**Лабораторная работа №9**

**Тема работы:** Разработка программ с применением рекурсивных функций. Переопределение функций

**Цель работы:** закрепить навыки по работе с рекурсивными функциями.

**Теоретические сведения**

**Значения по умолчанию**

Пусть у нас есть функция, которая возводит число в квадрат:

function func(num) {

return num **\*** num

}

Давайте с ее помощью возведем в квадрат число **3**:

alert(func(3)); //выведет 9

Что будет, если в функцию не передать параметр: это не приведет к ошибке, просто в переменную **num** запишется **undefined**. То есть: в JavaScript в функцию можно не передавать все параметры - если какой-то не передан, вместо него запишется undefined.

Эту особенность можно использовать для указания *значений по умолчанию*. Под ним понимается следующее: если параметр не передан, то вместо него возьмется заданное значение.

Давайте сделаем так, чтобы, если мы не передали параметр num, то по умолчанию бралось значение 3. То есть мы хотим, чтобы функция работала так:

//Параметр не передали

alert(func()); //хотим, чтобы вывело 9

//Параметр передали

alert(func(4)); //выведет 16

Для этого есть 2 способа. Первый способ - проверить переменную на undefined и, если она равна ему, записать в нее нужно значение:

function func(num) {

if (num **===** undefined) {

num **=** 3;

}

return num **\*** num;

}

Второй способ заключается в использовании логического ||, работу с которым мы разбирали в уроке [приемы работы с логическими значениями](http://old.code.mu/books/javascript/base/priemy-raboty-s-logicheskimi-znacheniyami-na-javascript.html):

function func(num) {

num **=** num **||** 3;

return num **\*** num;

}

Второй способ считает, что параметр отсутствует, если передана пустая строка, 0, или вообще любое значение, которое в логическом контексте является false.

В функцию можно передавать несколько параметров и часть из них (с конца) сделать необязательными, присвоив им значение по умолчанию:

function func(num1, num2, num3) {

num3 **=** num3 **||** 3;

num2 **=** num2 **||** 2;

return num **\*** num;

}

**Область видимости переменных**

В программировании существует такое понятие как *область видимости переменной*. Оно обозначает те места кода, в которых переменная имеет то значение, которое мы ей присвоили.

Различают *глобальные* и *локальные* переменные. Первые видны во всем скрипте, а вторые - только ограниченной области.

В следующем примере переменная внутри функции не видна снаружи, она является локальной внутри функции:

function func() {

var local **=** 'Тест!';

alert(local); //выведет 'Тест!'

}

func();

alert(local); //выведет undefined

А сейчас та переменная **test**, которая находится вне функции, это не та переменная **test**, которая находится внутри нее:

var test **=** 'Тест!';

function func() {

var test **=** 'Локальная!'; alert(test); //выведет 'Локальная!'

}

func();

А вот если внутри функции переменная не объявлена через **var** - она будет считаться глобальной:

var test **=** 'Тест!';

function func() {

alert(test); //выведет 'Тест!'

}

func();

Сейчас мы забыли написать **var** внутри функции и случайно затерли глобальную переменную (такая особенность может привести к трудноуловимым ошибкам, поэтому всегда ставьте var):

var global **=** 'Тест!';

function func() {

global **=** 'Поменялась!';

}

func();

alert(global); //выведет 'Поменялась!'

Если написать **var** - переменная будет локальной внутри функции:

var global **=** 'Тест!';

function func() {

var global **=** 'Поменялась!';

}

func();

alert(global); //выведет 'Тест!'

Еще раз напомню: переменная внутри функции не видна снаружи:

function func() {

var local **=** 'Тест!';

alert(local); //выведет 'Тест!'

}

func();

alert(local); //выведет undefined

А теперь мы забыли написать **var** перед переменной и она видна снаружи:

function func() {

local **=** 'Тест!';

alert(local); //выведет 'Тест!'

}

func();

alert(local); //выведет 'Тест!'

**Строгий режим**

Забыв поставить **var**, можно случайно затереть глобальную переменную, что является источником трудноуловимых ошибок. Поэтому рекомендуется включать так называемый *строгий режим* командой **'use strict';**, которая пишется в начале скрипта.

В следующем примере мы забыли поставить **var** и затерли глобальную переменную **test**:

var test **=** 'Привет!';

function func() {

test **=** 'Тест!';

}

func();

alert(test); //выведет 'Тест!', а не 'Привет!'

А если включить строгий режим, то скрипт просто выдаст ошибку, которую мы сможем заметить в консоли:

'use strict';

var test **=** 'Привет!';

function func() {

test **=** 'Тест!'; //будет ошибка, нет var

}

func();

alert(test);

**Глобальные переменные через window**

Давайте еще раз поговорим про глобальные переменные: все переменные, написанные вне функций, являются глобальными, то есть потенциально видны в любом месте.

Однако, это еще не все: все глобальные переменные являются свойствами объекта **window**:

var test **=** 'Тест!';

alert(window.test); //выведет 'Тест!'

И наоборот:

var window.test **=** 'Тест!';

alert(test); //выведет 'Тест!'

Такую особенность можно использовать для обмена данными между функциями:

function func1() {

window.test **=** 'Тест!';

}

function func2() {

alert(window.test); //выведет 'Тест!'

}

func1(); //<--- без этого вызова не сработает

func2();

Если не вызывать первую функцию, то код **window.test = 'Тест!'** не сработает (он срабатывает только в момент вызова **func1()**):

function func1() {

window.test **=** 'Тест!';

}

function func2() {

alert(window.test); //выведет undefined

}

//Тут func1() больше не вызывается

func2();

**Рекурсия**

В программировании есть такое понятие, как *рекурсия* - это когда функция вызывает сама себя.

Давайте разберемся с этим понятием на следующем примере: выведем все элементы массива с конца с помощью рекурсии (для примера, цикл тут эффективнее).

Для этого мы будем использовать метод pop, который забирает из массива последний элемент (массив при этом уменьшается). Затем будем проверять, осталось ли что в массиве с помощью **empty**.

Если осталось, опять запускаем эту же функцию **last**, которая забирает последний элемент, проверяет осталось ли что в массиве и запускает саму себя при необходимости:

var arr **=** [1, 2, 3, 4, 5];

last(arr);

function last(arr) {

document.write(arr.pop() **+** '<br>'); //выводим последний элемент массива

if(arr.length **!=** 0) {

last(arr); //это рекурсия

}

}

**Порядок выполнения работы**

**Практическая часть**

**Работа с рекурсией**

1. Дан массив с числами. Выведите последовательно его элементы используя рекурсию и не используя цикл.
2. Дано число. Сложите его цифры. Если сумма получилась более 9-ти, опять сложите его цифры. И так, пока сумма не станет однозначным числом (9 и менее).
3. Напишите функцию mul(n,m), которая принимает два аргумента и возвращает произведение этих аргументов. Проверьте ее работу.
4. Создайте функцию repeat(str, n), которая возвращает строку, состоящую и n повторений строки str. n — по умолчанию 2, str — пустая строка
5. Создайте функцию rgb(), которая будет принимать три числовых аргумента и возвращать строку вида «rgb(23,100,134)». Если аргументы не заданы, считать их равными нулю. Не проверять переменные на тип данных
6. Напишите функцию operation(m,n,o), в которой m и n — числовые переменные, а o — функциональный литерал, который берет два аргумента и выполняет математическую операцию над ними
7. Создайте функцию m(a,b) оболочку для mul(). m() должна принимать два аргумента а возвращать результат работы mul() с этими двумя аргументами После выполнения задания поэкспериментируйте, создайте функцию log(), которая будет принимать одно значение, а вызывать console.log() с этим значением.