях.

**Задание 4.** Что выведет на экран следующий код?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **var** a = 5, b = 10, c = 20, d = 7;  **function** Sum1() {  **var** result = a + b;  document.write("Sum1: " + result + "<br/>");  }  **function** Sum2(x1, x2) {  **var** result = x1 + x2;  document.write("Sum2: " + result);  }  Sum1();  Sum2(c, d); |

**Задание 5.** Необходимо запросить у пользователя имя. Вызывать функцию для вызова диалогового окна с сообщением «Привет, имя!»

## Все способы создания пользовательских функций

1. **классический синтаксис**

|  |
| --- |
| **function** a1(x, y) {  **return** x + y;  } |

1. **явное создание объекта Function**

|  |
| --- |
| *//обязательным является только последний аргумент – тело функции*  **var** a1 = **new** Function('x', 'y', 'return x + y'); |

или

|  |
| --- |
| **var** a1 = **new** Function('return "hello"'); |

1. **еще один вариант создания**

|  |
| --- |
| **var** a1 = **function**(x, y) { **return** x + y; }; |

**Вызов функций во всех случаях будет:**

|  |
| --- |
| **var** a = a1(3, 4); |

## Использование выражений с функциями

**Обычное использование javascript функции:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **function** sum(arg1,arg2)  {  **var** a=arg1+arg2;  **return** a;  }  **var** b=sum(1,2);  alert(b); |

**Функция как составная часть выражения:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **function** sum(arg1,arg2)  {  **var** a=arg1+arg2;  **return** a;  }  **var** b=sum(1,2);  alert(b);    **function** add()  {  **var** c=1+sum(1,2);  **return** c;  }  **var** d=add();  alert(d); |

Во второй функции используется выражение, ссылающееся на первую функцию (в 11-й строке).

**Задание 6.** Создать функцию, возвращающую наибольшее из трех чисел. Аргументами функции являются сами числа

**Рассмотрим другие варианты использования выражений с функциями (только для функций, которые возвращают результат):**  
Функция:

|  |  |
| --- | --- |
| 1  2  3  4 | **function** plRectangle(width, height){  **var** S = width \* height;  **return** S  } |

**Варианты выражений:**

1. **Вызов функции как часть выражения:**

|  |  |
| --- | --- |
| 1 | S3 = 0.5 \* plRectangle(a, b); |

1. **Вызов функции в логических выражениях**

|  |  |
| --- | --- |
| 1  2 | **if** (plRectangle(a, b) > plRectangle(c, d))  alert("Первый прямоугольник больше второго"); |

1. **Вызов javascript функции в качестве параметра другой функции**

|  |  |
| --- | --- |
| 1  2  3 | **var** х = "25рх";  **var** у = 12;  **var** S = plRectangle(parselnt(x), у); |

**Задание 7.** Создайте функцию для расчета степени введенного числа

## Область видимости переменных. Javascript глобальные и локальные переменные в функции

**Область видимости переменной** — область кода, в котором переменная доступна для использования.

1. **Глобальные переменные**

— создаются на уровне сценария и сохраняются до конца сценария;

— объявляются до описания javascript функции;

— могут быть причиной сложно находимых ошибок;

1. **Локальные переменные**

— создаются внутри фрагментов кода и не видны извне;

— явно объявляются в теле javascript функции;

— аргументы (параметры) функции — всегда локальные переменные;

— лучше использовать локальные переменные, так как доступ к ним больше контролируется.

**Задание 8**. Дополните код согласно заданию:

Создать 2 переменные глобальной и локальной области видимости (то есть внутри функции func) с именами: global, local.

В переменную global занести текст “Привет”, в local — “Мир”.

|  |
| --- |
| **function** func() {    }  func(); |

### ОБЛАСТЬ ВИДИМОСТИ ПЕРЕМЕННЫХ

Рассмотрим конкретные примеры области видимости переменных в javascript при использовании глобальных и локальных переменных.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **function** plRectangle(width, height){  **var** S = width \* height;  **return** S *// Локальная переменная S*  }  **var** S = 2; *// Глобальная переменная S*  z = plRectangle(2, 3);  alert(z);  alert(s); |

**Пример:** Значение z равно 6, а значение S осталось равным 2, то есть значению глобальной переменной, определенной во внешней программе

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **function** plRectangle(width, height)  {  **var** s = width \* height; *// аргументы всегда локальны*  width = width + 10;  **return** s  }  width = 2;  height = 3;  z = plRectangle(width, height);  alert(z);  alert(width); |

**Пример:** Значение z равно 6; значение переменной widthравно 2, то есть осталось без изменений

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **function** plRectangle(width, height)  {  S = width \* height;  **return** S *// S - глобальная переменная (т.к. без определения var)*  }  **var** z = plRectangle(2, 3);  alert(z);  alert(S); |

**Пример:** Значения и z и S равны 6; S — глобальная переменная

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **function** Plrectangle(width, height){  S = width \* height;  **return** S  }  z = Plrectangle(2, 3);  S=2;  alert(z);  alert (S); |

**Пример:** Значение z равно 6, а значение S осталось равным 2, то есть значению глобальной переменной, определенной во внешней программе

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **function** plRectangle(width, height)  {  **var** S = width \* height;  **var** x = 17;  **return** S  }  z = plRectangle(2,3);  alert(z);  alert(x); *// Локальная переменная x*  alert (S); *// Локальная переменная S* |

**Пример:** Значение z равно 6; переменная S во внешней программе не определена; переменная х во внешней программе не определена

**Задание 9.** Что выведет на экран следующий код?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **var** variable = "Глобальная переменная";  **function** f() {  **var** variable = "Локальная переменная";  document.write(variable + "<br/>");  }  f();  document.write(variable); |

## 

## Рекурсивная функция javascript

**Важно:** В информатике и программировании, а, соответственно, и в javascript, рекурсия — это вызов функции из самой же функции, т.е. функция в функции

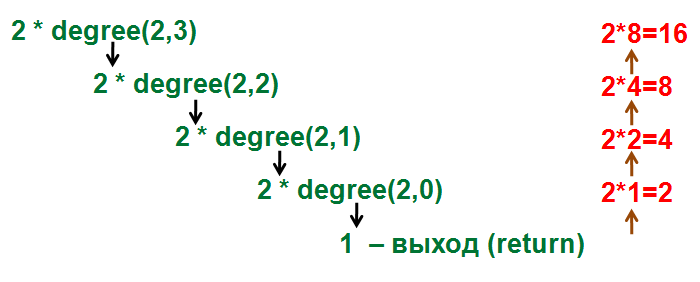
Бывают также **косвенная**или **сложная рекурсия**, когда функция вызывается не непосредственно из самой себя, а из вложенной в нее функции: например, функция A вызывает функцию B, а функция B — функцию A. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

Рекурсия javascript рассмотрена ниже на примере возведения числа в степень.

**Для начала рассмотрим итерационный вариант возведения в степень, т.е. с использованием цикла:**

**Пример:** вы**полним возведение числа в степень через рекурсию:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **var** chislo,stepen;  **function** degree(chislo,stepen) {  **if**(stepen) {  **return** chislo\*degree(chislo,stepen-1);  }  **return** 1;  }  document.write(degree(2,4)); *// выводит 16* |

[](http://labs.org.ru/wp-content/uploads/2016/08/1-5.png)

**Задание 10.** Что выведет на экран следующий код?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **function** f(counter) {  counter--;  document.write(counter + "<br/>");  **if** (counter != 0)  f(counter);  document.write(counter + "<br/>");  }  f(2); |

**Задание 11**. Вычислить (a! + b!)/a!, используя рекурсивную функцию вычисления факториала.

**Задание 12.** Составить рекурсивную функцию вычисления **n**-го члена арифметической прогрессии 3, 7, … и вывести первые 10 членов прогрессии.

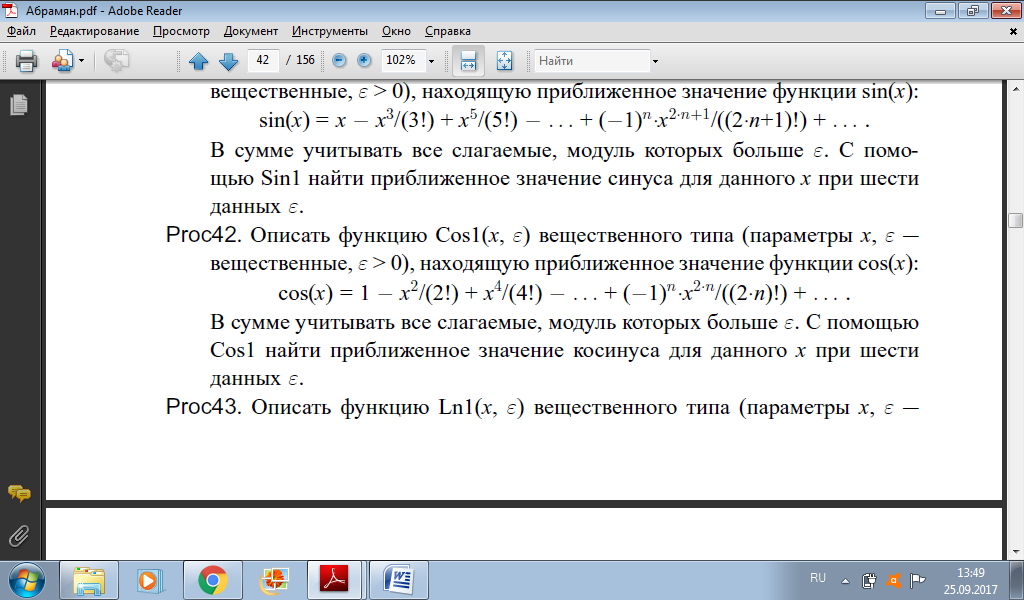
**Задание 13.** Составить рекурсивную процедуру, которая печатает введенное натуральное число в восьмеричном представлении.

**Задание 14.** Описать функцию *DigitN* (*K,N*) целого типа, возвращающую *N*-ю цифру целого положительного числа *K (*цифры в числе нумеруются справа налево*).*

Если количество цифр в числе *K* меньше *N,* то функция возвращает – 1 . Для каждого из пяти данных целых положительных чисел *K1, K2, …, K5* вызвать функцию *DigitN* с параметром *N,* изменяющимся от 1 до 5.

**Задание 15.**

Описать функцию Cos1(*x*, *Ɛ*) вещественного типа (параметры х, *Ɛ –* вещественные, *Ɛ>0*), находящую приближенное значение функции cos(*x*):

****

В сумме учитывать все слагаемые, модуль которых больше *Ɛ.* С помощью Cos1 найти приближенное значение косинуса для данного х при шести данных *Ɛ.*