ОПЕРАЦИОННЫЕ СИСТЕМЫ

ЛАБОРАТОРНАЯ РАБОТА №6

Основы JScript

Windows Script Host

С помощью командного интерпретатора *cmd.exe* трудно написать какую-либо сложную программу-сценарий (script): отсутствует полноценная интерактивность, нельзя напрямую работать с рабочим столом Windows и системным реестром и т. д. Для исправления этой ситуации компанией Microsoft был разработан сервер сценариев Windows Script Host (WSH), с помощью которого можно выполнять сценарии, написанные, в принципе, на любом языке (при условии, что для этого языка установлен соответствующий модуль (scripting engine), поддерживающий технологию ActiveX Scripting). В качестве стандартных языков поддерживаются Visual Basic Script Edition (VBScript) и JScript.

Возможности технологии ActiveX

Windows с самого начала для обеспечения обмена данными между приложениями была разработана технология связывания и внедрения объектов (Object Linking and Embedding, OLE). Вначале технология OLE использовалась для создания составных документов, а затем для решения более общей задачи — предоставления приложениями друг другу собственных функций (служб) и правильного использования этих функций. Технология, позволяющая одному приложению (клиенту автоматизации) вызывать функции другого приложения (сервера автоматизации) была названа OLE Automation. В основе OLE и OLE Automation лежит разработанная Microsoft базовая "компонентная" технология Component Object Model (COM). В общих словах, компонентное программное обеспечение — это способ разработки программ, при котором используются технологии создания программных модулей, подобные технологиям, применяемым для разработки аппаратных средств. Сложные элементные схемы собираются из стандартизированных микросхем, которые имеют четко определенные документированные функции. Разработчик может эффективно пользоваться такими микросхемами, не задумываясь об их внутренней структуре. В программных компонентах, написанных на каком-либо языке программирования, детали реализации используемых алгоритмов также скрыты внутри компонента (объекта), а на поверхности находятся общедоступные интерфейсы, которыми могут пользоваться и другие приложения, написанные на том же или другом языке.

В настоящее время термин OLE используется только по историческим причинам. Вместо него Microsoft с 1996 года использует новый термин - ActiveX, первоначально обозначавший WWW (World Wide Web) компоненты (объекты), созданные на базе технологии COM. Сервер сценариев WSH является мощным инструментом, предоставляющим единый интерфейс (объектную модель) для специализированных языков (VBScript, JScript, PerlScript, REXX, TCL, Python и т. п.), которые, в свою очередь, позволяют использовать любые внешние объекты ActiveX. С помощью WSH сценарии могут быть выполнены непосредственно в операционной системе Windows, без встраивания в HTML-страницы.

Назначение и основные свойства WSH

WSH предъявляет минимальные требования к объему оперативной памяти, и является очень удобным инструментом для автоматизации повседневных задач пользователей и администраторов операционной системы Windows. Используя сценарии WSH, можно непосредственно работать с файловой системой компьютера, а также управлять работой других приложений (серверов автоматизации). При этом возможности сценариев ограничены только средствами, которые предоставляют доступные серверы автоматизации.

Создание и запуск простейших сценариев WSH

Простейший WSH-сценарий, написанный на языке JScript или VBScript — это обычный текстовый файл с расширением js или vbs соответственно, создать его можно в любом текстовом редакторе, способном сохранять документы в формате "Только текст". Размер сценария может изменяться от одной до тысяч строк, предельный размер ограничивается лишь максимальным размером файла в соответствующей файловой системе.

В качестве первого примера создадим JScript-сценарий, выводящий на экран диалоговое окно с надписью: "Привет!". Для этого достаточно с помощью, например, стандартного Блокнота Windows (notepad.exe) создать файл First.js, содержащий всего одну строку:

WScript.Echo("Привет!");

Тот же самый сценарий на языке VBScript, естественно, отличается синтаксисом и выглядит следующим образом:

WScript.Echo "Привет!"

Несмотря на то, что для работы этих двух сценариев достаточно всего одной строки, желательно сразу приучить себя к добавлению в начало файла информации о находящемся в нем сценарии: имя файла, используемый язык, краткое описание выполняемых действий. На языке JScript такая информация, оформленная в виде комментариев, может выглядеть следующим образом:

```
/**********************************************************/
/* Имя: First.js */
/* Язык: JScript */
/* Описание: Вывод на экран приветствия */
/**********************************************************/
```

На языке VBScript то же самое выглядит следующим образом:

```
' ******************************************************
' Имя: First.vbs
' Язык: VBScript
' Описание: Вывод на экран приветствия
'******************************************************
```

Для запуска сценариев WSH существует несколько способов.

Запуск сценария из командной строки в консольном режиме

Можно выполнить сценарий из командной строки с помощью консольной версии WSH cscript.exe. Например, чтобы запустить сценарий, записанный в файле C:\Script\First.js, нужно загрузить командное окно и выполнить в нем команду

cscript C:\Script\First.js

В результате выполнения этого сценария в командное окно выведется строка *"Привет!"*.

Запуск сценария из командной строки в графическом режиме

Сценарий можно выполнить из командной строки с помощью (оконной) графической версии WSH wscript.exe. Для нашего примера в этом случае нужно выполнить команду

wscript C:\Script\First.js

Тогда в результате выполнения сценария на экране появится нужное нам диалоговое окно. Таким образом, мы видим, что при запуске сценария в консольном режиме, вывод текстовой информации происходит в стандартный выходной поток (на экран), при запуске в графическом режиме - в диалоговое окно.

Запуск сценария с помощью меню Пуск

Для запуска сценария с помощью пункта *Выполнить (Run)* меню *Пуск (Start)*, достаточно написать полное имя этого сценария в поле *Открыть (Open)* и нажать кнопку *Ok*. В этом случае по умолчанию сценарий будет выполнен с помощью *wscript.exe*, т. е. вывод информации будет вестись в графическое диалоговое окно.

Запуск сценария с помощью Проводника Windows (Windows Explorer)

Самым простым является запуск сценария в окнах *Проводника Windows* или на рабочем столе - достаточно просто выполнить двойной щелчок мышью на имени файла со сценарием или на его значке (аналогично любому другому исполняемому файлу). При этом, как и в случае запуска с помощью меню *Пуск (Start)*, сценарий по умолчанию выполняется с помощью *wscript.exe*.

Установка и изменение свойств сценариев

В случае необходимости для сценариев можно задавать различные параметры, влияющие на ход их выполнения. Для консольной (*cscript.exe*) и графической (*wscript.exe*) версий сервера сценариев эти параметры задаются по-разному. Если сценарий запускается в консольном режиме, то его исполнение контролируется с помощью параметров командной строки для *cscript.exe*, которые включают или отключают различные опции WSH (все эти параметры начинаются с символов "//"). Например, команда

cscript //Nologo C:\Script\First.js

запустит сценарий *First.js* без информации о версии WSH.

Сценарий можно запускать с параметрами командной строки, которые указываются после имени этого сценария. Например, команда

cscript //B C:\Script\First.js /a /b

запустит сценарий *First.js* в пакетном режиме, при этом */a* и */b* будут являться параметрами этого сценария, а *//B* — параметром приложения *cscript.exe.*

Если сценарий запускается в графическом режиме (с помощью *wscript.exe*), то свойства сценария можно устанавливать с помощью вкладки *Сценарий (Script)* диалогового окна, задающего свойства файла в Windows. После задания свойств сценария автоматически создается файл с именем этого сценария и расширением wsh, который имеет структуру наподобие ini-файла, например:

```
[ScriptFile]
Path=C:\Script\First.js
[Options]
Timeout=0
DisplayLogo=1
```

Если дважды щелкнуть в Проводнике Windows по wsh-файлу или запустить такой файл из командной строки, то соответствующий сервер сценариев (wscript.exe или cscript.exe) запустит сценарий, которому соответствует wsh-файл, с заданными в секции *Options* параметрами. При запуске сценариев с помощью wscript.exe для задания параметров командной строки сценария можно использовать технологию drag-anddrop - если выделить в Проводнике Windows несколько файлов и перетащить их на ярлык сценария, то этот сценарий запустится, а имена выделенных файлов передадутся ему в качестве параметров.

Таблица. Параметры командной строки для cscript.exe

Параметр	**Описание**
//I	Выключает пакетный режим (по умолчанию). При этом на экран будут выводиться все сообщения об ошибках в сценарии
//B	Включает пакетный режим. При этом на экран не будут выводиться никакие сообщения
//T:nn	Задает тайм-аут в секундах, т. е. сценарий будет выполняться nn секунд, после чего процесс прервется. По умолчанию время выполнения не ограничено
//Logo	Выводит (по умолчанию) перед выполнением сценария информацию о версии и разработчике WSH
//Nologo	Подавляет вывод информации о версии и разработчике WSH

//H:CScipt или //H:Wscript	Делает cscript.exe или wscript.exe приложением для запуска сценариев по умолчанию. Если эти параметры не указаны, то по умолчанию подразумевается wscript.exe
//S	Сохраняет установки командной строки для текущего пользователя
//?	Выводит встроенную подсказку для параметров командной строки
//E:engine	Выполняет сценарий с помощью модуля, заданного параметром engine
//D	Включает отладчик
//X	Выполняет программу в отладчике
//Job:<JobID>	Запускает задание с индексом JobID из многозадачного WS-файла (структура WS-файлов будет описана ниже)
//U	Позволяет использовать при перенаправлении ввода-вывода с консоли кодировку Unicode

Язык JScript

Подобно многим другим языкам программирования, код на Microsoft *JScript* пишется в текстовом формате, и организован в инструкции, блоки, состоящие из связанных наборов инструкций, и комментариев. В пределах инструкции Вы можете использовать переменные и данные, такие как строки, числа и выражения.

Инструкции

Код JScript-инструкции состоит из одного или более символов в строке. Новая строка начинает новую инструкцию, но хорошим стилем является объявление конца инструкции явно. В JScript для этого используется точка с запятой (;).

```
aBird = "Robin";
var today = new Date();
```

Группа JScript-инструкций, заключенная в фигурные скобки ({}), называется блоком. Блоки инструкций используются как функции и условные выражения. В следующем примере, первая инструкция определяет функцию, которая состоит из блока пяти инструкций. Последние три инструкции, которые не окружены фигурными скобками - блоком не являются.

```
function convert(inches) {
feet = inches / 12; // Эти пять инструкций - блок.
miles = feet / 5280;
nauticalMiles = feet / 6080;
cm = inches * 2.54;
meters = inches / 39.37;
}
km = meters / 1000; // Эти инструкции блоком не являются.
kradius = km;
mradius = miles;
```

Комментарии

Комментарием в JScript является текст, расположенный после двойного слэша (//) до конца строки. Многострочный комментарий начинается слэшем со знаком умножения (/*), и кончается их обратной комбинацией (*/).

```
aGoodIdea = "Comment your code thoroughly."; // Однострочный комментарий.
/* Это многострочный комментарий. */
```

Присваивание и равенство

Знак равенства (=) используется в JScript как присваивание. Следующий код

```
anInteger = 3;
```

подразумевает "*Присвоить значение 3 переменной anInteger*," или *"anInteger принимает значение 3."* При сравнении двух значений на равенство применяется двойной знак равенства *(==).*

Выражения

JScript выражения можно разделить на Логические или Числовые. Выражения содержат некоторые особенности, к примеру, символ "+" означает "*добавить к...".* Любая допустимая комбинация значений, переменных, операторов, и других выражений является выражением.

```
var anExpression = "3 * (4 / 5)";
var aSecondExpression = "Math.PI * radius * 2";
var aThirdExpression = aSecondExpression + "%" + anExpression;
var aFourthExpression = "(" + aSecondExpression + ") % (" +
anExpression + ")";
```

Переменные

Переменные используются в Microsoft JScript для присваивания значений в сценариях. Для различия переменных, им присваивают имена.

Объявление переменных

Необязательно, но считается хорошим стилем программирования, объявление переменной перед использованием. Это делается с помощью инструкции var. Вы обязаны применять инструкцию var при объявлении локальной (local) переменной внутри функции. В остальных случаях объявление инструкции var перед применением в сценариях рекомендуется.

Примеры объявления переменных:

```
var mim = "A man, a plan, a canal, Panama!"; // Строковый тип
var ror = 3; // Целый числовой тип.
var nen = true; // Boolean или логический тип.
var fif = 2.718281828 // Числовой тип.
```

Имя переменной

JScript различает регистр в имени переменной: *myCounter* отличается от *MYCounter*. На практике присваивания имен требуется соблюдать следующие правила:

Первым символом может быть буква любого регистра, или символ подчеркивания (_), или знак доллара ($).

Следующими символами могут быть буквы, символы подчеркивания, цифры и знаки доллара.

Именами переменных не могут служить зарезервированные слова.

Несколько примеров правильных имен:

```
pagecount
Part9
Number_Items
```

Некоторые неправильные имена:

```
99Balloons // Первый символ - цифра.
Smith&Wesson // Амперсанд(&) не разрешено применять в именах.
```

Если необходимо объявить и инициализировать переменную, но без присваивания определенного значения, можно применить значение null.

```
var zaz = null;
var notalot = 3 * zaz; // В данном случае notalot равен 0.
```

Если переменная объявлена, а значение не присвоено, она существует, но значение не определено - *undefined.*

```
var godot;
var waitingFor = 1 * godot; // waitingFor имеет значение NaN, так как значение переменной godot не определено.
```

Разрешается объявление переменной неявно - без инструкции var. Однако, в выражениях применять необъявленные переменные не допускается.

```
lel = ""; // Переменная lel объявлена неявно.
var aMess = vyv + zez; // Ошибка, так как vyv and zez не существуют.
```

Преобразование типов

Так как JScript - язык с нестрогим контролем типов, переменные в JScript не имеют строго фиксированного типа. Переменные имеют тип, эквивалентный типу значения, которое они содержат. Однако, в некоторых случаях, необходимо принудительное преобразование переменной в определенный тип. Числа могут быть объявлены как строки, а строки необходимо преобразовать в числовой тип. Для этого применяют конверсионные функции parseInt() и parseFloat().

```
var theFrom = 1;
var theTo = 10;
var doWhat = "Count from ";
doWhat += theFrom + " to " + theTo + ".";
```

После выполнения кода, переменная *doWhat* принимает значение *"Count from 1 to 10."* Числовой тип преобразовывается в строковый тип.

```
var nowWhat = 0;
nowWhat += 1 + "10"; // В этом случае, "10" является строкой,
// "+=" - оператор конкатенации.
```

После исполнения кода, переменная nowWhat принимает значение "0110". Нижеследующее объясняет выполнение кода:

1. Посмотрите на типы 1 и "10". "10" - строковый, а 1 - числовой, поэтому число было преобразовано в строку.
2. Оператор + над строками, является оператором конкатенации. Результатом является "110".
3. Посмотрите на типы по обе стороны оператора +=. nowWhat включает число, и "110" - строку, и поэтому число преобразуется в строку.
4. В данный момент по обе стороны оператора += находятся строки, происходит конкатенация строк. Результатом является "0110".
5. Результат хранится в переменной nowWhat.

```
var nowThen = 0;
nowThen += 1 + parseInt("10"); // В данном случае, "+=" является оператором сложения
```

После выполнения кода, переменная nowThen принимает значение 11.

Операторы JScript

В JScript применяются множество операторов: арифметические, логические, разрядные, присваивания и прочие. Операторы JScript представлены в таблицах.

Таблица. Операторы JScript

Вычислительные		Логические		Разрядные	
Название	Символ	Название	Символ	Название	Символ
Унарный минус	-	Логическое НЕ	!	Разрядное НЕ	~
Инкремент	++	Меньше	<	Поразрядный левый сдвиг	<<
Декремент	--	Больше		Поразрядный правый сдвиг	>>
Умножение	*	Меньше или равно	<=	Беззнаковый поразрядный правый сдвиг	>>>
Деление	/	Больше или равно	>=	Разрядное И	&
Остаток от деления	%	Равно	==	Разрядное ИСКЛЮЧАЮЩЕЕ ИЛИ	^
Сложение	+	Не равно	!=	Разрядное ИЛИ	|
Вычитание	-	Логическое И	&&		

		Логическое ИЛИ	||		
		Условное выражение (Тринар)	?:		
		Запятая	,		
		Тождественно	===		
		Нетождественно	!==		

Таблица. Операторы JScript

Присваивания		Прочие	
Название	Символ	Название	Символ
Присваивание	=	Удаление	delete
Составное присваивание	OP=	Тип	typeof
Пусто	void		

Приоритет операторов

В JScript операторы выполняются в определенном порядке, называемом приоритет операций. Следующий список отражает приоритет операторов от высшего к низшему. Операторы, указанные в одной строке, выполняются слева направо.

Таблица. Приоритет операций в JScript

Приоритет	**Оператор**	**Описание**
1	. [] ()	Точка, индексы массивов, вызов функции
2	++ -- - ~ ! typeof new void delete	Унарные операции, вывод типов данных, создание объектов, неопределенные значения
3	* / %	Умножение, деление, остаток от деления
4	+ - +	Сложение, вычитание, конкатенация строк
5	<< >> >>>	Поразрядные сдвиги
6	< <= > >=	Меньше, меньше или равно, больше, больше или равно
7	== != === !==	Равно, неравно, тождественно, нетождественно
8	&	Разрядное И
9	^	Разрядное ИСКЛЮЧАЮЩЕЕ ИЛИ
10	|	Разрядное ИЛИ
11	&&	Логическое И
12	||	Логическое ИЛИ
13	?:	Условное выражение
14	= OP=	Операторы присваивания
15	,	Запятая

Круглые скобки используются, чтобы изменить порядок выполнения операторов. Выражение в круглых скобках полностью вычисляется прежде, и его значение используется как остаточный член инструкции. Оператор с более высоким приоритетом выполняется ранее оператора с низким приоритетом. Например:

z = 78 * (96 + 3 + 45)

В данном выражении пять операторов: =, *, (), +, и +. Приоритет операторов - следующий: (), *, +, +, =.

1. Первым вычисляется значение выражения в круглых скобках: К сумме операции 96 и 3 прибавляется 45, общая сумма равна 144.

2. Далее выполняется умножение: произведение 78 и 144 дает результат 11232.
3. Переменной z присваивается значение 11232.

Управление ходом программы

Существует несколько видов проверки условий. Все условные выражения в Microsoft JScript - логические, поэтому результат их проверки равен либо true, либо false. Вы можете проверять значения логического, числового строкового типов данных. В JScript простейшими структурами управления являются условные выражения.

Использование условных выражений

В JScript поддерживаются условные выражения if и if...else. В выражении if проверяется условие, при соответствии этому условию, выполняется написанный разработчиками JScript-код. (В выражении if...else выполняется при несоответствии условию другой код.) Простейшая форма оператора if может быть написана одной строкой. Но обычно операторы if и if...else записываются в несколько строк. Следующий пример демонстрирует синтаксис выражений if и if...else. В первом примере - простейшая логическая проверка. Если выражение в круглых скобках равно true, выражение или блок выражений после if выполняется.

```
// Функция smash() определена в другом месте кода.
if (newShip)
smash(champagneBottle,bow); // Логическая проверка newShip на равенство true.
// В данном примере, условие выполняется, если оба подусловия равны true.
if (rind.color == "ярко-желтый " && rind.texture == "большие и малые пятна")
{
theResponse = ("Это тыква? <br> ");
}
// В следующем примере, условие выполнится, если хотя бы одно из подусловий (или оба) равны true.
var theReaction = "";
if ((lbsWeight > 15) || (lbsWeight > 45))
{
theReaction = ("Неплохо! <br>");
}
else
theReaction = ("Не очень! <br>");
```

Условный оператор

В JScript также поддерживается упрощенная форма условия. Это использование вопросительного знака после условия для проверки (подобно if до условия), и двух альтернативных выражений, одно применяется при успешной проверки, другое - при несоответствии условию. Альтернативные выражения разделяются двоеточием.

```
var hours = "";
// Код для проверки времени
// theHour, or theHour - 12.
hours += (theHour >= 12) ? " PM" : " AM";
```

Циклы

Существует несколько вариантов выполнения инструкции или блоков инструкций неоднократно. Повторное выполнение называется циклом. Цикл обычно управляется исходя из значения некоторых условий и(ли) переменных, значения которых меняется в конце каждого цикла. В Microsoft JScript применяется несколько типов циклов: for, for...in, while, do...while и switch.

Применение цикла for

В выражении for определены переменная-счетчик, условие проверки и действие, изменяющее значение счетчика. После каждого выполнения цикла (это называется

итерацией цикла), проверяется условие, производится обработка выполнения действий на переменной-счетчиком, и если условия проверки выполнены, выполняется новая итерация цикла. Если условие для выполнения цикла никогда не будет выполнено, цикл никогда не завершится. Если проверяемое условие всегда выполняется, цикл бесконечен. Поэтому разработчикам следует позаботиться об этом.

```
/*Изменение выражения (" icount ++ " в приведенных примерах)
выполняется в конце цикла, после блока инструкций,
прежде, чем условие проверено.*/
var howFar = 11; // Ограничение выполнения цикла - 11 итераций.
var sum = new Array(howFar); // Создание массива размер - 11,
//индексы от 0 по 10.
var theSum = 0;
sum[0] = 0;
for(var icount = 1; icount < howFar; icount++) { // Счетчик от 1 до
// 10 в данном случае.
theSum += icount;
sum[icount] = theSum;
}
var newSum = 0;
for(var icount = 1; icount > howFar; icount++) { // Цикл не будет
// выполнен.
newSum += icount;
}
var sum = 0;
for(var icount = 1; icount > 0; icount++) { // Бесконечный цикл.
sum += icount;
}
```

Применение цикла for...in

В JScript используется специальный цикл для пошагового обхода свойств объекта. Счетчик цикла for...in проходит все индексы массива. Это строковый тип, а не числовой.

```
for (j in tagliatelleVerde) // tagliatelleVerde является объектом с
//несколькими свойствами
{
// Код JScript .
}
```

Применение цикла while

Цикл while напоминает цикл for. Различие в том, что в цикле while отсутствует встроенная переменная-счетчик, а, следовательно, и условия ее изменения. Выражения над условием для выполнения цикла while находятся внутри цикла в блоке инструкций.

```
var theMoments = "";
var theCount = 42; // Инициализация переменной-счетчика.
while (theCount >= 1) {
if (theCount > 1) {
theMoments = "Осталось " + theCount + " секунд!";
}
else {
theMoments = "Осталась одна секунда!";
}
theCount--; // изменение счетчика.
}
theMoments = "Время истекло!";
```

Применение выражений break и continue в циклах

В Microsoft JScript существует инструкция остановки выполнения цикла. Оператор завершения break может использоваться, чтобы остановить цикл, при выполнении какого-либо условия. Инструкция continue используется, чтобы немедленно перейти к выполнению следующей итерации, пропуская остальную часть выполнения кода текущей итерации, но обновляя переменную-счетчик как в обычных циклах for или for...in.

```
var theComment = "";
var theRemainder = 0;
var theEscape = 3;
var checkMe = 27;
for (kcount = 1; kcount <= 10; kcount++)
{
theRemainder = checkMe % kcount;
if (theRemainder == theEscape)
{
break; // выход при выполнении условия (theRemainder ==
//theEscape).
}
theComment = checkMe + " divided by " + kcount + " leaves a remainder
of " + theRemainder;
}
for (kcount = 1; kcount <= 10; kcount++)
{
theRemainder = checkMe % kcount;
if (theRemainder != theEscape)
{
continue; // При неравенстве theRemainder и theEscape переходим
//к следующей итерации.
}
// JScript код.
}
```

Функции в JScript

Функции в Microsoft JScript выполняют определенные действия. Они могут возвращать некоторый результат, например, результат вычислений или сравнения Функции исполняют при вызове определенный блок инструкций. Это позволяет однажды определить функцию, а в дальнейшем вызывать ее, когда потребуется. Вы передаете данные функции, включая их в круглые скобки после имени функции. Данные в круглых скобках называются параметрами. В некоторых функциях нет параметров вообще; в некоторых – один параметр; иногда параметров несколько. В JScript имеется два вида функций: встроенные и определяемые.

Специальные встроенные функции

Язык JScript включает несколько встроенных функций. Некоторые из них позволяют обрабатывать выражения и специальные символы, и преобразовывать строки в числовые значения. Например, escape() и unescape() используются для перевода кода HTML и спецсимволов, то есть символов, которые нельзя непосредственно размещать в тексте. К примеру, символы "<" и ">", применяются для обозначения HTML-тэгов. Функция escape принимает в параметре спецсимволы, а возвращает ESC-код символа. Каждый код ESC-код начинается со знака процента (%) и последующим двухзначным числом. Функция unescape является инверсионной. В качестве параметра вводится ESC-код, а возвращает символ. Еще одна встроенная функция - eval(), которая выполняет любое правильное математическое выражение, представленное в виде строки. Функция eval() имеет один аргумент - выражение для выполнения.

```
var anExpression = "6 * 9 % 7";
var total = eval(anExpression); // Вычисляет выражение, равно 5.
var yetAnotherExpression = "6 * (9 % 7)";
total = eval(yetAnotherExpression) // Вычисляет выражение, равно 12.
var totality = eval("Текст."); // Возвращает ошибку.
```

Создание собственных функций

Вы можете создавать собственные функции и вызывать их, когда потребуется. Определение функции состоит из объявления параметров и блока инструкций JScript. Функция checkTriplet в следующем примере принимает в качестве параметров длины трех сторон треугольника, и определяет, является ли треугольник прямоугольным, проверяя согласно Пифагорову правилу (триплету). (Квадрат длины гипотенузы прямоугольного треугольника равен сумме квадратов длин катетов.) Функция checkTriplet в ходе проверки вызывает одну из двух других функций. При проверке значений при входных значениях чисел с плавающей точкой используется "машинное эпсилон" ("epsilon") - очень маленькое число. Из-за погрешности округления в вычислениях с плавающей точкой, без применения этого числа, функция может возвращать неверный результат. "Машинное эпсилон" является максимальной допустимой погрешностью.

```
var epsilon = 0.0000000000001; // "Машинное эпсилон".
var triplet = false;
function integerCheck(a, b, c) { // Функция проверки при целых числах.
if ( (a*a) == ((b*b) + (c*c)) ) { // Код проверки.
triplet = true;
}
} // Конец функции проверки при целых числах.
function floatCheck(a, b, c) { // Функция проверки при числах с
//плавающей точкой.
var theCheck = ((a*a) - ((b*b) + (c*c))) // Контрольное число.
if (theCheck < 0) { // Вычисление модуля контрольного числа.
theCheck *= -1;
}
if (epsilon > theCheck) { // Сравнение отклонения с допустимой
//погрешностью!
triplet = true;
}
} // Конец функции проверки при числах с плавающей точкой.
function checkTriplet(a, b, c) { // Проверка триплета. Первым делом,
// присвоим "a" наибольшее значение из трех входных чисел.
var d = 0; // Временная переменная.
if (c > b) { // При c > b, меняем местами.
d = c;
c = b;
b = d;
} // Иначе - не меняем.
if (b > a) { // При b > a, меняем местами.
d = b;
b = a;
a = d;
} // Иначе - не меняем.
// Сторона "a" является гипотенузой.
if (((a%1) == 0) && ((b%1) == 0) && ((c%1) == 0)) { // Проверка
// чисел - целые ли они?
integerCheck(a, b, c); // Функция для целых чисел.
```

```
}
else
floatCheck(a, b, c); // Иначе вызываем функцию для чисел с
//плавающей точкой
} // Окончание проверки правила Пифагора (триплета).
// Практическая проверка - объявляем три переменные.
var sideA = 5;
var sideB = 5;
var sideC = Math.sqrt(50);
checkTriplet(sideA, sideB, sideC); // Функция проверки на триплет.
```

Объекты JScript

В Microsoft JScript объекты по сути являются совокупностями методов и свойств. Метод - функция, которая выполняется внутри объекта, а свойство - значение или набор значений (в виде матрицы или другого объекта), являющееся частью объекта. В JScript объекты можно разделить на три вида: встроенные, созданные и браузерные. В JScript, обработка объектов и массивов идентична. Вы можете обратиться к любой части объекта (его свойствам и методам) либо по имени, либо по индексу. Нумерация индексов в JScript начинается с нуля. Для удобства работы, частям можно присвоить имена. Существует несколько вариантов обращений. Следующие выражения эквивалентны.

```
theWidth = spaghetti.width;
theWidth = spaghetti[3]; // [3] является индексом "width".
theWidth = spaghetti["width"];
```

При использовании числовых индексов, в обращении по имени нельзя использовать точку (.). Следующее выражение вызывает ошибку.

```
theWidth = spaghetti.3;
```

В случаях, когда объект является свойством другого объекта, обращение образуется прямым путем.

```
var init4 = toDoToday.shoppingList[3].substring(0,1); // массив
// shoppingList - свойство toDoToday.
```

Так как объекты могут быть свойствами других объектов, возможно создание массивов с более чем одним измерением, которые непосредственно не поддерживаются. Следующий код создает таблицу умножения для значений от 0 до 16.

```
var multTable = new Array(17); // Создание оболочки для таблицы
for (var j = 0; j < multTable.length; j++) { // Подготовка к заполнению
//строками
var aRow = new Array(17); // Создание строки
for (var i = 0; i < aRow.length; i++) { // Подготовка к заполнению
// строки
aRow[i] = (i + " times " + j + " = " + i*j); // Создание и размещение
//одного значения
}
multTable[j] = aRow; // Заполнение таблицы строкой
}
```

Обращение к каждому элементу производится указанием нескольких индексов

```
var multiply3x7 = multTable[3][7];
```

Следующее выражение вызовет ошибку.

```
var multiply3x7 = multTable[3, 7];
```

Зарезервированные слова JScript

В JScript имеется ряд зарезервированных ключевых слов. Эти слова разделяются на три типа: зарезервированные ключевые слова JScript, слова, зарезервированные для будущих версий, и слова, которые следует избегать.

Таблица. Ключевые слова JScript

Break	false	in	this	void
Continue	for	new	true	while
Delete	function	null	typeof	with
Else	if	return	var	

Таблица. Зарезервированные слова для будущих версий JScript

Case	debugger	export	super
Catch	default	extends	switch
Class	do	finally	throw
Const	enum	import	try

К словам, которых следует избегать, относятся имена уже существующих встроенных объектов и функций. Например, в языке уже применяются слова String и parseInt. Применение любого из ключевых слов первых двух типов вызовет ошибку трансляции, когда ваш сценарий начнет выполняться. Использование зарезервированных слов третьего вида может вызвать непредсказуемое исполнение сценария, если Вы попытаетесь применить вашу переменную и вызвать объект с тем же именем в одном сценарии. Например, следующий сценарий не выполняет того, что нужно:

var String = "Новая строка";
var text = new String("Эта строка - новый экземпляр объекта");

В этом случае получим ошибку, выводящую информацию о том, что строка не является объектом.

Использование массивов

Массивы в JScript разряжены. Так, если в массиве три значения пронумерованы как 0, 1 и 2, вы можете создать элемент 50, не волнуясь об элементах от 3 до 49. Если массив имеет автоматическую переменную размера (смотрите встроенные объекты (Intrinsic Objects) для определения размера массива), переменная размера установлена как 51, несмотря на размер 4. Конечно, можно создавать массивы без разрывов в нумерации элементов, но это не обязательное условие. Фактически в JScript, массивы вообще могут не иметь нумерации. В JScript объекты и массивы идентичны друг другу. Настоящая разница не в данных, а скорее в адресации элементов массива или свойств (properties) и методов объекта. Существует два основных способа для адресации элементов массива. Обычно, для адресации используют индексы. Индексы содержат числовое значение или выражение (expression), которое оценивается как неотрицательное целое. В следующем примере предполагается, что переменная entryNum определена, и ей присвоено значение в другом месте сценария.

theListing = addressBook[entryNum];
theFirstLine = theListing[1];

Метод адресации эквивалентен методу адресации объектов, хотя при адресации объекта, индекс должен быть именем существующего свойства. Если такого свойства нет, возникает ошибка при исполнении кода.

Второй способ адресации массива состоит в том, чтобы создать объект (массив), который содержит свойства, которые пронумерованы числами в цикле. В следующем примере создается два массива, один для имени, второй для адреса, внесенных в список addressBook. Каждый из них содержит четыре свойства. Образец theName, например, формируется от [Name1] до [Name4] свойств theListing, может содержать "G." "Edward" "Heatherington" "IV" или "George" "" "Sand" "".

theListing = addressBook[entryNum];
for (i = 1; i < 4; i++) {
theName[i] = theListing["Name" + i];
theAddress[i] = theListing["Address" + i];
}

В то же время код мог бы легко быть написан в "dot"-стиле системы обозначений (то есть адресуя theListing, theName и theAddress скорее, как объекты, чем матрицы, через точку), но подобное не всегда возможно. Иногда какое-либо свойство не может существовать до времени выполнения, или нельзя его узнать заранее. Например, если массив addressBook упорядочен по фамилии вместо нумерации, пользователь будет вероятно вводить названия " на лету", в то время как сценарий функционирует, просматривая людей. Следующий пример показывает применение определений функций в другом месте сценария.

```
theListing = addressBook[getName()];
theIndivListing = theListing[getFirstName()];
```

Это ассоциативная адресация массива, то есть адресация посредством полностью произвольных строк. Объекты в JScript являются ассоциативными массивами. Хотя чаще всего используется "dot"-стиль, это не всегда требуется.

Организация диалога с пользователем

Благодаря тому, что при написании скриптов WSH, используется либо VBScript, либо JScript (довольно мощные языки программирования), появляется возможность создавать сценарии, позволяющие получить от пользователя какую-либо информацию, влияющую на процесс работы сценария. Эта важная возможность помогает создавать более гибкие и функциональные сценарии, предусматривающие различные нужды пользователей. Получить информацию от пользователя можно с помощью диалогового окна или строки ввода информации. Рассмотрим, для начала диалоговые окна. Ниже приведен простой скрипт, выводящий пример диалогового окна и выдающий сообщение о выборе пользователя:

```
// Диалоговое окно. JScript
// http://www.whatis.ru
var WSHShell = WScript.CreateObject("WScript.Shell");
// Подготовка переменных для диалогового окна
var vbOKCancel = 1;
var vbInformation = 64;
var vbCancel = 2;
var Message = "Пример создания диалогового окна";
var Title = "Нажмите OK или Cancel";
// Вызов диалогового окна
var intDoIt;
intDoIt=WSHShell.Popup(Message,0,Title, vbOKCancel+vbInformation );
// Результат выбора пользователя
WScript.Echo(intDoIt);
```

Вызов диалогового окна осуществляется с помощью метода Popup объекта WSHShell. Первым параметром передается текст, выводимый в диалоговом окне, третьим - заголовок окна, четвертым - набор кнопок и иконка в диалоговом окне. Вот на последнем и остановимся подробнее. Каждому набору кнопок соответствует цифровая переменная:

0 - ОК;
1 - ОК и Отмена;
2 - Прервать, Повтор, Пропустить;
3 - Да, Нет, Отмена;
4 - Да, Нет;
5 - Повтор, Отмена;
6 - Отмена, Повторить, Продолжить.

Для лучшей читаемости кода удобнее определить переменную с названием, отражающим набор кнопок, в начале сценария, как это сделано в примере, а непосредственно при вызове диалогового окна использовать не цифру, а эту переменную. Аналогично наборам кнопок, иконки в диалоговом окне определяются с помощью цифровой переменной. Четвертый параметр метода Popup представляет собой сумму переменных набора кнопок и иконки,

выводимых в диалоговом окне. Так, если вам надо вывести иконку вопроса и кнопки Да, Нет, Отмена, нужно передать в параметр 32 + 3, т.е. 35. Создавать диалоговое окно мы научились, теперь неплохо бы узнать какой выбор сделал пользователь. В примере результат выбора в диалоговом окне сохраняется в переменной intDoIt, а потом выводится на экран. Всем кнопкам диалогового окна соответствует числовое значение, которое и возвращается при выборе одной из них. Полный список приведен ниже:
1 - ОК;
2 - Отмена;
3 - Прервать;
4 - Повтор;
5 - Пропустить;
6 - Да;
7 - Нет;
10 - Повторить;
11 - Продолжить.
Таким образом, получив результат выбора пользователя, можно предусмотреть несколько вариантов работы скрипта. Однако, только кнопками Да, Нет, Отмена и т.п. не всегда можно обойтись. Например, как узнать у пользователя каталог, куда он хочет сохранить какой-то файл? Или букву диска, куда подмонтировать сетевой ресурс? В таких случаях поможет строка ввода информации (InputBox). Ниже приведен простой сценарий, демонстрирующий работу такого диалогового окна.

```
' Диалоговое окно. VBScript
' http://www.whatis.ru
Dim s,s1
s1="Введите ваше имя"
' Выводим диалоговое окно со строкой ввода на экран
s=InputBox(s1,"Пример получения данных от пользователя")
' Результат ввода
MsgBox "Вас зовут " & s
```

Дальше уже можно обрабатывать введенную информацию в своем сценарии. Вот только есть один небольшой подводный камень: InputBox присутствует только в VBScript. А как быть, если вам надо написать сценарий на языке JScript? В WSH есть возможность объединять несколько сценариев, написанных на одном или разных языках, в один файл. Для этого служат wsf-файлы. Тема эта заслуживает отдельной статьи, но если кратко, то это XML файл, имеющий определенную структуру, в которой, каждый сценарий помещается в отдельный элемент XML, и может обращаться к функциям и переменным из других сценариев.

Задания к лабораторной работе

Задание 1. Основы работы с JScript
Создайте сценарий JScript, выполняющий следующие действия:
1) Рассчет значений функции tg(x) в промежутке [0,1], с шагом 0.0001;
2) Вывод значений в консоль.
Задание 2. Пользовательские функции и массивы
Создайте сценарий JScript, выполняющий следующие действия:
1) формирование матрицы (NxN) вида:
1 7 7 7… 7
4 1 7 7 …7
4 4 1 7 …7
…
4 4 4 4… 1
2) вычисление количества элементов кратных 7;
3) вычисление суммы элементов матрицы в каждой нечетной строке;
4) вычисление произведения четных элементов в каждом нечетном столбце;
5) вычисление суммы элементов выше побочной диагонали;
6) сортировка каждой четной строки матрицы по возрастанию, каждой нечетной – по убыванию;
7) вычисление обратной матрицы для исходной;
8) умножение исходной матрицы на сортированную;
9) вывод результата умножения в одну строку.
Каждый пункт реализуйте в виде отдельной пользовательской функции. Организуйте вывод результатов работы функций в консоль.
Задание 3. Рекурсия
Создайте сценарий JScript, рассчитывающий суму n первых чисел ряда при помощи рекурсивной функции. Ряд задается рекуррентно:
$F(1)=1,$
$F(i)=\ln(F(i-1)*i)+9$