3 Управляющие конструкции в JavaScript

В этой главе вы узнаете об управляющих конструкциях в JavaScript, их назначении и правилах использования.

Ветвление с помощью оператора if

Иногда возникает необходимость выполнить различные действия в зависимости от условий.

В JavaScript проверка определенных условий выполняется с помощью ключевого слова **if**. Условный оператор **if** реализует выполнение конкретных действий при условии, что некоторое логическое выражение принимает значение **true** или **false**. Если результат равен **true**, выполняется одна инструкция, в противном случае — другая.

Необходимо, чтобы проверяемое условие после ключевого слова **if** было заключено в круглые скобки.

Синтаксис оператора **if** выглядит следующим образом:

if (*условие*) *код выполняется, если условие истинно*

Выполняемые инструкции могут быть представлены одной строкой или блоком кода, заключенными в фигурные скобки.

```
if ( условие )
{
   код выполняется, если условие истинно
   код выполняется, если условие истинно
   код выполняется, если условие истинно
}
```

Оценка условий и выполнение действий в соответствии с их результатом — это настоящий мыслительный процесс. Например, запрограммируем действия, которые вы можете выполнить в летний день:

Рекомендуется заключать в фигурные скобки даже отдельные операторы — для сохранения согласованного стиля.

```
let temperature = readThermometer( )
const tolerable = 25

if ( temperature > tolerable )
{
   turn_on_air-conditioning( )
   get_a_cool_drink( )
   stay_in_shade( )
}
```

Проверка условий эквивалентна следующему синтаксису **if(условие === true)**. Но поскольку JavaScript автоматически преобразует результат выражения в логическое значение, т. е. выполняет проверку равенства для истинного значения, нет необходимости включать в скобки оператор **===true**.

1. Создайте HTML-документ с самовызывающейся функцией, которая начинается с инициализации логической переменной.

```
let flag = true
```

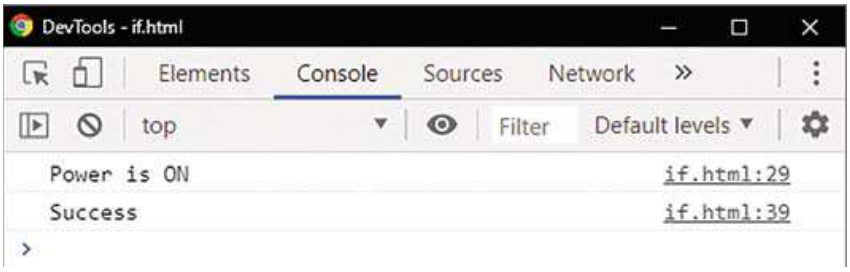
2. Вставьте операторы для выполнения условий логического значения переменной.

```
if( !flag )
{
  console.log( 'Power is OFF' )
}
if( flag )
{
  console.log( 'Power is ON' )
}
```

3. Вставьте операторы для выполнения условий выражения, сравнивающего два целых числа.

```
if( 7 < 2 )
{
  console.log( 'Failure' )
}
if( 7 > 2 )
{
 console.log( 'Success' )
}
```

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



if.html

В этом примере оператор NOT используется для инвертирования проверки условий, чтобы он стал эквивалентным if (flag===false).

Код сценария, создающий функциональный блок, в нашем примере опущен. Более подробную информацию по созданию анонимных самовызывающихся функций вы можете прочитать на стр. 24.

Альтернативное ветвление

Оператор **if**, который проверяет условие для логического значения и выполняет свои операторы только тогда, когда результат равен **true**, обеспечивает единственную ветвь для выполнения действий. Альтернативная ветвь, по которой действия могут быть выполнены при ложном результате, обозначается ключевым словом **else**.

После оператора **if** следует оператор **else**, например:

```
if ( условие ) выполняется, если условие истинно
else выполняется, если условие ложно
```

Оператор **if else** может также выполнять несколько инструкций, если заключить их в фигурные скобки, например:

```
if ( условие )
{
  выполняется, если условие истинно
  выполняется, если условие истинно
}
else
{
  выполняется, если условие ложно
  выполняется, если условие ложно
}
```

Как только условие в операторе **if else** оказывается равным **true**, выполняются связанные с ним операторы, затем поток продолжается после оператора **if else** — без проверки последующих операторов **else**.

Иногда возникает необходимость проверить несколько вариантов условия. Для этого используется блок **else if**, например:

```
if ( условие )
{
  выполняется, если условие истинно
}
else if ( условие )
{
  выполняется, если условие истинно
}
```

```
else if ( условие )
{
  выполняется, если условие истинно
}
else
{
  выполняется, если условие ложно
}
```

Оператор **if else if** может многократно проверять переменную или условия. Последний блок **else** срабатывает по умолчанию, когда не выполнено ни одно условие.

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

else.html

```
let flag = false
const num = 10
```

2 Для выполнения условий логического значения первой переменной вставьте следующие операторы.

```
if( !flag )
{
  console.log( 'Power is OFF' )
}
else
{
  console.log( 'Power is ON' )
}
```

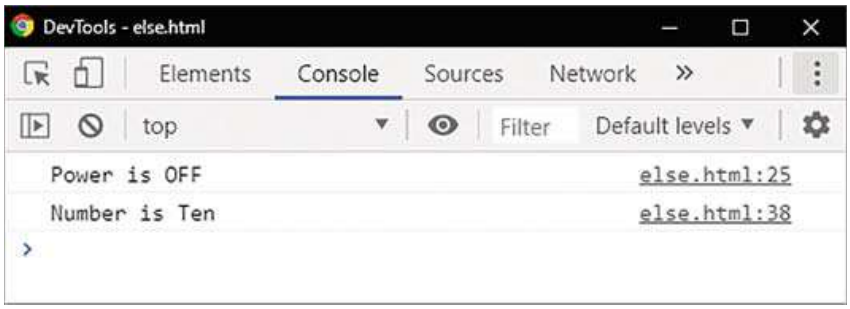
3 Для выполнения условий числового значения второй переменной добавьте следующие операторы.

```
if( num === 5 )
{
  console.log( 'Number is Five' )
}
else if( num === 10 )
{
  console.log( 'Number is Ten' )
}
```

Оператор if может содержать блок else. Он выполняется, когда условие ложно. Блок else if используется при необходимости проверить несколько вариантов условия.

```
else
{
   console.log( 'Number is Neither Five nor Ten' )
}
```

4 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте полученные результаты — условия, которые были выполнены, а которые проигнорированы.



Ветвление с помощью оператора switch

Оператор **if else** подходит для проверки лишь нескольких условий, однако может стать неудобным при столкновении со множеством условий. В такой ситуации наиболее эффективно использовать оператор **switch**.

Оператор **switch** работает необычным образом. Он сравнивает выражение с вариантами, перечисленными внутри нее, а затем выполняет соответствующие действия. Если совпадение найдено, оператор **switch** выполнит один или несколько операторов, связанных с этим значением, в противном случае он выполнит один или несколько операторов, указанных как «операторы по умолчанию».

Оператор **switch** начинается с заключения выражения, которое необходимо проверить, в круглые скобки после ключевого слова **switch**. Затем следуют фигурные скобки **{}**, которые содержат возможные

совпадения. Каждое значение соответствия следует за ключевым словом **case** и использует символ двоеточия: для связывания одного или нескольких выполняемых операторов. Необходимо отметить, что каждый случай должен заканчиваться оператором **break**, который выполняет выход из блока **switch**.

Опциональный оператор **break** может располагаться в каждом из случаев, однако он необязателен. В случае отсутствия оператора **break** выполняются следующие операторы из блока **switch**.

Синтаксис оператора **switch** выглядит таким образом:

```
switch (выражение)
{
  case значение-1 : операторы, которые выполняются
  при нахождении совпадения ; break
  case значение-2 : операторы, которые выполняются
  при нахождении совпадения ; break
  case значение-3 : операторы, которые выполняются
  при нахождении совпадения ; break
  default : операторы, которые выполняются, если совпадения не найдено
}
```

Игнорирование оператора **break** позволяет сценарию выполнять операторы, связанные с последующими несоответствующими значениями блока **case**.

Ограничений на количество блоков **case**, которые могут быть включены в блок операторов **switch**, не существует. Поэтому его использование — это идеальный вариант, чтобы сопоставить любое из десятков, сотен или даже тысяч различных значений.

1. Создайте HTML-документ с самовызывающейся функцией, объявите и проинициализируйте переменную.

```
let day
```

switch.html

2. Затем добавьте оператор **switch**, чтобы присвоить значение переменной после проверки выражения.

```
switch( 5 – 2 )
{
  case 1 : day = 'Monday' ; break
```

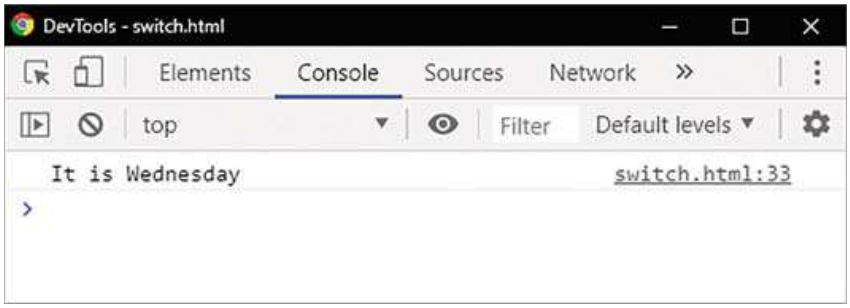
Строковые значения, предлагаемые в качестве возможных совпадений в блоке case, должны быть заключены в кавычки, как и все другие строковые значения.

```
case 2 : day = 'Tuesday' ; break
case 3 : day = 'Wednesday' ; break
case 4 : day = 'Thursday' ; break
case 5 : day = 'Friday' ; break
default : day = 'Weekend'
}
```

3 Добавьте оператор для вывода значения, назначенного оператором **case** при совпадении.

```
console.log( 'It is ' + day )
```

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



Цикл for

Цикл — это структура, содержащая условие и один или несколько операторов, которые повторно выполняются, пока выполняется условие. Повторение процесса с целью получения результата называется итерацией. Если условие не выполняется, дальнейшие итерации также игнорируются, и выполнение продолжается со следующего оператора, идущего за структурой цикла.

Цикл **for** — наиболее распространенная структура цикла в JavaScript. Он имеет такой синтаксис:

for (*инициализатор* ; *условие* ; *модификатор*) { *операторы на выполнение* }

В скобках после ключевого слова **for** указываются три выражения, управляющие числом итераций цикла:

- Инициализатор — инструкция, указывающая начальное значение переменной, которая будет использоваться для подсчета количества итераций цикла. Обычно эту переменную счетчика называют просто «i».

- Условие — выражение, которое при каждой итерации проверяется на логическое значение **true**. Если выражение принимает значение **true**, выполняются операторы цикла для завершения итерации. Если выражение принимает значение **false**, операторы не выполняются, и цикл завершается. Обычно в условии проверяется значение переменной счетчика цикла.

- Модификатор — оператор, который изменяет значение в условии, поэтому в какой-то момент в результате вернется значение **false**. Обычно при этом увеличивается или уменьшается значение переменной счетчика цикла.

Если модификатор не позволяет в какой-то момент вернуть значение **false**, создается бесконечный цикл, т. е. цикл будет выполняться бесконечное число раз.

Например, структура цикла **for** при выполнении набора операторов сто раз может выглядеть следующим образом:

```
let i
for ( i = 0 ; i < 100 ; i++ ) { операторы-на-выполнение }
```

В этом случае при каждой итерации переменная счетчика увеличивается, пока ее значение не достигнет значения **100**, после чего в результате вернется значение **false**, и цикл завершится.

for.html

1. Создайте HTML-документ с самовызывающейся функцией, объявите и проинициализируйте переменную счетчика цикла.

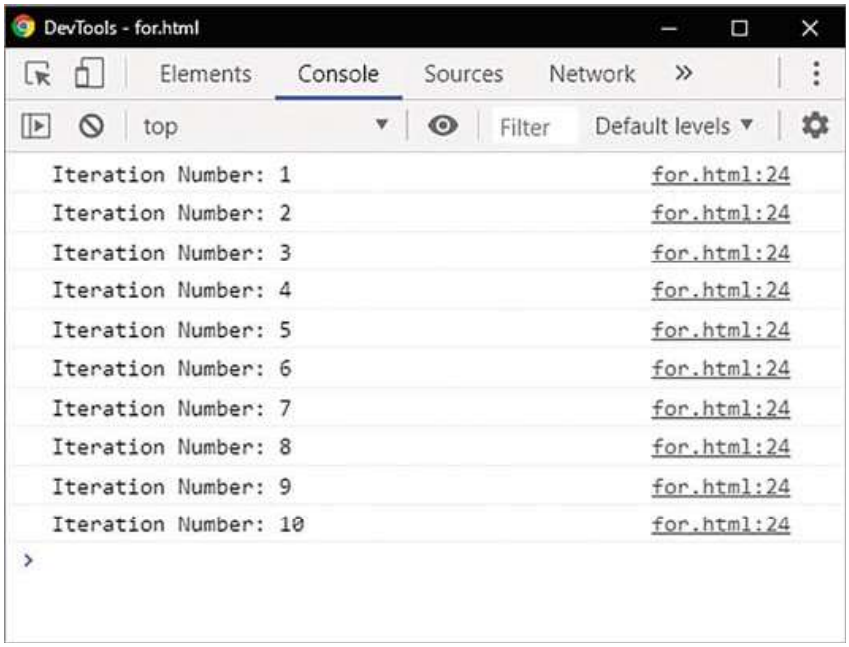
```
let i = 0
```

2. Вставьте структуру цикла **for**, которая будет выполнять 10 итераций и при каждой итерации выводить на экран значение счетчика цикла.

```
for( i = 1 ; i < 11 ; i++ )
```

```
{
  console.log( 'Iteration Number: ' + i )
}
```

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



Мы рассмотрели обычный цикл for. Существует также специальный цикл for in, который используется для перебора свойств объекта. На стр. 76 вы можете найти о нем более подробную информацию.

Цикл while

Структура цикла **for**, описанная на стр. 000, идеальна, когда количество требуемых итераций известно. В ином случае предпочтительнее использовать структуру цикла **while**. Так выглядит его синтаксис:

```
while( условие )
{
  операторы-на-выполнение
  модификатор
}
```

В скобках после ключевого слова **while** указывается условие или выражение, логическое значение которого проверяется каждый раз перед заходом в цикл. Операторы, которые должны выполняться при

каждой итерации, заключаются в фигурные скобки вместе с оператором, который изменяет значение в условии. Поэтому в определенный момент в результате вернется значение **false**, и цикл завершится. Пока результат равен **true**, операторы будут выполняться на каждой итерации цикла.

Если на первой итерации условие принимает значение **false**, цикл немедленно завершается, поэтому операторы, указанные в фигурных скобках, никогда не выполняются. Циклы **while** и циклы **for** иногда называют циклами предварительного тестирования, поскольку их тестовое условие проверяется до выполнения каких-либо операторов.

При отсутствии модификатора в структуре цикла **while** возникнет бесконечный цикл, т. е. цикл будет выполняться бесконечное число раз.

Цикл **while** может быть выполнен для определенного количества итераций, аналогично циклу **for**, используя переменную счетчика в качестве условия проверки и увеличения ее значения на каждой итерации. Например, структура цикла **while** для выполнения набора операторов 100 раз может выглядеть следующим образом:

```
let i = 0
while ( i < 100 )
{
  операторы-на-выполнение
  i++;
}
```

На каждой итерации переменная счетчика увеличивается до тех пор, пока значение не достигнет **100**, после чего результат будет равен **false**, и цикл завершится.

1 Создайте HTML-документ с самовызывающейся функцией, объявите и проинициализируйте переменную счетчика цикла.

while.html

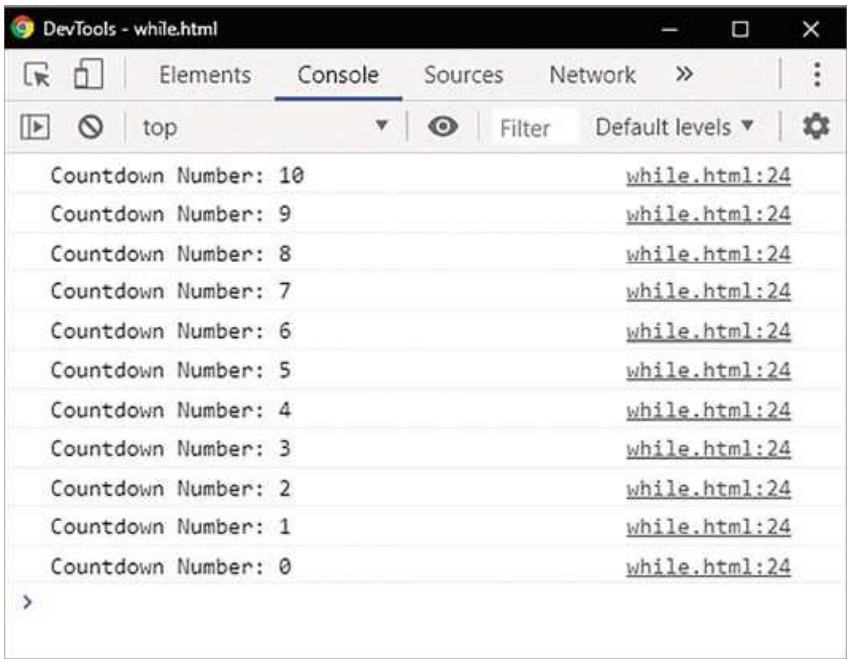
```
let i = 10
```

2 Вставьте структуру цикла **while**, которая будет выполнять итерации и при каждой итерации выводить на экран значение счетчика цикла, пока не достигнет значения 0.

```
while( i > -1 )
{
  console.log( 'Countdown Number: ' + i )
  i--
}
```

Каждый цикл **while** необходимо заключать в фигурные скобки, так как содержится как минимум два оператора — один выполняемый оператор и модификатор.

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



Цикл do..while

Другой пример циклической структуры в JavaScript — это цикл **do..while**. Он похож на инвертированную версию цикла **while**, описанную на стр. 62–63 Цикл **do..while** используется тогда, когда известно, что цикл необходимо запустить хотя бы один раз. Его синтаксис выглядит так:

```
do
{
  операторы-на-выполнение
  модификатор
}
while( условие )
```

В скобках после ключевого слова **while** указывается условие или выражение, логическое значение которого проверяется каждый раз после захода в цикл. Операторы, которые должны выполняться при каждой итерации, заключаются в фигурные скобки вместе с оператором, который изменяет значение в условии. Поэтому в определенный момент в результате вернется значение **false**, и цикл завершится. Пока результат равен **true**, операторы будут выполняться на каждой итерации цикла.

Цикл **do..while** рекомендуется использовать только в том случае, если операторы необходимо выполнить хотя бы один раз.

Если на первой итерации условие принимает значение **false**, цикл немедленно завершается, поэтому операторы, указанные в фигурных скобках, выполняются только один раз. Цикл **do..while** иногда называют циклом с постусловием, так как в отличие от предыдущих циклов он вначале исполняет команды, а затем проверяет заданное условие.

Цикл **do..while** можно принудительно заставить выполнять итерации определенное количество раз, аналогично циклу **for**, используя переменную счетчика в качестве условия проверки и увеличивая ее значение на каждой итерации. Например, структура цикла **do while** для выполнения набора операторов 100 раз может выглядеть следующим образом:

```
let i = 0
do
{
  операторы-на-выполнение
  i++
}
while ( i < 100 )
```

Переменная счетчика увеличивается на каждой итерации до тех пор, пока ее значение не достигнет **100**, после чего в результате вернется значение **false**, и цикл завершится.

1 Создайте HTML-документ с самовызывающейся функцией. Объявите и проинициализируйте переменную счетчика цикла.

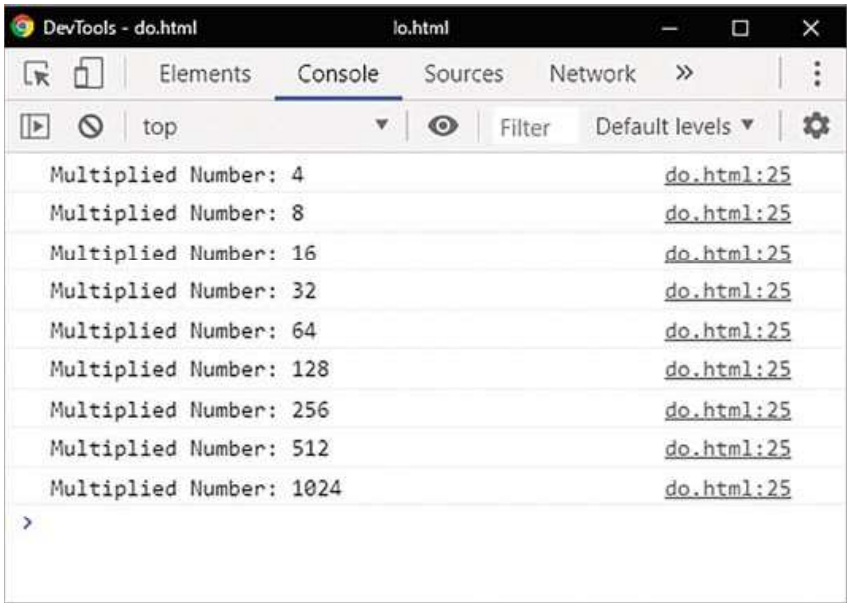
do.html

```
let i = 2
```

2 Вставьте структуру цикла **do..while**, которая будет выполнять итерации и при каждой итерации выводить на экран значение счетчика цикла, пока не превысит значения 1000.

```
do
{
  i *= 2
  console.log( 'Multiplied Number: ' + i )
}
while( i < 1000 )
```

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



Обратите внимание, что окончательное значение превышает предельное значение условия, так как оно записывается на выходе перед выполнением проверки.

Выход из циклов

Оператор **break** прерывает выполнение текущего цикла при обнаружении заданного условия. В циклах он обычно используется для немедленного выхода, когда в зависимости от каких-либо условий требуется завершить выполнение цикла.

Если оператор **break** используется в цикле, вложенном во внешний цикл, блок операторов выполняется во внешнем цикле.

Оператор **continue** прерывает выполнение итерации текущего или отмеченного цикла и продолжает его выполнение на следующей итерации.

Если оператор **continue** используется в цикле, вложенном во внешний цикл, блок операторов выполняется на следующей итерации внутреннего цикла.

break.html

1 Создайте HTML-документ с самовызывающейся функцией. Объявите и проинициализируйте переменную счетчика цикла.

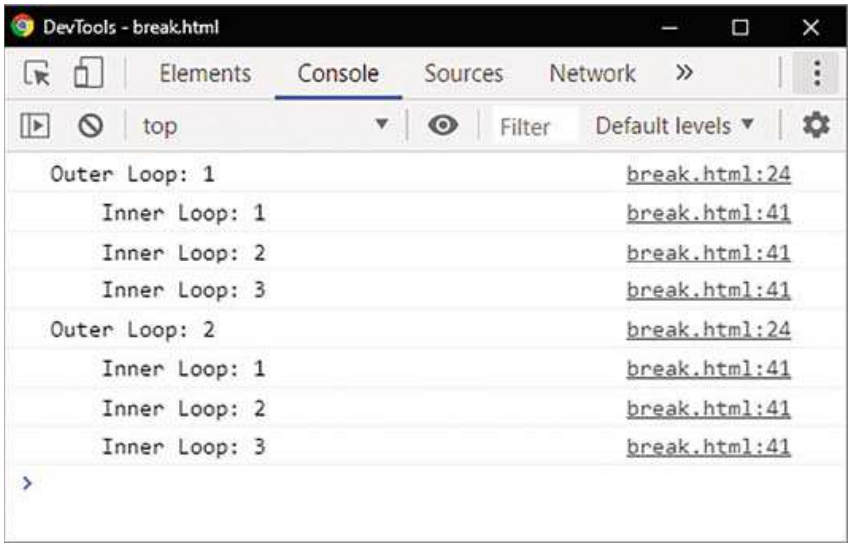
```
let i = 0
let j = 0
```

2 Добавьте цикл **for**, содержащий внутренний вложенный цикл **for**.

```
for ( i = 1 ; i < 3 ; i++ )
{
  console.log( 'Outer Loop: ' + i )
  for ( j = 1 ; j < 4 ; j++ )
  {
    // Здесь будет ваш код.
    console.log( '\tInner Loop: ' + j )
  }
}
```

В данном случае escape-последовательность \t используется для перемещения позиции печати к следующей позиции горизонтальной табуляции.

3 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте полученные результаты — две итерации внешнего цикла и три итерации внутреннего цикла.



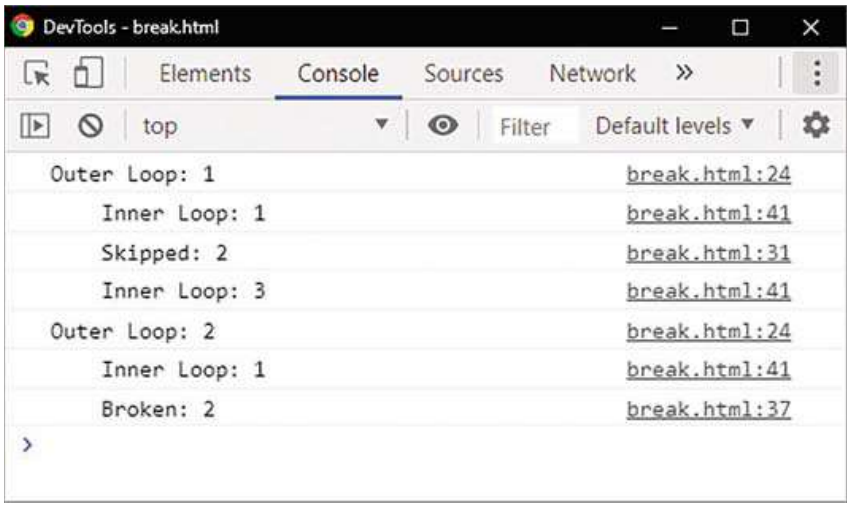
В контексте оператора **switch** оператор **break** обычно используется в конце каждого блока для его завершения.

4 Во внутренний цикл вставьте операторы, чтобы пропустить его итерацию и выйти из внешнего цикла.

```
if( ( i === 1 ) && ( j === 2 ) ) {
   console.log( '\tSkipped: ' + j )
   continue
}

if( ( i === 2 ) && ( j === 2 ) ) {
   console.log( '\tBroken: ' + j )
   break
}
```

5 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте полученные результаты — выход из цикла.



Оператор **break** удобно использовать, когда в определенной ситуации необходимо выйти из цикла.

Обработка ошибок

В JavaScript существует конструкция **try..catch**, позволяющая обрабатывать ошибки (исключения). Операторы, которые должны быть выполнены, внутри блока **try..catch** заключаются в фигурные скобки, а исключения передаются в качестве аргумента в следующий блок **catch**. Конструкция **try..catch** может содержать секцию **finally**, содержащую операторы, которые должны выполняться после обработки ошибок.

JavaScript имеет встроенный объект ошибки, который предоставляет информацию при ее возникновении. JavaScript распознает следующие ошибки: **Error**, **EvalError**, **InternalError**, **RangeError**, **ReferenceError**, **SyntaxError**, **TypeError** и **URIError**. Они могут быть автоматически созданы и переданы в блок **catch** с помощью ключевого слова **new** и конструктора, а затем переданы при использовании ключевого слова **throw**.

Каждый объект ошибки имеет два свойства: **name** и **message**, позволяющие устанавливать или возвращать сообщение об ошибке.

В качестве альтернативы для идентификации ошибки строка может быть передана в блок **catch** с помощью оператора **throw**. Вы можете генерировать ошибки (исключения). Исключения могут быть строкой, числом, логическим значением или объектом JavaScript.

catch.html

1 Создайте HTML-документ с самовызывающейся функцией. Объявите и проинициализируйте переменную.

```
let day = 32
```

2 Для распознавания недопустимых целочисленных значений вставьте блок **try**.

```
try
{
  if( day > 31 )
  {
    throw new RangeError( 'Day Cannot Exceed 31' )
  }

  if( day < 1 )
  {
    throw 'invalid'
  }
}
```

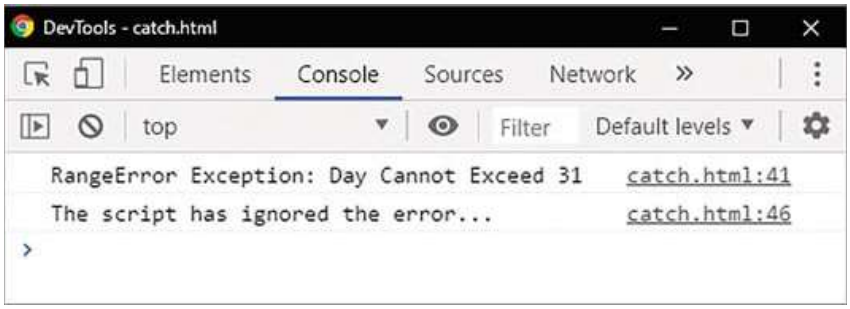
3 Для обработки недопустимых целочисленных значений к блоку **try** добавьте блок **catch**.

```
catch( err )
{
   if( err === 'invalid' )
   {
      console.log( 'Variable has invalid value of ' + day )
   }
   else
   {
      console.log( err.name + ' Exception: ' +
      err.message )
   }
}
```

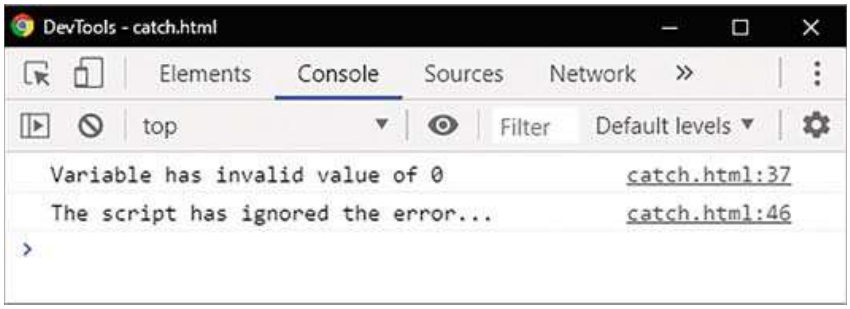
4 Для вывода окончательного сообщения к блоку **catch** добавьте блок **finally**.

```
finally
{
   console.log( 'The script has ignored the error...' )
}
```

5 Сохраните HTML-документ, затем откройте его в браузере и запустите консоль. Проанализируйте результаты — полученное сообщение об ошибке.



6 Чтобы увидеть обнаруженную ошибку, измените значение переменной на 0, затем сохраните HTML-документ и обновите страницу в браузере.



Чтобы обнаружить автоматическую ошибку ReferenceError, удалите или закомментируйте объявление переменной day, затем сохраните и обновите этот пример.

Заключение

- Оператор **if** проверяет условие на наличие логического значения **true** или **false**.
- Альтернативная ветвь, по которой действия могут быть выполнены при ложном результате, обозначается ключевым словом **else**.
- С помощью оператора **switch** можно сравнить выражение сразу с несколькими условиями.
- В блоке **switch** каждый блок **case** должен заканчиваться оператором **break**, чтобы прервать выполнение.
- Блок **switch** может содержать оператор **default**. Оператор **default** выполняется, если ни один вариант не совпал.
- В цикле **for** указывается инициализатор, условие, которое должно быть проверено на логическое значение **true** или **false**, и модификатор.
- Модификатор — это оператор, который изменяет значение в условии, поэтому в какой-то момент в результате вернется значение **false**, и происходит выход из цикла.
- Циклы **while** и **for** проверяют условие перед выполнением блока операторов.
- Цикл **do..while** проверяет условие после того, как блок операторов был выполнен.
- Оператор **break** прерывает выполнение текущего цикла.
- Оператор **continue** прерывает выполнение текущей итерации в цикле и продолжает на следующей итерации.
- Структура **try..catch** используется для обработки возникающих в сценарии ошибок (исключений).

- Каждый объект ошибки имеет два свойства: **name** и **message**, позволяющие устанавливать или возвращать сообщение об ошибке.

- В качестве альтернативы для идентификации ошибки строка может быть передана в блок **catch** с помощью оператора **throw**.

- Конструкция **try..catch** может содержать секцию **finally** с операторами, которые должны выполняться после обработки ошибок.