Преобразование типов

Во избежание непредвиденных результатов перед выполнением операций в JavaScript важно определить типы данных значений, с которыми вы работаете. Например, значение **42** — это число, а значение **'42'** — строка, поэтому в результате сложения **'42' + 8** получится строковое выражение **'428'**, а не число **50**. JavaScript предоставляет несколько способов преобразования типов данных без изменения исходного значения.

Преобразование строки в число

Встроенная функция **parseInt()** принимает строку в качестве аргумента и возвращает целое число в соответствии с указанным основанием системы счисления. Например, **parseInt('42')** вернет число **42**, поэтому в результате сложения **42 + 8** получится число **50**.

Встроенная функция **parseFloat()** принимает строку в качестве аргумента и возвращает десятичное число (число с плавающей запятой).

Оба метода позволяют буквенным символам следовать за числовой частью указанной строки и удалять их из результата — например, **parseInt('42nd Street')** в результате возвращает число **42**.

Если в начале указанной строки ни одна из этих функций не находит числовое значение, результатом будет **NaN** (Not-A-Number) — свойство JavaScript, означающее «не число». С помощью функции **isNaN( )** вы можете проверить, является ли переменная или литерал нечисловым значением. В начале указанного значения она попытается найти число и вернет **false**, если число обнаружено (даже в указанной строке). В противном случае, если этого не произойдет, в результате вернется значение **true**.

Преобразование числа в строку

Метод **String( )** – строковое представление числа, указанного в круглых скобках, например, **String(42)** вернет в результате строковый тип данных **'42'**.

Преобразование или приведение типов данных может быть явным или неявным. Например, ‹42› + 8 в результате выполнения возвращает строку ‹428› — это неявное приведение. Метод String(42) возвращает в результате строку '42' — это явное приведение.

Чтобы вернуть строковое представление сохраненного числового типа данных, к имени переменной можно добавить вызов метода **toString()**. Например, если переменной с именем «num» был присвоен номер, метод **num.toString()** в результате вернет строковое представление этого сохраненного числа.

conversion.html

1 Создайте HTML-документ с самовызывающейся анонимной функцией, объявите и проинициализируйте три переменные.

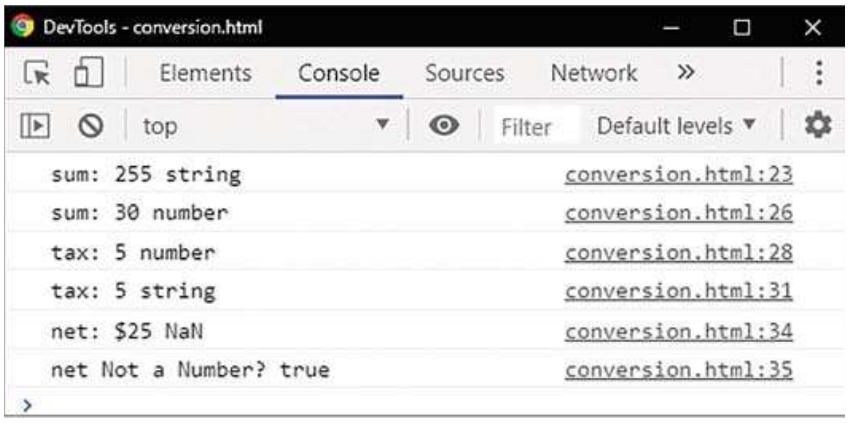
```
( function ( ) {
  let sum, net = '25', tax = 5.00
  // Здесь будет ваш код.
} ) ( )
```

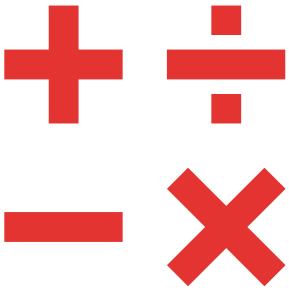
2 Добавьте операторы, которые создают разные типы данных и выводят результат в выходной строке.

```
sum = net + tax
console.log( 'sum: ' + sum + ' ' + typeof sum )
sum = parseFloat( net ) + tax
console.log( 'sum: ' + sum + ' ' + typeof sum )
console.log( 'tax: ' + tax + ' ' + typeof tax )
tax = tax.toString( )
console.log( 'tax: ' + tax + ' ' + typeof tax )
net = '$' + net
console.log( 'net: ' + net + ' ' + parseInt( net ) )
console.log( 'net Not a Number? ' + isNaN( net ) )
```

Если вы вызовете функцию isNan(net) до того, как к строке будет добавлен префикс '$', результат будет false (ложь), так как в начале строки функция находит число '25'.

3 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте результаты вывода.





Арифметические операторы

Перед вами наиболее распространенные в JavaScript арифметические операторы:

Оператор	Описание
+	Оператор сложения возвращает сумму числовых операндов или объединяет строки
–	Оператор вычитания
*	Оператор умножения
/	Оператор деления
%	Остаток от деления
++	Инкремент
– –	Декремент
**	Возведение в степень

Оператор возведения в степень ** возвращает результат первого операнда, возведенного в степень.

Данные, обрабатываемые сценарием JavaScript, называются операндами. Например, в выражении **5+2** оператору **+** передаются значения операндов **5** и **2**. Обратите внимание, что в зависимости от типа операндов оператор **+** выполняет два типа операций. Числовые операнды добавляются для получения суммы чисел, однако при сложении строковых операндов в результате возвращается объединенная строка.

Оператор **%** возвращает целый остаток от деления левого операнда на правый. Деление на **2** вернет либо **1**, либо **0**.

Пример использования оператора % с нечетными или четными числами можно найти на стр. 45.

Оператор инкремент **++** и оператор декремент увеличивают и уменьшают значение операнда на единицу соответственно и возвращают новое значение. Эти операторы чаще всего используются для подсчета итераций цикла двумя разными способами. Если форма операции постфикс (оператор после операнда), значение операнда возвращается, а затем увеличивается или уменьшается на единицу. Если применяется префиксная форма (оператор перед операндом), значение операнда возвращается увеличенным или уменьшенным на единицу.

1 Создайте HTML-документ с самовызывающейся функцией.

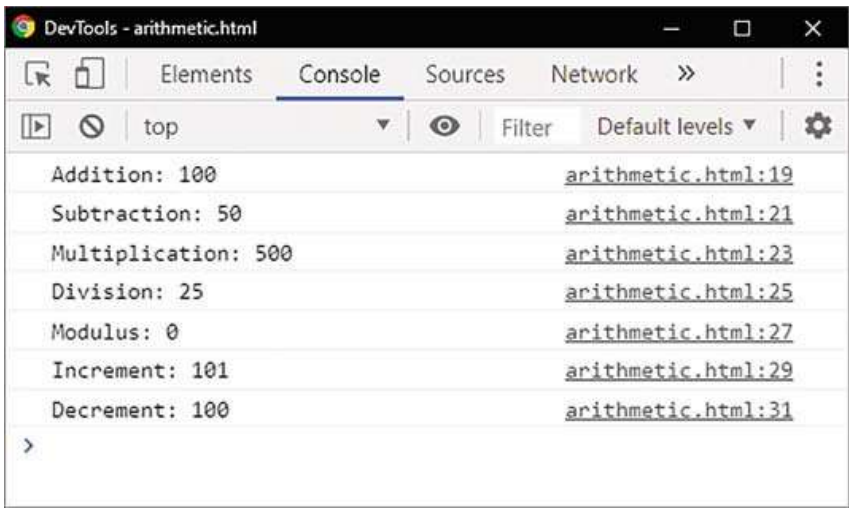
arithmetic.html

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

2 Добавьте операторы, которые присваивают значения переменным, используя каждый арифметический оператор, и проанализируйте результат.

```
let sum = 80 + 20 ; console.log( 'Addition: ' + sum )
let sub = sum - 50 ; console.log( 'Subtraction: ' + sub )
let mul = sum * 5 ; console.log( 'Multiplication: ' + mul )
let div = sum / 4 ; console.log( 'Division: ' + div )
let mod = sum % 2 ; console.log( 'Modulus: ' + mod )
let inc = ++sum ; console.log( 'Increment: ' + inc )
let dec = --sum ; console.log( 'Decrement: ' + dec )
```

3 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте результаты вывода — значения, полученные в результате выполнения арифметических операций.



Операторы присваивания

Ниже в таблице представлены наиболее распространенные в JavaScript операторы присваивания. Все операторы, кроме =, — это сокращенная форма более длинных выражений, поэтому каждому оператору приводится эквивалент.

Оператор	Пример	Эквивалент
=	a = b	a = b
+=	a += b	a = (a + b)
–=	a –= b	a = (a – b)
*=	a *= b	a = (a * b)
/=	a /= b	a = (a / b)
%=	a %= b	a = (a % b)
**=	a **= b	a = (a ** b)

Оператор равенства === сравнивает значения. Более подробная информация об операторах сравнения приведена на стр. 40.

Важно понимать, что оператор = означает «присваивать», а не «равно». Для сравнения оператор равенства в JavaScript выглядит следующим образом: ===.

На примере в таблице оператор присваивания = присваивает переменной a значение, содержащееся в переменной b, и в результате возвращается новое сохраненное значение.

Оператор присваивания += считается наиболее полезным и может использоваться для добавления новой строки к существующей. На следующем примере `let str='JavaScript'` и `str+='Fun'` мы видим, что теперь переменная хранит объединенную строку `'JavaScript Fun'`.

На примере в таблице оператор присваивания += добавит значение, содержащееся в переменной a, к значению, содержащемуся в переменной b, а затем назначит итоговую сумму, чтобы она стала новым значением, сохраненным в переменной a.

Все остальные комбинированные операторы присваивания работают аналогично. Каждый из них сначала выполняет арифметическую операцию с двумя

операндами, а затем присваивает результат этой операции первой переменной, и она становится ее новым сохраненным значением.

assignment.html

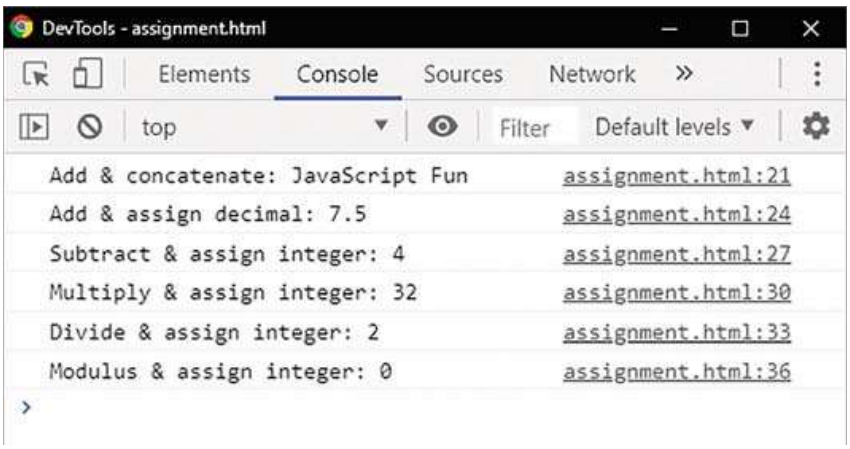
1 Создайте HTML-документ с самовызывающейся анонимной функцией, объединяющей две строки.

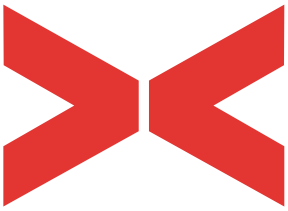
```
( function ( ) {
  let msg = 'JavaScript' ; msg += ' Fun'
  console.log( 'Add & concatenate: ' + msg )
  // Здесь будет ваш код.
} ) ( )
```

2 Добавьте операторы, использующие для выполнения арифметических операций комбинированные операторы, и выведите результат.

```
let sum = 5.00 ; sum += 2.50
console.log( 'Add & assign decimal: ' + sum )

sum = 8 ; sum -= 4
console.log( 'Subtract & assign integer: ' + sum )

sum = 8 ; sum *= 4
console.log( 'Multiply & assign integer: ' + sum )
sum = 8 ; sum /= 4
console.log( 'Divide & assign integer: ' + sum )

sum = 8 ; sum %= 4
console.log( 'Modulus & assign integer: ' + sum )
```

3 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте вывод — результат выполнения операторов присваивания.





Операторы сравнения

Ниже в таблице представлены наиболее распространенные в JavaScript операторы сравнения.

Оператор	Описание
===	Строго равно
!==	Строго не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Пример использования оператора меньше < в структуре цикла можно найти на стр. 61.

Также существуют стандартные оператор равенства == и оператор неравенства !=. Однако они могут привести к непредвиденным результатам. В отличие от операторов строгого равенства, они не гарантируют, что сравниваемые значения относятся к одному типу данных. Например, в результате сравнения 25== '25' возвращается значение true, а в результате сравнения 25=== '25' — false. Для точных сравнений всегда используйте трехсимвольные операторы.

Оператор равенства === сравнивает два операнда и возвращает логическое значение **true** (истина) в том случае, если операнды строго равны. В противном случае возвращается логическое значение **false** (ложь). Если операнды — одинаковые числа, они равны; если операнды представляют собой строки, содержащие одинаковые символы в одинаковых позициях, они равны; если операнды — логические значения, которые оба принимают значение **true** или оба принимают значение **false**, они равны. Оператор не равно!== возвращает значение **true** (истина) в том случае, если операнды не равны, используя те же правила, что и для оператора ===.

Операторы строго равно и строго не равно полезны при сравнении двух значений для выполнения «условного ветвления», когда необходимо выполнить различные действия в зависимости от условий.

Оператор больше > возвращает значение **true** в том случае, если значение левого операнда больше, чем правого. Оператор меньше < возвращает значение **true** в том случае, если значение левого операнда меньше, чем правого. При добавлении символа = после оператора > или оператора < в результате возвращается значение **true** еще в том случае, если два операнда равны.

Операторы больше > и меньше <, как правило, используются для проверки значения переменной счетчика в структуре цикла.

1 Создайте HTML-документ с самовызывающейся анонимной функцией, объявите и проинициализируйте три переменные.

comparison.html

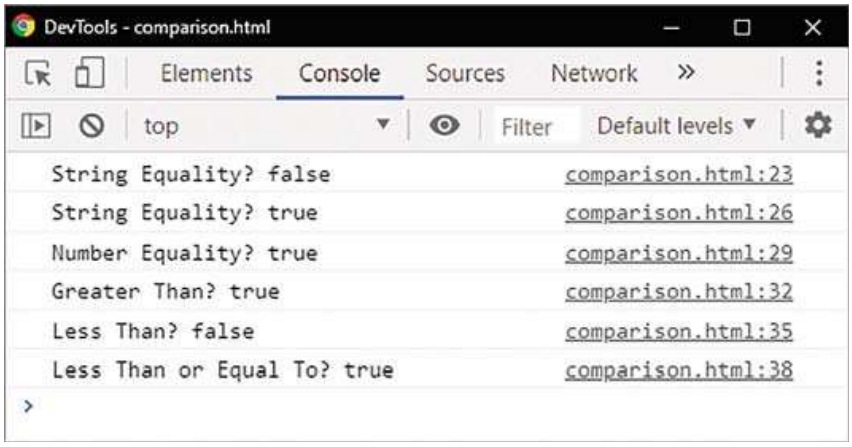
```
( function ( ) {
  let comparison, sum = 8, str = 'JavaScript'
  // Здесь будет ваш код.
} ) ( )
```

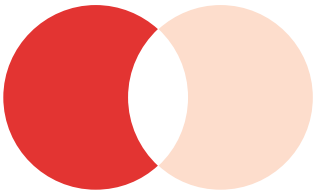
2 Добавьте операторы, использующие операторы сравнения и выведите результат.

```
comparison = str === 'JAVASCRIPT'
console.log( 'String Equality? ' + comparison )

comparison = str === 'JavaScript'
console.log( 'String Equality? ' + comparison )

comparison = sum === 8
console.log( 'Number Equality? ' + comparison )

comparison = sum > 5
console.log( 'Greater Than? ' + comparison )

comparison = sum < 5
console.log( 'Less Than? ' + comparison )

comparison = sum <= 8
console.log( 'Less Than or Equal To? ' + comparison )
```

JavaScript чувствителен к регистру, поэтому регистр символов должен совпадать, чтобы сравниваемые строки были равны.

3 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте вывод — результат выполнения операторов сравнения.



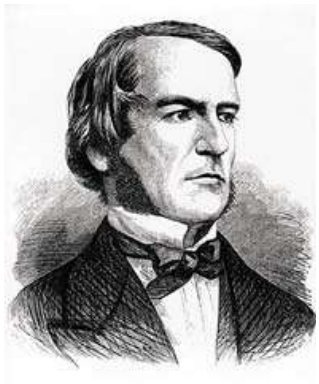


Логические операторы

В JavaScript существуют три логических оператора.

Оператор	Описание
&&	Логическое И
||	Логическое ИЛИ
!	Логическое НЕ

Логические операторы обычно используются вместе с операндами, имеющими логическое значение **true** или **false**, или значениями, которые могут быть в них преобразованы.



Термин «булев» относится к системе логического мышления, разработанной английским математиком Джорджем Булем (1815–1864).

Оператор логическое И **&&** будет оценивать два операнда и возвращать значение **true**, если оба операнда истинны, а иначе — **false**. Это часто используется при условном ветвлении, когда необходимо выполнить различные действия в зависимости от условий.

В отличие от логического оператора **&&**, которому необходимо, чтобы оба операнда были истинными, оператор логическое ИЛИ || находит первое истинное значение и возвращает значение **true**, а иначе — **false**. Это полезно для выполнения сценариев определенного действия в зависимости от условий.

Третий логический оператор — это оператор логического НЕ!, который сначала приводит аргумент к логическому типу **true/false**, а затем возвращает противоположное значение. Например, если переменная с именем «tog» имеет значение **true**, то **!tog** вернет в результате значение **false**. Это полезно для инвертирования значения переменной в последовательных итерациях цикла, например **tog=!a**. А при каждой итерации значение будет меняться на противоположное. Например, включение и выключение светового переключателя.

1 Создайте HTML-документ с самовызывающейся анонимной функцией, объявите и проинициализируйте три переменные.

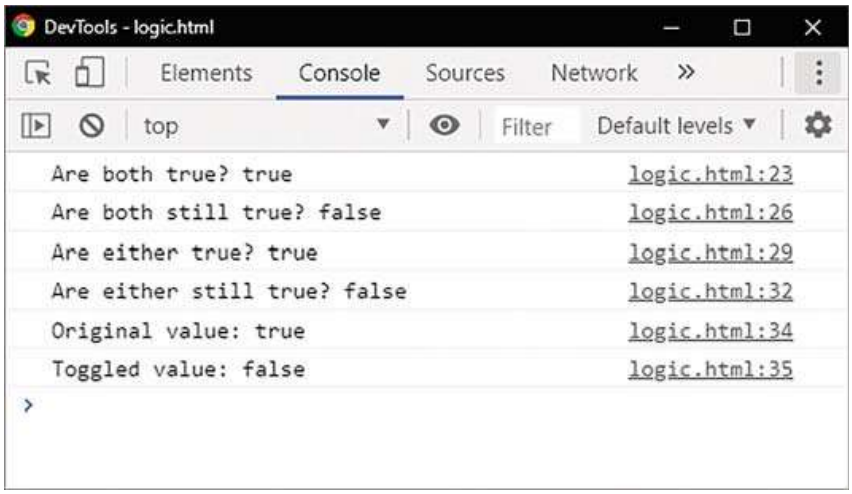
logic.html

```
( function ( ) {
  let result, yes = true, no = false
  // Здесь будет ваш код.
} ) ( )
```

2 Добавьте операторы, использующие логические операторы, и выведите результат.

```
result = yes && yes
console.log( 'Are both true? ' + result )

result = yes && no
console.log( 'Are both still true? ' + result )

result = yes || no
console.log( 'Are either true? ' + result )

result = no || no
console.log( 'Are either still true? ' + result )

console.log( 'Original value: ' + yes )
yes =!yes
console.log( 'Toggled value: ' + yes )
```

3 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте вывод — результат выполнения логических операторов.



Обратите внимание, что выражение false&&false возвращает значение false, а не true, как может показаться, действуя по принципу «два заблуждения — еще не правда».

Условный (тернарный) оператор

Возможно, наиболее полюбившийся оператор создателя JavaScript — это условный оператор?:. **Он также известен как тернарный, то есть состоящий из трех операндов.**

Первый операнд вычисляется и используется как логическое значение. Если он имеет значение true, то вычисляется и возвращается значение выражения во втором операнде. Если же значение **false**, то вычисляется и возвращается значение выражения в третьем операнде. Вычисляется всегда только какой-то один из операндов, второй или третий, и никогда оба. Его синтаксис выглядит так:

условие* ? *если-true-выполнить-это* : *если-false-выполнить-это

Если в зависимости от условий требуется выполнить несколько действий, каждый указанный оператор может быть вызовом функции для выполнения нескольких операторов в каждой из них. Например, вызов функций для выполнения нескольких операторов в соответствии с логическим значением переменной с именем «flag»:

```
flag === true ? doThis( ) : doThat( )
```

В этом примере нет необходимости использовать оператор равенства === и ключевое слово **true**, так как операторы, вычисляющие выражение для логического значения, автоматически выполняют эту проверку. Поэтому пример можно упростить:

```
flag ? doThis( ) : doThat( )
```

В качестве альтернативы два оператора, указанные для тернарного оператора, могут присвоить значение переменной в зависимости от условий проверки, например:

```
flag ? str = 'Go left' : str = 'Go right'
```

Это синтаксически верно, но также можно упростить:

```
str = flag ? 'Go left' : 'Go right'
```

Тернарный оператор состоит из трех операндов. Первый предшествует символу ?, второй — между символами ? и :, третий — после символа :.

Если в условии проверяется четность числового значения, два оператора могут предоставлять варианты в зависимости от того, четное число или нечетное.

1. Создайте HTML-документ с самовызывающейся анонимной функцией, объявите и проинициализируйте две переменные.

```
( function ( ) {
  const numOne = 8, numTwo = 3
  // Здесь будет ваш код.
} ) ( )
```

ternary.html

2. Добавьте операторы для вывода строки, обозначающей количество.

```
let verb = ( numOne !== 1 ) ? ' are ' : ' is '
console.log( 'There' + verb + numOne )
```

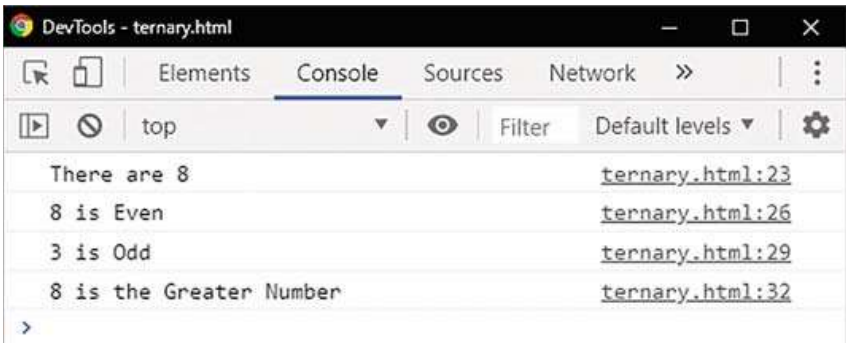
3. Для вывода строк, корректно описывающих четность двух значений переменных, добавьте следующие операторы.

```
let parity = ( numOne % 2 !== 0 ) ? 'Odd' : 'Even'
console.log( numOne + ' is ' + parity )

parity = ( numTwo % 2 !== 0 ) ? 'Odd' : 'Even'
console.log( numTwo + ' is ' + parity )
```

4. Добавьте операторы для вывода строки, сообщающей большее из двух значений переменных.

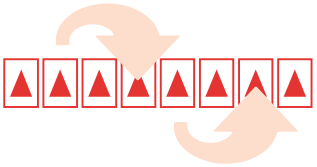
```
let max = ( numOne > numTwo ) ? numOne : numTwo
console.log( max + ' is the Greater Number' )
```

5. Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте результаты вывода.



Тернарный оператор может возвращать значения любого типа данных, будь то строка, число, логическое значение и т. д.

Побитовые операции



Побитовые операторы JavaScript интерпретируют операнды как последовательность из 32 битов (нулей и единиц). Каждый бит передает десятичный компонент только в том случае, если он содержит единицу. Компоненты рассматриваются справа налево от «младшего значащего бита» (LSB) до «старшего значащего бита» (MSB). Ниже представлено двоичное число в восьмибитном представлении — десятичное число 50, что обозначено битами, установленными со значением 1 (2 + 16 + 32 = 50).

Многие авторы книг по программированию на JavaScript не дают информацию по побитовым операторам. Однако полезно понять, что они из себя представляют и как их можно использовать.

Биты	8	7	6	5	4	3	2	1
Десятичное представление	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Двоичное представление	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**

В следующей таблице перечислены все побитовые операторы, используемые в JavaScript.

Оператор	Название	Binary number operation:
|	Побитовое ИЛИ (OR)	Возвращает 1 в тех разрядах, которые хотя бы у одного из операндов были равны 1. Пример: 1010 | 0101 = 1111
&	Побитовое И (AND)	Возвращает 1 в тех разрядах, которые у обоих операндов были равны 1. Пример: 1010 & 1100 = 1000
~	Побитовое НЕ (NOT)	Заменяет каждый бит операнда на противоположный. Возвращает 1, если бит не равен 1, и 0 — если бит равен 1. Пример: ~ 1010 = 0101
^	Побитовое исключающее ИЛИ (XOR)	Возвращает 1 в тех позициях, которые только у одного из операндов были равны 1. Пример: 1010 ^ 0100 = 1110

Байт состоит из 8 бит, и каждая половина байта известна как полубайт (4 бита). Двоичные числа, представленные в примерах таблицы, описывают значения, хранящиеся в полубайте.

Оператор	Название	Binary number operation:
<<	Левый сдвиг	Сдвигает двоичное представление числа на некоторое количество разрядов влево, добавляя справа нули. Пример: 0010 << 2 = 1000
>>	Правый сдвиг, переносящий знак	Сдвигает двоичное представление числа на некоторое количество разрядов вправо. Освобождающиеся разряды заполняются знаковым битом. Пример: 1000 >> 2 = 0010
>>>	Правый сдвиг с заполнением нулями	Сдвигает двоичное представление числа на некоторое количество разрядов вправо. Освобождающиеся разряды заполняются нулями. Пример: 1000 >> 2 = 0010

1 Создайте HTML-документ с самовызывающейся анонимной функцией, объявите и проинициализируйте две переменные.

bitwise.html

```
( function ( ) {
  let numOne = 10, numTwo = 5
  // Здесь будет ваш код.
} ) ( )
```

2 Добавьте операторы для простого вывода строк, подтверждающих начальные значения, хранящиеся в каждой переменной.

```
console.log( 'numOne: ' + numOne )
console.log( 'numTwo: ' + numTwo )
```

3 Добавьте операторы, чтобы с помощью побитовых операций поменять местами значения, хранящиеся в каждой переменной.

```
numOne = numOne ^ numTwo
    // 1010 ^ 0101 = 1111 = (десятичное число 15)
```

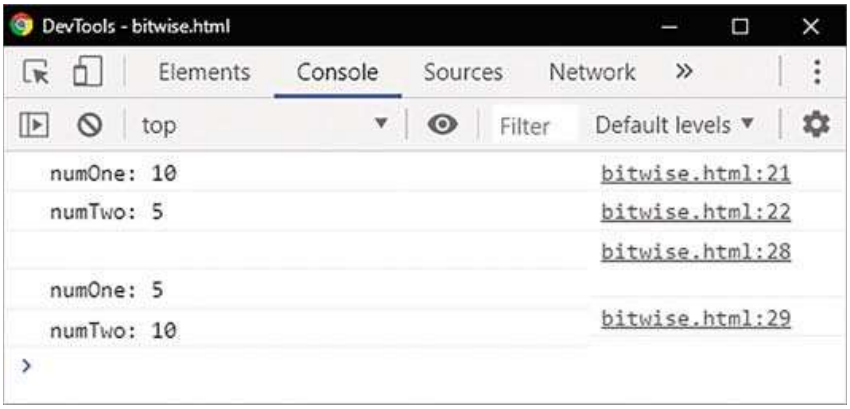
Обратите внимание, каким образом в нашем примере для разрыва строки используется специальная escape-последовательность \n.

```
numTwo = numOne ^ numTwo
    // 1111 ^ 0101 = 1010 (десятичное число 10)
numOne = numOne ^ numTwo
    // 1111 ^ 1010 = 0101 (десятичное число 5)
```

4 Добавьте операторы для вывода разрыва строки и строк, чтобы подтвердить окончательные значения, хранящиеся в каждой переменной.

```
console.log( '\n' + 'numOne: ' + numOne )
console.log( 'numTwo: ' + numTwo )
```

5 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте результаты вывода.



Приоритет операторов

Операторы JavaScript имеют разные уровни приоритета. Приоритет операторов определяет порядок, в котором операторы выполняются. Операторы с более высоким приоритетом выполняются раньше операторов с более низким приоритетом.

В таблице сверху вниз перечислены операторы каждого типа в порядке от самого высокого до самого низкого уровня приоритета:

Оператор	Описание
( )	Группировка
. [ ] ( )	Оператор доступа к объекту Оператор доступа к массиву Вызов функции
++ —	Постфиксный инкремент, постфиксный декремент
++ — ! ~	Префиксный инкремент, префиксный декремент Логическое отрицание, побитовое отрицание
**	Возведение в степень
* /%	Умножение, деление, остаток от деления
+ –	Сложение, вычитание
<< >> >>>	Побитовый сдвиг
< <= => >	Сравнение
=== ==!==!=	Строго равно, равно, строго не равно, не равно
&	Побитовое И
^	Побитовое исключающее ИЛИ
|	Побитовое ИЛИ
&&	Логическое И
||	Логическое ИЛИ
?:	Тернарный оператор
= += –= *= /=%= &= ^= |= <<= >>= >>>=	Операторы присваивания
,	Запятая

Приоритет

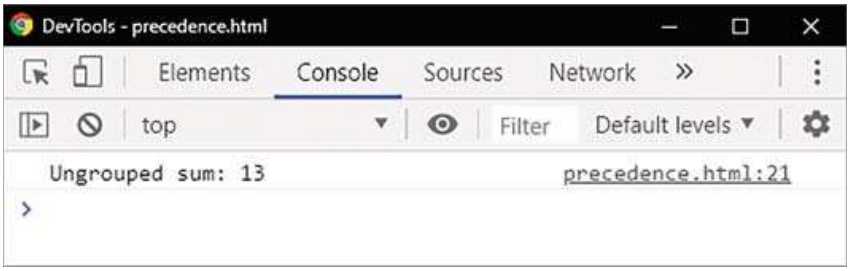
Оператор [ ] описан в разделе о массивах, начиная со стр. 81. Также обратите внимание, что оператор. (точка) используется в точечной нотации, например console.log( ). Он нужен для доступа к свойствам или методам объекта (массива или функции).

1 Создайте HTML-документ с самовызывающейся анонимной функцией, которая инициализирует переменную с результатом разгруппированного выражения и выводит его значение.

precedence.html

```
( function ( ) {
  let sum = 9 + 12 / 3                  // Эквивалентно 9 + 4.
  console.log( 'Ungrouped sum: ' + sum )
} ) ( )
```

2 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте полученные результаты.



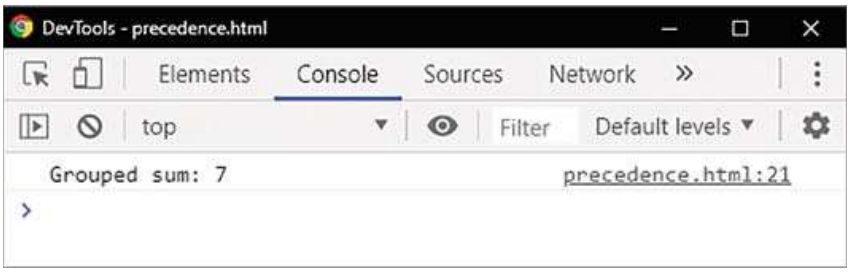
Сначала вычисляется деление, так как оператор деления имеет более высокий приоритет по сравнению с оператором сложения, поэтому результат равен 13. Также вы можете определить порядок приоритета, сгруппировав выражение в круглых скобках, и оно будет вычислено первым, так как оператор ( ) имеет самый высокий приоритет.

Для определения порядка вычислений в выражениях рекомендуется использовать круглые скобки ( ).

3 Чтобы установить порядок вычисления, вы можете изменить операторы в функции, и сложение будет вычисляться перед делением.

```
let sum = ( 9 + 12 ) / 3                // Эквивалентно 21 / 3.
console.log( 'Grouped sum: ' + sum )
```

4 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте результаты вывода.



Заключение

- Функции **parseInt( )** и **parseFloat( )** преобразовывают строки в числа. Функции **String( )** и **toString( )** преобразовывают числа в строки.

- Функция **isNaN( )** определяет, является ли литерал или переменная нечисловым значением **NaN**.

- Арифметические операторы выполняют математические операции: сложение **+**, остаток от деления **%**, инкремент **++**, декремент **—** и возведение в степень ******.

- Если операция используется как постфикс (оператор после операнда), значение операнда возвращается, а затем увеличивается или уменьшается на единицу. Если используется префиксная форма (оператор перед операндом), значение операнда возвращается увеличенным или уменьшенным на единицу.

- Оператор **=** может быть объединен с арифметическим оператором для выполнения арифметической операции, а затем присвоения ее результата.

- Оператор присваивания **+=** считается наиболее полезным и может использоваться для добавления строки к уже существующей.

- Операнды можно сравнивать. Существуют следующие операторы сравнения: строго равно **===**, строго не равно **!==**, больше **>** или меньше **<**.

- Комбинированные операторы сравнения **<=** и **>=** возвращают значение **true**, если оба операнда равны.

- Оператор логическое И **&&** проверяет два операнда и возвращает **true**, если оба истинны. Оператор логическое ИЛИ **||** находит первое истинное значение и возвращает значение **true**, а иначе — **false**.

- Оператор логического НЕ! сначала приводит аргумент к логическому типу **true/false**, а затем возвращает противоположное значение.

- Тернарный оператор **?**: состоит из трех операндов. Первый вычисляется и используется как логическое значение. Условие тернарного оператора задается в первом операнде. Если он имеет значение **true**, то вычисляется и возвращается значение выражения во втором операнде. Если же значение **false**, то вычисляется и возвращается значение выражения в третьем операнде.

- Побитовые операторы JavaScript для выполнения двоичной арифметики могут манипулировать отдельными битами двоичной последовательности.

- Операторы JavaScript имеют разные уровни приоритета. Приоритет операторов определяет порядок, в котором операторы выполняются.

- Для принудительного вычисления определенных частей выражения перед другими используются круглые скобки.