Знакомство с JS

Язык программирования JavaScript («JS») — это объектно-ориентированный язык сценариев, объекты которого встроены в программное обеспечение веб-браузера, например, Google Chrome, Microsoft Edge, Firefox, Opera и Safari. Для обеспечения функциональности это позволяет при загрузке страницы в браузере интерпретировать содержащиеся на веб-странице сценарии. В целях безопасности язык JavaScript не может считывать или записывать файлы, за исключением файлов cookie, в которых хранится минимальный объем данных.

Самая первая реализация JavaScript была создана Бренданом Эйхом (Айком) (Brendan Eich) в компании Netscape. Этот язык программирования был впервые представлен в декабре 1995 года и первоначально назывался «LiveScript». Однако вскоре из коммерческих соображений LiveScript был переименован в JavaScript, получив соответствующую лицензию у Sun Microsystem.

Брендан Эйх, создатель языка программирования JavaScript, а также соучредитель проекта Mozilla, помог запустить веб-браузер Firefox.

До введения JavaScript браузер вызывал сценарии на стороне сервера, а в связи с долгим ответом снижалась производительность. Эта проблема решилась с помощью вызова сценариев на стороне клиента.

Популярность языка JavaScript быстро росла. Но между компаниями Netscape и Microsoft возникли разногласия по поводу его лицензирования, поэтому Microsoft представила собственную версию языка под названием «JScript». Несмотря на то что новая версия JScript очень похожа на JavaScript, она имеет некоторые расширенные функции. В июне 1997 года Ecma International предложила стандартизировать JavaScript, и в результате появился «ECMAScript». Это помогло стабилизировать основные функции. Однако название не прижилось среди юзеров, поэтому язык программирования все же получил название JavaScript.

Добавление JavaScript в HTML-документ

Вы можете добавить JavaScript-код в HTML-документ, поместив его между тегами **<script>** и **</script>**, например:

```
<script>
document.getElementById( 'message' ).innerText = 'Hello World!'
</script>
```

HTML-документ может включать в себя несколько сценариев, которые помещаются в раздел заголовка или тела документа. Однако рекомендуется размещать сценарии в конце основного раздела (непосредственно перед закрывающим тегом **</body>**), чтобы браузер мог отобразить веб-страницу до выполнения сценария.

Также в HTML-документ можно добавить JavaScript код, расположенный во внешнем файле с расширением **.js**. Это позволяет нескольким различным веб-страницам вызывать один и тот же сценарий. Подключение внешнего сценария выполняется с помощью атрибута **src** тега **<script>** следующим образом:

```
<script src="external_script.js"> </script>
```

Этот код также можно поместить в раздел заголовка или тела документа, и браузер будет рассматривать сценарий так, как если бы он был написан непосредственно в этой позиции в HTML-документе.

Если атрибуту src тега **<script>** присваивается только имя файла внешнего сценария, необходимо, чтобы файл сценария находился в одной папке (каталоге)

В теге <script> применим атрибут type="text/javascript". Однако это не требуется, поскольку JavaScript — язык сценариев для HTML по умолчанию.

Не включайте теги <script> и </script> во внешний файл JavaScript. В них необходимо добавлять только код сценария.

Внешние файлы сценариев упрощают сопровождение кода. Почти все примеры, представленные в этой книге, автономные, поэтому код сценария можно вставлять между тегами в HTML-документе.

с HTML-документом. Если сценарий находится в другом месте, вы можете указать относительный путь к файлу, например:

```
<script src="js/external_script.js"> </script>
```

Если сценарий расположен в другом месте сайта, можно указать абсолютный путь к файлу, например:

```
<script src="https://www.example.com/js/external_script.js">
</script>
```

Кроме того, можно указать содержимое, которое будет отображаться на веб-странице только в том случае, если пользователь в своем веб-браузере отключил JavaScript, включив в HTML-документ элемент **<noscript>**, например:

```
<noscript>JavaScript is Not Enabled!</noscript>
```

Вывод JavaScript

В JavaScript существует несколько вариантов отображения данных. Например:

```
document.getElementById( 'message' ).innerText = 'Hello World!'
```

Обратите внимание на использование оператора. (точка) в описании свойств или методов объекта с использованием «точечной нотации».

Атрибут **id** определяет уникальный идентификатор HTML-элемента. Свойство **innerText** определяет содержимое HTML-документа.

Также для отображения данных можно использовать всплывающее окно сообщений.

```
window.alert( 'Hello World!' )
```

Метод **alert( )** выводит на экран диалоговое окно с сообщением, указанным в коде в круглых скобках ().

В процессе изучения языка JavaScript предпочтительнее выводить данные в консоль браузера, например:

```
console.log( 'Hello World!' )
```

Метод **log( )** объекта **console** выводит в окно консоли содержимое, указанное в круглых скобках ( ). Фактически в любом браузере есть специальная панель веб-разработчика, доступ к которой осуществляется нажатием клавиши F12. Поскольку Google Chrome — самый популярный браузер на момент написания книги, я использовал его для демонстрации JavaScript, а консоль этого браузера используется для отображения данных.

1. Создайте HTML-документ, содержащий пустой абзац и программный код для отображения данных тремя способами.

```
<p id="message"></p>
<script>
document.getElementById( 'message' ).innerText =
                                        'Hello World!'
window.alert( 'Hello World!' )
console.log( 'Hello World!' )
</script>
```

2. Сохраните HTML-документ, затем откройте его в браузере, чтобы проанализировать полученный результат, как показано на рисунке ниже.

3. Нажмите клавишу **F12** или воспользуйтесь меню браузера, чтобы открыть **Developers Tools** (Инструменты разработчика).

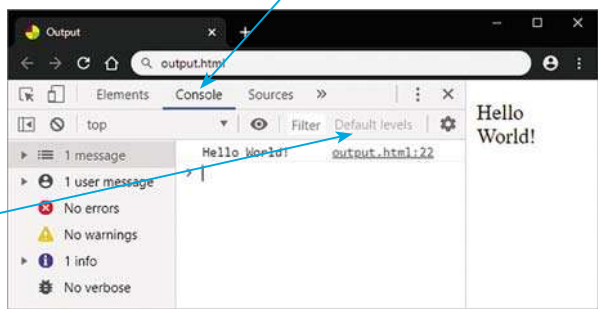
Существует также метод **document.write( )**, который перезаписывает весь заголовок и тело веб-страницы. Стоит отметить, что его использование считается плохой практикой.

При возникновении ошибок в коде консоль предоставляет полезные сообщения, что очень удобно для отладки программы.

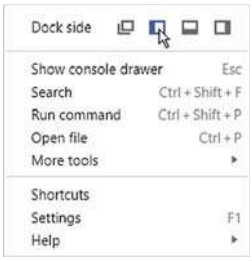
output.html

Убедитесь, что в консоли отображается содержимое, а также имя HTML-документа и номер строки, где есть соответствующий JavaScript-код.

4 Выберите вкладку **Console** (Консоль), чтобы увидеть содержимое, записанное в консоли.



5 Нажмите кнопку **Show/Hide** (Показать/Скрыть), чтобы скрыть или показать боковую панель. Нажмите кнопку **Customize** (Настроить), чтобы выбрать способ закрепления консоли в окне браузера. Затем нажмите кнопку **Clear** (Очистить), чтобы очистить все содержимое консоли.



Структура кода

JavaScript-код состоит из ряда инструкций, называемых «операторами». Обычно все инструкции на странице выполняются сверху вниз.

Каждая инструкция может содержать любой из следующих элементов:

- Ключевые слова — слова, имеющие особое значение в языке JavaScript.
- Операторы — специальные символы, выполняющие операции с одним или несколькими операндами.
- Значения — текстовые строки, числа, логическое значение **true** или **false**, значения **undefined** и **null**.

- Выражения — любая часть исходного кода программы, которая вычисляет значение.

Ранее в коде JavaScript каждый оператор должен был заканчиваться символом ; (точка с запятой), а каждое предложение — символом . (точка) в английской раскладке. Теперь это необязательно, поэтому, если вы не хотите писать несколько операторов в одной строке, то это правило можно опустить. В таком случае операторы необходимо разделять точкой с запятой, например:

```
оператор; оператор; оператор
```

Некоторые разработчики JavaScript по-прежнему предпочитают заканчивать каждый оператор символом ; (точка с запятой). В приведенных в книге примерах он опускается, но выбор остается за вами.

Интерпретатор JavaScript игнорирует отступы и пробелы. Поэтому, чтобы улучшить читабельность кода, необходимо использовать пробелы. Например, при сложении двух чисел:

```
total = 100 + 200 вместо total=100+200
```

Операторы JavaScript обычно сгруппированы с помощью фигурных скобок { }, разделяющих код на функциональные блоки, которые можно при необходимости многократно вызывать. Для удобства и читабельности кода рекомендуется делать отступы двумя пробелами, например:

```
{
  оператор
  оператор
  оператор
}
```

Синтаксис JavaScript — это набор правил, по которым строится программа. В JavaScript существуют два типа значений: фиксированные и переменные. Фиксированные числовые и текстовые строковые значения называются литералами:

Результат вычисления выражения — это значение, тогда как оператор выполняет действие.

Для удобства и читабельности кода создавайте отступы с помощью клавиши Пробел, так как при отладке кода в текстовых редакторах отступы, созданные клавишей Tab, могут обрабатываться по-разному.

Для использования строковых литералов рекомендуется выбрать один вид кавычек и придерживаться его. В наших примерах используются одинарные кавычки.

- Числовые литералы — целые числа, например 100, или числа с плавающей запятой, например 3,142.
- Строковые литералы — строка символов, заключенная в двойные кавычки, например **"JavaScript Fun"**, или одинарные кавычки, например **'JavaScript Fun'**.

Переменные — это некий контейнер, внутри которого можно хранить значения для дальнейшего использования в программе. В JavaScript предусмотрен способ объявления переменных с помощью ключевого слова **let**. Например, с помощью кода **let total** создается переменная с именем «total». Переменной можно присвоить значение, используя оператор присваивания **=**, например:

```
let total = 300
```

Прочие операторы JavaScript используются для создания выражений, которые будут вычислять одно значение. Как правило, выражение заключается в круглые скобки **( )**. Например, приведенное ниже выражение содержит числа и оператор сложения **+** и вычисляет одно значение 100:

```
( 80 + 20 )
```

Также выражения могут содержать значения переменных. Например, для вычисления одного значения 100 выражения могут содержать предыдущее значение переменной, оператор вычитания и число:

```
( total – 200 )
```

Иногда возникает необходимость закомментировать строки кода, чтобы предотвратить их выполнение при отладке.

JavaScript чувствителен к регистру символов, поэтому переменные с именами **total** и **Total** — это две совершенно разные переменные.

Хорошей практикой считается добавление к коду пояснительных комментариев. Они делают код более понятным для других пользователей, а также для вас при его повторном просмотре. Все, что находится в одной строке после символа **//** или между символов **/*** и ***/** в одной или нескольких строках, будет проигнорировано.

```
let total = 100 // Этот код БУДЕТ выполнен.
/* let total = 100
   Этот код НЕ БУДЕТ выполнен. */
```

Избегайте ключевых слов

В программе для переменных и функций вы можете указывать собственные имена. Лучше всего давать переменным осмысленные имена, которые будут отражать суть переменной или функции. Имя переменной может содержать буквы, цифры и символы подчеркивания, но никогда не начинается с цифры. Также в нем не используются пробелы. Запрещается использовать в качестве имен переменных приведенные ниже в таблице ключевые слова, которые имеют особое значение:

Ключевые (зарезервированные) слова JavaScript			
abstract	arguments	await	boolean
break	byte	case	catch
char	class	const	continue
debugger	default	delete	do
double	else	enum	eval
export	extends	false	final
finally	float	for	function
goto	if	implements	import
in	instanceof	int	interface
let	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Объекты, свойства и методы JavaScript			
Array	Date	eval	function
hasOwnProperty	Infinity	isFinite	isNaN
isPrototypeOf	length	Math	NaN
name	Number	Object	prototype
String	toString	undefined	valueOf

HTML-имена, объекты окна и свойства			
alert	all	anchor	anchors
area	assign	blur	button
checkbox	clearInterval	clearTimeout	clientInformation
close	closed	confirm	constructor
crypto	decodeURI	decodeURIcomponent	defaultStatus
document	element	elements	embed
embeds	encodeURI	encodeURIcomponent	escape
event	fileUpload	focus	form
forms	frame	innerHeight	innerWidth
layer	layers	link	location
mimeTypes	navigate	navigator	frames
frameRate	hidden	history	image
images	offscreenBuffering	open	opener
option	outerHeight	outerWidth	packages
pageXOffset	pageYOffset	parent	parseFloat
parseInt	password	pkcs11	plugin
prompt	propertyIsEnum	radio	reset
screenX	screenY	scroll	secure
select	self	setInterval	setTimeout
status	submit	taint	text
textarea	top	unescape	untaint
window			

Атрибуты событий в HTML Например:			
onclick	ondblclick	onfocus	onfocusout
onkeydown	onkeypress	onkeyup	onload
onmousedown	onmouseup	onmouseover	onmouseout
onmousemove	onchange	onreset	onsubmit

Хранение значений

Переменная — это контейнер, общий для каждого языка сценариев и языка программирования, в котором хранятся данные и в процессе работы могут быть извлечены. В отличие от строго типизированных переменных в большинстве других языков программирования, которые должны декларировать определенный тип данных, переменные JavaScript в использовании намного проще. JavaScript — язык со слабой типизацией, поэтому переменные могут содержать любой тип данных:

Тип данных	Пример	Описание
String	'Hello World!'	Последовательность символов
Number	3.142	Целое число или число с плавающей запятой
Boolean	true	Значение true (1) или false (0)
Object	console	Пользовательский или встроенный объект
Function	log( )	Пользовательская функция, встроенная функция или метод объекта
Symbol	Symbol( )	Уникальный идентификатор
null	null	Отсутствие какого-либо объектного значения
undefined	undefined	Пустое значение

Имя переменной — это псевдоним для значения. С помощью него можно ссылаться на ее сохраненное значение.

Для объявления переменных в JavaScript используются ключевые слова **let**, **const** или **var**, за которыми следует пробел и выбранное вами имя. По мере выполнения программного кода ключевое слово **let** позволяет присваивать переменным новые значения, тогда как **const** (константа) не предполагает никаких изменений. Ключевое слово **var** использовалось в JavaScript до введения ключевого слова **let**. Однако сейчас его лучше избегать, так как оно позволяет дважды объявлять одну и ту же переменную в одном и том же контексте.

Для удобства рекомендуется выбирать понятные и осмысленные имена переменных.

Объявление переменной через **let** позволяет в программе создать переменную, значение которой может быть присвоено позже. Директива **let** объявляет переменную с блочной областью видимости с возможностью инициализировать ее значением.

```
let myNumber                    // Объявление переменной.
myNumber = 10                   // Инициализация переменной.
let myString = 'Hello World!'   // Объявление и инициализация переменной.
```

Также в одной строке можно объявить несколько переменных:

```
let i, j, k                     // Объявление трех переменных.
let num = 10, char = 'C'        // Объявление и инициализация двух переменных.
```

Однако константы должны быть инициализированы во время объявления:

```
const myName = 'Mike'
```

После инициализации JavaScript автоматически устанавливает тип переменной для присвоенного значения. Последующее назначение другого типа данных выполняется в программе позже, чтобы изменить тип данных. Текущий тип переменной можно узнать по ключевому слову **typeof**.

variables.html

1. Создайте HTML-документ. Объявите и проинициализируйте несколько переменных разного типа.

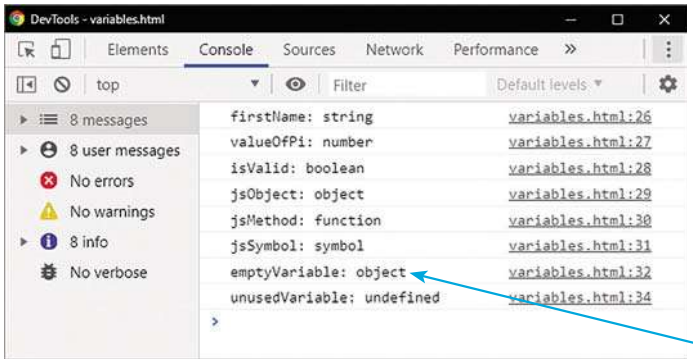
```
const firstName = 'Mike'
const valueOfPi = 3.142
let isValid = true
let jsObject = console
let jsMethod = console.log
let jsSymbol = Symbol( )
let emptyVariable = null
let unusedVariable
```

2. Для вывода типа данных каждой переменной добавьте операторы:

```
console.log( 'firstName: ' + typeof firstName )
console.log( 'valueOfPi: ' + typeof valueOfPi )
console.log( 'isValid: ' + typeof isValid )
console.log( 'jsObject: ' + typeof jsObject )
console.log( 'jsMethod: ' + typeof jsMethod )
console.log( 'jsSymbol: ' + typeof jsSymbol )
console.log( 'emptyVariable: ' + typeof emptyVariable )
console.log( 'unusedVariable: ' + typeof unusedVariable )
```

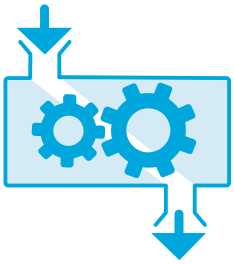
Оператор конкатенации + в данном примере используется для вывода объединенной текстовой строки.

3 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте полученные результаты — типы данных.



Вы будете удивлены, увидев, что переменная, которой присвоено значение null, описывается как тип object, а не тип null. Это распространенная ошибка языка JavaScript.

Создание функций



Объявление функции состоит из одного или нескольких операторов, сгруппированных вместе в фигурные скобки { }. Функция вызывает свои инструкции и возвращает в результате единственное значение. Функции могут вызываться по требованию программы. Чтобы отличать встроенные функции от определяемых пользователем, функции, которые являются свойством объекта, например **console.log()**, называются методами. Встроенные и определяемые пользователем функции содержат завершающие круглые скобки, которые могут принимать значения аргумента. Например, аргумент, переданный в скобках метода **console.log( )**.

Количество аргументов, передаваемых функции, как правило, должно соответствовать количеству

Обратите внимание, что в предпочтительном формате объявления функции открывающая фигурная скобка {находится в той же строке, что и ключевое слово function.

параметров, указанных в скобках. Например, определяемая пользователем функция, требующая только один аргумент, выглядит так:

```
function имя функции ( параметр ) {
  // Здесь будет ваш код.
}
```

Функция может иметь несколько параметров, которые указываются через запятую в скобках. При необходимости вы можете указать значение по умолчанию, которое будет использоваться, когда вызов функции не передает аргумент, например:

```
function имя функции ( параметр, параметр = значение ) {
  // Здесь будет ваш код.
}
```

Выбирая собственные имена параметров, необходимо следовать тем же правилам, что и при выборе имен переменных. Имена параметров можно использовать в функции для ссылки на значения аргументов, переданных при вызове функции.

Вы можете опустить оператор return или использовать ключевое слово return без указания значения, тогда функция в результате вернет значение undefined.

Функциональный блок может включать оператор **return**. Он предназначен для возвращения значения выражения в качестве результата выполнения функции. Как только выполнение программы доходит до этого места, функция останавливается, и значение возвращается в вызвавший ее код.

```
function имя функции ( параметр, параметр = значение ) {
  // Здесь будет ваш код.

  return результат
}
```

В функциональном блоке также может находиться вызов других функций.

functions.html

1 Создайте HTML-документ. Создайте функцию, возвращающую значения переданного аргумента, возведенного в квадрат (во вторую степень).

```
function square ( arg ) {
  return arg * arg
}
```

2 Добавьте функцию, возвращающую результат сложения.

```
function add ( argOne, argTwo = 10 ) {
  return argOne + argTwo
}
```

3 Теперь добавьте функцию, возвращающую результат возведения в квадрат и сложения, как показано ниже.

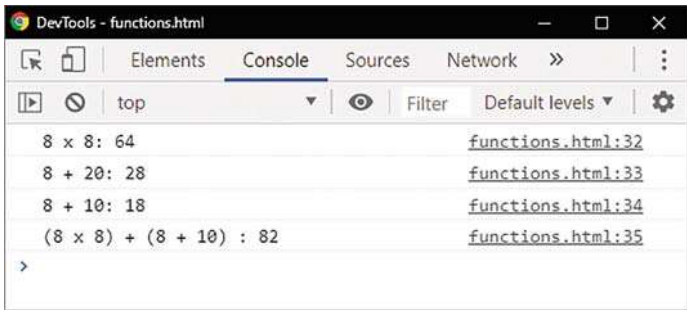
```
function squareAdd ( arg ) {
  let result = square( arg )
  return result + add( arg )
}
```

4 Добавьте операторы, которые вызывают функции и выводят возвращаемые значения на экран.

```
console.log( '8 x 8: ' + square( 8 ) )
console.log( '8 + 20: ' + add( 8, 20 ) )
console.log( '8 + 10: ' + add( 8 ) )
console.log( '(8 x 8) + (8 + 10): ' + squareAdd( 8 ) )
```

Обратите внимание, что в нашем случае используется значение второго параметра по умолчанию (10), когда при вызове функции передается только одно значение аргумента.

5 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте полученный результат — возвращаемые функциями значения.



Символ * — это оператор арифметического умножения в JavaScript.

Назначение функций

При присвоении переменной именованной функции указывайте в операторе только имя функции.

Функции — важный инструмент в JavaScript. Это некоторый фрагмент кода, который можно описать один раз, а затем вызывать на выполнение в разных частях программы любое число раз.

Важно понимать, что оператор ( ) — это компонент оператора вызова, который фактически вызывает функцию. Это означает, что оператор может назначить функцию переменной, указав только имя функции. Затем переменную можно использовать для вызова функции. Однако будьте внимательны, если вы попытаетесь назначить функцию переменной, указав ее имя, за которым следует символ ( ). Такая функция будет вызвана, и будет присвоено значение, возвращаемое этой функцией.

Переменные, которые были объявлены с использованием прежнего ключевого слова var, также были подняты. Переменные, которые были объявлены с помощью ключевых слов let или const, не поднимались.

Поднятие

Хотя код читается интерпретатором JavaScript сверху вниз, на самом деле он выполняет два этапа. На первом этапе ищутся объявления функций и запоминается все, что находится в процессе, то есть поднятие. Второй этап — это когда код фактически выполняется интерпретатором.

Поднятие позволяет вызовам функций появляться в коде до объявления функции, так как интерпретатор уже распознал функцию на первом этапе. Однако на первом этапе не распознаются функции, которые были присвоены переменным с помощью ключевых слов **let** или **const**!

Анонимные функции

При присвоении функции переменной имя функции можно не указывать, так как ее можно вызвать в операторе, указав имя переменной и оператор ().
Такие функции называются анонимными функциональными выражениями. Их синтаксис выглядит следующим образом:

let *переменная* = function (*параметры*) { *операторы* ;
return *значение*}

Анонимные функциональные выражения также можно сделать самовызывающимися, заключив функцию целиком в круглые скобки ( ) и добавив в конце выражения оператор ( ). Это означает, что при первой загрузке сценария браузером операторы автоматически выполняются один раз. Синтаксис функционального выражения с автоматическим запуском выглядит так:

```
( function ( ) { операторы ; return значение } ) ( )
```

В приведенных в книге примерах для выполнения кода при загрузке скрипта обычно используются самовызывающиеся функции.

Самовызывающиеся функциональные выражения также известны как немедленно вызываемые функции (IIFE — часто произносится как «iffy»).

1 Создайте HTML-документ. Вызовите функцию, которая еще не была объявлена.

```
console.log( 'Hoisted: ' + add( 100, 200 ) )
```

anonymous.html

2 Добавьте функцию, которая вызывается выше.

```
function add( numOne, numTwo ) {
  return numOne + numTwo
}
```

3 Добавьте функцию, которая назначает указанную выше функцию переменной, а потом вызывает назначенную функцию.

```
let addition = add
console.log( 'Assigned: ' + addition( 32, 64 ) )
```

4 Назначьте аналогичную, но анонимную функцию переменной и вызовите назначенную функцию.

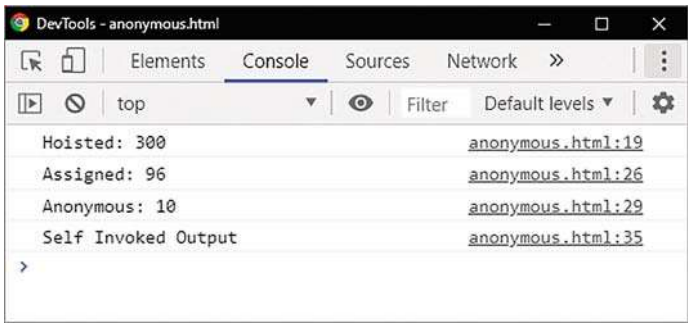
```
let anon = function ( numOne, numTwo ) {
  let result = numOne + numTwo ; return result
}
console.log( 'Anonymous: ' + anon( 9, 1 ) )
```

Смысл самовызывающихся функций может быть не сразу понятен, но к концу этой главы их важность станет более ясной и очевидной.

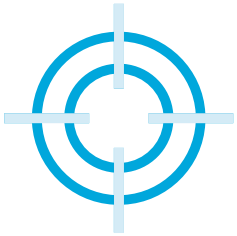
5 Присвойте значение, возвращаемое самовызывающейся функцией, переменной и отобразите полученное значение.

```
let iffy = ( function ( ) {
  let str = 'Self Invoked Output' ; return str
} ) ( )
console.log( iffy )
```

6 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте полученные результаты — возвращаемые функцией значения.



Область видимости



Все переменные в JavaScript имеют определенную область видимости, в пределах которой они могут действовать. Таких областей две — глобальная и локальная.

Глобальная область видимости

Переменная или функция, созданная в этой области видимости, доступна из любой точки программы. Это означает, что переменные существуют постоянно и доступны для функций в одной области. На первый взгляд это может показаться очень удобным, однако имеется серьезный недостаток, заключающийся в том, что одноименные переменные могут конфликтовать. Представьте, что вы создали глобальную переменную **myName**, которой вами было присвоено имя, но затем подключился внешний код, в котором другой разработчик создал глобальную переменную **myName** с таким же именем. Обе переменные существуют в одной области программы, то есть конфликтуют. Таких ситуаций лучше избегать, поэтому для хранения примитивных значений

не рекомендуется создавать глобальные переменные (всех типов данных, за исключением **Object** и **Function**).

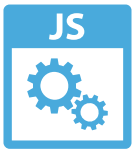
Локальная область видимости

Переменные, созданные внутри функциональных блоков, доступны локально на протяжении всего цикла функции. Они существуют только во время ее выполнения, а затем удаляются. Их область программы ограничена — от места, в котором они были созданы, до последней фигурной скобки} или момента возврата из функции. Рекомендуется объявлять переменные в начале функционального блока, чтобы их лексическая область видимости соответствовала времени жизни функции. Это означает, что переменные с одинаковыми именами могут существовать внутри отдельных функций без создания конфликта. Например, локальная переменная **myName** может успешно существовать в программе внутри отдельных функций и внутри функций во включенных внешних кодах. В программе для хранения значений рекомендуется создавать только локальные переменные.

Наиболее эффективный вариант

Объявление глобальных переменных с использованием ключевого слова **var** позволяет конфликтующим переменным с одинаковыми именами перезаписывать присвоенные им значения без предупреждения. В этом случае в результате использования более новых ключевых слов **let** и **const** выдается ошибка **Uncaught SyntaxError**. Поэтому в программе для хранения значений рекомендуется создавать переменные, объявленные с использованием ключевых слов **let** или **const**.

1 Создайте внешний код, в котором вызывается функция для вывода значения глобальной переменной.



external.js

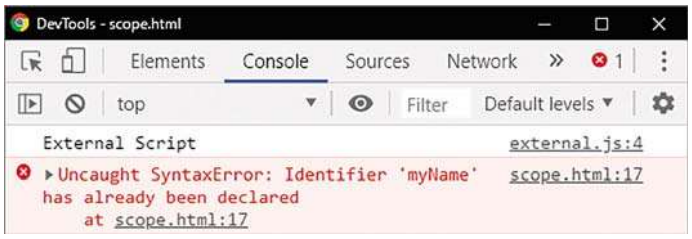
```
let myName = 'External Script'
function readName( ) {console.log( myName ) }
readName( )
```

scope.html

2 Создайте HTML-документ, который подключает внешний код и добавляет аналогичный.

```
<script src="external.js"></script>
<script>
let myName = 'Internal Script'
function getName( ) {console.log( myName ) }
getName( )
</script>
```

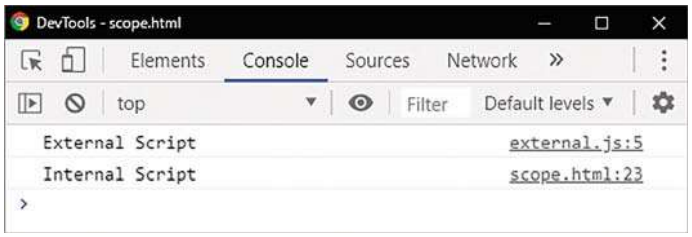
3 Сохраните оба файла в одной папке, затем откройте HTML-документ, чтобы увидеть сообщение об ошибке конфликта в консоли.



4 Отредактируйте оба скрипта, чтобы преобразовать глобальные переменные в локальные. Во избежание конфликта обновите браузер.

```
function readName( ) {
  let myName = 'External Script'; console.log  (
  myName )
}

function getName( ) {
  let myName = 'Internal Script'; console.log (
  myName )
}
```



Вызовы функций readName( ) и getName( ) остаются в коде без редактирования.

Замыкания

На предыдущем примере была продемонстрирована опасность создания глобальных переменных для хранения значений в JavaScript. Однако возникает необходимость сохранить значения, которые остаются постоянно доступными, — например, чтобы запоминать увеличение счетчика в ходе выполнения программы. Как это сделать без использования глобальных переменных для хранения примитивных значений? Ответ заключается в использовании замыканий.

Замыкание — это комбинация функции и лексического окружения вместе со всеми доступными внешними переменными. В JavaScript замыкания создаются каждый раз при создании функции, во время ее создания.

closure.html

1. Создайте HTML-документ с кодом, который присваивает глобальной переменной самовызывающуюся анонимную функцию.

```
const add = ( function ( ) {
  // Здесь будет ваш код.
} ) ( )
```

2. Добавьте операторы для инициализации локальной переменной и назначения функции локальной переменной в той же области.

```
let count = 0
const nested = function ( ) { return count = count + 1
}
```

3. Теперь добавьте оператор для возврата внутренней функции — присвоения внутренней функции глобальной переменной.

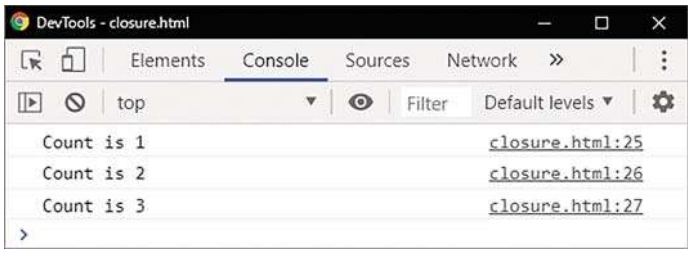
```
return nested
```

4. Наконец добавьте три идентичных вызова функций к внутренней функции, которая теперь назначена глобальной переменной.

```
console.log( 'Count is ' + add( ) )
```

```
console.log( 'Count is ' + add( ) )
console.log( 'Count is ' + add( ) )
```

5 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте полученные результаты — значения, возвращенные при закрытии.



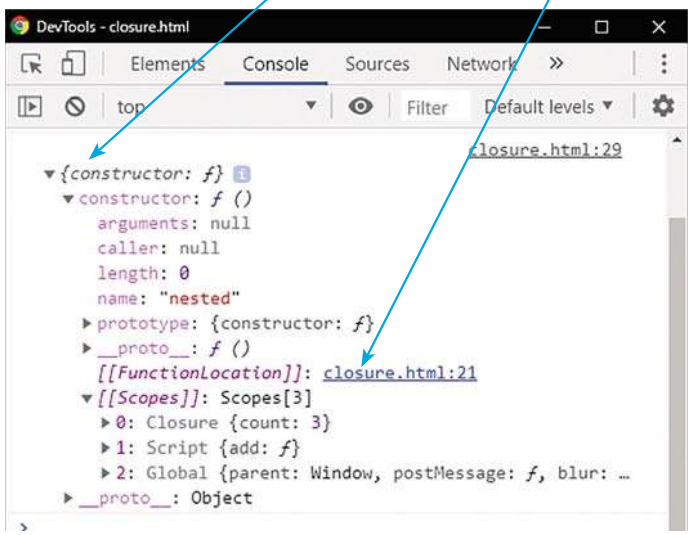
Все объекты JavaScript наследуют свойства и методы от свойства prototype. Стандартные объекты JavaScript, такие как функции, для создания объекта вызывают функцию внутреннего конструктора.

Понять концепцию замыканий непросто — может показаться, что переменная **count** в нашем примере должна быть удалена, когда самовызывающаяся функция завершит выполнение. Чтобы разобраться, как работают замыкания, вы можете изучить свойство **prototype**.

6 В конце кода добавьте следующий оператор.

```
console.log( add.prototype )
```

7 Сохраните HTML-документ, затем обновите браузер и разверните раскрывающийся список constructor, чтобы увидеть области.



Не беспокойтесь, если вы сразу не сможете понять, как работают замыкания. Сначала это покажется сложным, но со временем все станет гораздо проще и понятнее. Вы можете продолжить изучение, а затем вернуться к этому инструменту.

Присмотритесь внимательнее, и вы обнаружите, что назначенная функция имеет особую область видимости (замыкание) в дополнение к обычной локальной (скрипт или код) и внешней (глобальной) области. Таким образом, переменная **count** остается доступной через назначенную функцию. Однако на нее невозможно ссылаться каким-либо другим способом.

Считается важным использование замыканий для сокрытия постоянных переменных от других областей кода. Точно так же частные переменные могут быть скрыты в других языках программирования и доступны только при использовании методов чтения.

Заключение

- JavaScript-код может быть включен в HTML-документ напрямую или из внешнего файла с помощью тегов **<script> </script>**.
- JavaScript позволяет отображать результаты вывода в диалоговом окне предупреждения или в окне консоли браузера.
- Операторы JavaScript могут содержать ключевые слова, операторы, значения и выражения.
- Интерпретатор JavaScript игнорирует отступы и пробелы.
- Операторы JavaScript могут быть сгруппированы в функциональные блоки с помощью фигурных скобок {}, которые при необходимости вызываются для выполнения.
- Имена переменных и функций могут состоять из букв, цифр и символов подчеркивания, но запрещается использовать в качестве имен переменных ключевые слова.
- Переменные JavaScript могут содержать следующие типы данных: строка (string), число (number), булев или логический тип (boolean),

объекты (object), функция (function), символы (symbol), значение (null) и специальное значение (undefined).

- Переменным, объявленным с использованием ключевого слова **let**, можно дать новые значения. Ключевое слово const не предполагает изменений.

- Объявление функции состоит из одного или нескольких операторов, сгруппированных вместе в фигурные скобки { }. Функция вызывает свои инструкции и возвращает в результате единственное значение.

- В функциональном выражении в круглых скобках ( ) могут содержаться параметры значений аргументов, передаваемых от вызывающей стороны.

- Функциональный блок может включать оператор return для указания данных, которые должны быть переданы обратно вызывающей стороне.

- Оператор ( ) вызывает функцию.

- Поднятие позволяет вызовам функций появляться в коде до объявления функции.

- Функция может быть анонимной, то есть не иметь имени.

- Лексическая область видимости — это область, где была создана переменная, которая может быть глобальной, локальной или замыканием.

- Замыкание — это функция, вложенная во внешнюю функцию, которая сохраняет доступ к переменным, объявленным во внешней функции.