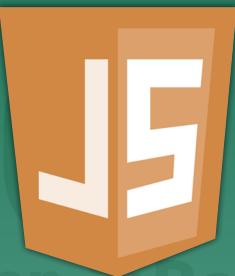


# Разработка клиентских сценариев с использованием JavaScript и библиотеки jQuery



# Unit 3

## Циклы

## Contents

Циклы .....	3
Что такое цикл? .....	7
While .....	10
Do while .....	15
For .....	17
For in, for of .....	25
Break, continue .....	26
Понятие метки .....	29
Задание для самостоятельной работы .....	31

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

# Циклы

Отвлечемся от программирования и представим, что нам надо приготовить обед, для чего необходимо начистить некоторое количество картошки. Формально, для того чтобы это сделать нужно повторить однообразное действие «почистить картошку» для нескольких разных (или одинаковых) картофелин.

Разберем процесс чистки более детально. Во-первых, необходимо взять картофелину, осмотреть, удалить «глазки» при их наличии. Во-вторых, очистить ее от кожуры. В-третьих, промыть и сложить в емкость для чищенной картошки. Затем эти действия должны быть повторены в той же последовательности для других картофелин. Обобщая, получаем циклическое повторение набора действий для исходного набора картошки.

Следующая особенность циклического процесса «почистить картошку» касается вопроса, когда же следует остановиться, прекратить процесс, ведь действия, которые повторяются, ничем не ограничены. Ответ на этот вопрос вытекает из деталей того задания, которое перед нами было поставлено изначально. А это задание могло быть поставлено по-разному: 1) «начистить 5 картофелин», 2) «начистить всё, что лежит в пакете» или 3) «начистить сколько получится за 10 минут».

В первом случае нам заранее известно количество картофелин, то есть мы знаем сколько раз нужно повторить действие, нужно просто подсчитывать количество почищенных картофелин (количество повторенных дей-

ствий). Во втором случае количество заранее неизвестно, но получив пакет, мы можем пересчитать его содержимое и действовать аналогично первому случаю, либо руководствоваться другим условием — повторять действия, пока не опустеет пакет. Третий вариант практически ничего не сообщает нам о количестве повторов, оно будет зависеть от множества факторов, как то размеры картофелин, наша скорость и опытность по их чистке и т.п. Однако условие остановки нам все же известно, только зависит оно от другого параметра — прошедшего времени.

Задачи подобного характера довольно часто возникают и при построении сценариев, когда требуется повторное выполнение одних и тех же команд для разных (а иногда и одинаковых) данных. Представим, что перед программистом поставили задачу разработать кредитный калькулятор, и рассмотрим, к каким повторным действиям нам придется прибегать. В простейшем случае предполагается, что кредит берется на год и погашается равномерными платежами каждый месяц. Например, 1000 у.е. было взято под 20% годовых, что предусматривает возврат суммы в 1200 у.е. Значит, в каждый из 12 месяцев года платеж по кредиту будет составлять 100 у.е. Программа должна вывести график погашения кредита:

Месяц 1 — платеж 100 у.е.

Месяц 2 — платеж 100 у.е.

...

Месяц 12 — платеж 100 у.е.

Данный пример можно соотнести с чисткой известного количества картошки, т.к. нам заранее известно сколько раз нужно вывести надпись на экран. С одной

стороны, это позволяет написать 12 одинаковых инструкций по выводу данных, только в каждой из них указать свой номер месяца. С другой стороны, такой подход кажется неоптимальным, особенно, если его обобщать и на большее количество повторений.

Вторая функция калькулятора будет допускать, что срок кредита пользователь вводит самостоятельно. Программа все так же должна вывести график погашения кредита, но уже не на 12 месяцев, а на указанный срок.

В такой формулировке задача напоминает «начистить пакет картошки». До того, как мы начали выполнять задачу количество повторных действий неизвестно. Обратите внимание: задача сформулирована, а количество не определено. Однако когда мы приступим к выполнению задания (и получим пакет), количество будет установлено. Также и в калькуляторе — расчет начинается после пользовательского ввода, а значит, к началу счета срок будет уже определен.

Тем не менее, программу мы пишем до того, как пользователь введет цифры, и она должна быть готова обработать любые введенные данные. Идея написать несколько инструкций вывода подряд приобретает еще больше недостатков, так как мы не можем знать, сколько инструкций использовать в программе. Можно, конечно, написать инструкции «с запасом» и каждую из них поместить в условный оператор

```
...
if(term >= 12) console.log("Month 12 – sum 100");
if(term >= 13) console.log("Month 13 – sum 100");
if(term >= 14) console.log("Month 14 – sum 100");
...
```

Однако, даже при наличии хорошего «запаса» все равно такая программа будет иметь предел, после которого расчет будет невозможен. С одной стороны, никто не будет давать кредит на 100 лет, то есть рассчитать «запас» все же можно. С другой стороны, такое решение выглядит совсем непривлекательно.

Дополним наш кредитный калькулятор еще одной функцией — пользователь вводит сумму, которую он готов выплачивать ежемесячно, а программа должна рассчитать срок кредитования исходя из этой суммы.

В таком случае срок кредитования нам неизвестен даже после ввода данных пользователем. Программа должна моделировать процесс: каждый месяц задолженность возрастает на кредитную ставку и уменьшается на ежемесячный платеж. Тот месяц, в котором задолженность будет погашена, станет последним и позволит рассчитать срок кредитования. По степени определенности данных имеем аналогию с чисткой картошки в течение 10 минут — количество будет известно только по прошествии заданного времени и не может быть определено до его окончания.

При таком алгоритме действий составить программу в виде последовательных инструкций будет совсем не-приемлемо (хотя и принципиально возможно). Каждая следующая команда должна сопровождаться рядом проверок, по результатам которых следует либо продолжать моделирование, либо остановиться и вывести итоги. И таких команд нужно предусмотреть на все допустимые сроки кредитования.

При наличии принципиальных отличий рассмотренных вариантов задачи между собой, во всех них можно

выделить общую черту — необходимость повторять одинаковые действия (один и тот же блок команд). Разница будет заключаться лишь в том, когда повторение следует прекратить: после 12 месяцев, после введенного срока или после обнуления задолженности.

## Что такое цикл?

Программные механизмы, позволяющие несколько раз выполнить один и тот же блок кода, называют циклами. При работе цикла программа снова и снова «возвращается» на начало блока команд, выполняя его заново, инструкция за инструкцией, до того, как цикл будет прекращен (остановлен).

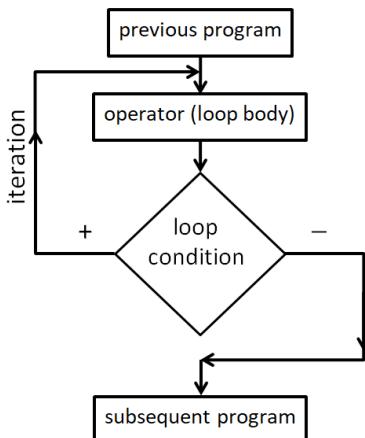


Рисунок 1

Повторяемый блок кода носит название «тело цикла» (*loop body*), а процесс его выполнения — итерация (*iteration*). Фраза «на второй итерации» означает «при втором повторе тела цикла». Обычно, повтор (*итерация*) цикла

происходит в зависимости от некоторого выражения, называемого цикловым условием (*loop condition*). Если это условие истинно, цикл будет повторяться, если ложно — прекратится. Схематически, работу цикла иллюстрирует рисунок 1.

Если циклового условия нет, повторения цикла не прекращаются, и получается «бесконечный цикл». Такой цикл выполняется в течение всей работы сценария и прерывается только после его завершения. Бесконечные циклы могут применяться для постоянного мониторинга каких-либо параметров (например, сетевой активности), поддержания анимации или пользовательского ввода в играх. С негативной стороны, бесконечный цикл может получиться в результате ошибок программиста. В таком случае возможно «зависание» сценария, потеря его управляемости. Скорее всего, при этом придется закрыть вкладку браузера, исправить ошибки и заново открыть документ.

Если в цикле необходим подсчет номера итерации, то для этого вводится отдельная переменная, меняющая свое значение при каждом повторе цикла. Эта переменная-счетчик носит название «цикловая переменная» (англ. *cycle index* или *counter*). Обычно, она принимает значение равное номеру повтора цикла 0-1-2-3... или 1-2-3-4..., хотя бывают задачи, когда удобнее использовать другие множества значений, например, только четные числа. В таком случае, величина, на которую изменяется цикловая переменная, называют «шагом» (*step*).

По способу организации повторного выполнения кода, циклы делят на циклы-счетчики и условные ци-

клы. Последние дополнительно подразделяются на циклы с предусловием и циклы с постусловием.

Циклы-счетчики (или циклы со счетчиком) выполняются заданное количество раз. Количество повторов может задаваться как числом, так и значением переменной, но это значение должно быть известно до начала цикла. Задача по расчету кредитных платежей на конкретный срок является типичным примером использования циклов данного вида. Даже если изначально срок кредита неизвестен, в момент расчета платежей это число уже будет введено пользователем, и количество повторов цикла будет известным.

Условные циклы (или циклы с условием) выполняются до тех пор, пока некоторое условие является истинным. В некоторых языках программирования (JavaScript к ним не относится) есть разновидности циклов, выполняющиеся пока условие ложно. При помощи таких циклов, например, может быть решена задача по расчету кредита исходя из размера ежемесячного платежа. Не зная перед количеством повторов цикла, нам всё же известно условие, по которому цикл должен быть прерван.

Если цикловое условие проверяется до перехода к телу цикла, то такие циклы называют «циклы с предусловием», если после выполнения тела — «циклы с постусловием». Цикл с предусловием может не выполниться ни разу, если условие изначально ложно. Цикл с постусловием будет выполнен, по крайней мере, один раз до первой проверки условия. В остальном, эти циклы подобны и часто могут быть заменены один на другой. В некоторых языках программирования (например, в Python) есть только один из этих циклов.

**Задание.** Программа «угадай число» загадывает некоторое число, а пользователь пытается его угадать, вводя свои предположения с клавиатуры. В случае если предложенный вариант не совпал с загаданным, пользователю предоставляется следующая попытка, и так до тех пор, пока число не будет угадано. Определите, какой тип цикла будет предпочтительным для программы «угадай число». (Пользователь должен ввести свой вариант числа хотя бы один раз — нельзя угадать число, не введя предположения. Поэтому предпочтительным будет цикл с постусловием.)

## While

Цикл с предусловием создается при помощи оператора «**while**». Общий синтаксис оператора довольно прост:

```
while(condition)
    statement;
```

За ключевым словом «**while**» в круглых скобках указывается условное выражение, после скобок — инструкция для выполнения (тело цикла). Цикл обеспечивает повторное выполнение тела до тех пор, пока условие остается истинным. Как обычно, в случае необходимости включения в тело цикла нескольких языковых инструкций, применяется группирующий оператор «{}».

```
while(condition) {
    statement1;
    statement2;
    statement3;
}
```

Для того чтобы цикл имел окончание, в теле цикла должны быть предусмотрены инструкции, оказывающие некоторое влияние на условие повторения цикла. В противном случае, цикл либо не выполнится ни разу (при ложном условии), либо будет повторяться бесконечно (при истинном). Следует обратить внимание на то, что это довольно популярная ошибка даже у опытных программистов — забыть включить в тело цикла инструкции по управлению цикловым условием.

Далее приведены два примера, выводящие в консоль числа от 1 до 9, один правильный, другой — содержащий ошибку. Причем, ошибка эта не синтаксическая, фрагмент может быть выполнен. В теле цикла отсутствует инструкция «`i++`», влияющая на цикловое условие «`i < 10`», из-за чего цикл становится бесконечным — переменная «`i`» всегда равна 1, то есть всегда меньше 10.

Правильно	Неправильно
<pre>i = 1; while(i &lt; 10){     console.log(i);     i++; }</pre>	<pre>i = 1; while(i &lt; 10){     console.log(i); }</pre>

Цикл «`while`» удобно применять в случаях, когда

- заранее неизвестно количество повторов тела цикла,
- условие окончания цикла прямо не зависит от цикловой переменной,
- цикл зависит от входящих данных, например, от действия пользователя.

В качестве примера рассмотрим реализацию кредитного калькулятора, требующую условного цикла. Пусть

величина ежемесячного платежа хранится в переменной «`monthlyPayment`», сумма кредита — в переменной «`creditAmount`», кредитная ставка — в переменной «`creditRate`». Дополнительно в переменной «`month`» будем хранить количество моделированных месяцев. Тогда программный фрагмент будет выглядеть следующим образом:

```
month = 0;
while(creditAmount > 0){
    creditAmount *= creditRate;
    creditAmount -= monthlyPayment;
    month++;
}
```

Изначально устанавливаем счетчик месяцев в ноль. Запускаем цикл с условием (`creditAmount > 0`), то есть пока сумма кредита не погашена. В теле цикла увеличиваем остаток кредита на величину ставки (`creditAmount *= creditRate`) и тут же уменьшаем ее на ежемесячный платеж (`creditAmount -= monthlyPayment`). Затем увеличиваем на 1 счетчик месяцев.

По окончанию цикла в переменной «`month`» будет храниться срок кредита (количество месяцев) до его окончательного погашения.

**Рассмотрим еще одну задачу:** мы хотим накопить 1 миллион у.е., получая проценты по банковскому вкладу (депозиту). Каждый год на наш вклад начисляется 10% и эти начисления добавляются к основному вкладу. То есть при вкладе 1000 у.е. в конце первого года будет начислено 100 у.е. и они добавятся к вкладу, увеличив его до 1100 у.е. В конце второго года 10% уже будет начислено

на текущую сумму и составит 110 у.е., увеличив в итоге вклад до 1210 у.е. Нас интересует, за сколько лет наш вклад достигнет 1 миллиона у.е.

Задача подобна предыдущей, только сумма вклада не уменьшается, а увеличивается, и условие выхода заключается не в обнулении суммы, а достижения заданного предела (1 млн.). Соответственно, для данной задачи также предпочтительно использовать условный цикл.

Оформим решение задачи в виде отдельного самостоятельного документа. Создайте новый файл, наберите или скопируйте в него следующее содержание (*код также доступен в папке Sources — файл js1\_6.html*)

```
<!doctype html>
<html>
    <head>
    </head>
    <body>
        <script>
            var fund = +prompt("Initial deposit:");
            var years = 0;
            while(fund < 1e6) {
                years++;
                fund += fund * 0.1; // 0.1 = 10%
            }
            alert("You'll become millionaire after" +
                  years + " years");
        </script>
    </body>
</html>
```

Для ввода данных воспользуемся диалоговым окном «***prompt***». Поскольку начальная сумма вклада должна быть величиной числового типа, применим оператор «***+***»

перед результатом ввода и сохраним его в переменной «`fund`». Введем переменную-счетчик «`years`» инициализировав ее значением `0`. Далее циклично увеличиваем «`years`» на `1` (`years++`), а вклад — на `10%` (`fund += fund * 0.1`). Условием повторения цикла будет выражение, сравнивающее накопленную сумму с миллионом. Обратите внимание, для удобства подсчета нулей, а также для сокращения записи «`1000000`» использована экспоненциальная форма представления числа миллион — «`1e6`».

По окончанию цикла выводим окно-сообщение «`alert`» с данными о количестве лет, необходимых для выполнения условия задачи.

Сохраните файл, откройте его при помощи браузера. В появившемся диалоговом окне введите сумму первого вклада, нажмите «`OK`». Результат работы скрипта должен выглядеть как сообщение о протяженности пути к миллиону.

**Найдите ошибку:** программа должна запросить у пользователя три числа и вывести их в консоль браузера

```
<script>
    var limit = 3;
    var numbers = 0, number;
    while(numbers < limit) {
        number = +prompt("Type number:");
        console.log(number);
    }
</script>
```

(В цикле пропущены инструкции, меняющие цикловое условие, и цикл становится бесконечным. В теле нужно дописать «`numbers++`».)

## Do while

Вторая разновидность условных циклов, — цикл с постусловием, — создается при помощи оператора «**do while**» по следующей схеме:

```
do
    statement
while(condition)
```

Тело цикла (*выражение statement*) выполняется повторно до тех пор, пока условие (*condition*) остается истинным. Несмотря на то, что операторные рамки «**do**» и «**while**» позволяют однозначно определить начало и конец тела цикла, при использовании нескольких инструкций в цикле их все равно нужно объединять группирующим оператором «**{}**».

```
do {
    statement1;
    statement2;
    statement3;
} while(condition)
```

В отличие от цикла «**while**», цикл с постусловием при ложном условии выполнится один раз, поскольку условие повтора проверяется после выполнения тела. По аналогии с предыдущим циклом следует помнить о необходимости влияния на условие повторения во избежание «зацикливания» — бесконечной работы цикла из-за неизменности циклового условия. Циклы с постусловием удобно применять как раз в тех случаях, когда тело цикла должно быть выполнено, по крайней мере, один раз.

**Например**, мы хотим получить два случайных числа, но не равных между собой. То есть при совпадении двух чисел нам нужно повторить запрос случайного числа (сопадение часто бывает, когда количество чисел небольшое, например, при играх с кубиком). Если после повтора снова обнаруживается равенство, нужно обеспечить еще один повтор, и так далее.

Первое число получаем обычным образом «`x1=Math.random()`». Второе число надо рассчитать как минимум один раз, а при совпадении с первым — рассчитывать повторно. Для этого как нельзя лучше подходит цикл с постусловием: «`do x2= Math.random(); while(x2==x1)`». Если второе число отличается от первого, то никаких повторов не будет. Если же совпадет — сработает условие цикла и расчет будет повторен.

```
x1=Math.random();
do
    x2= Math.random();
while(x2==x1)
```

**Другой пример:** у пользователя нужно получить подтверждение некоторого действия. Мы не хотим использовать диалог «`confirm`», т.к. в нем можно случайно нажать пробел, «`Enter`» или «`Esc`». Нам нужно уверенное подтверждение, то есть пользователь должен ввести либо «`yes`», либо «`no`». В ином случае мы будем выводить запрос повторно, ожидая одного из двух ответов. Так как запрос нужно выводить как минимум один раз, цикл с постусловием будет предпочтительным:

```
<script>
    var txt;
    do
        txt = prompt("Confirm: yes or no")
    while(txt!="yes" && txt!="no")
</script>
```

В остальном же особенности, замечания и предостережения по применению условных циклов одинаковы, независимо от месторасположения циклового условия.

## For

Цикл-счетчик (или цикл со счетчиком) организуется при помощи оператора «**for**». Синтаксис оператора, кроме тела цикла, содержит три декларативных блока:

```
for (initialization; condition; expression)
    statement
```

Первый блок «**initialization**» используется для начальной инициализации цикловой переменной. Этот блок выполняется один раз перед началом цикла. Второй блок «**condition**» содержит условие, при котором цикл продолжается. Обычно это условие содержит ограничение на цикловую переменную. Третий блок «**expression**» задает выражение для изменения цикловой переменной. Второй и третий блоки выполняются на каждой итерации цикла.

Телом цикла является одна языковая инструкция «**statement**», которой может быть группирующий оператор «**{}**», содержащий несколько собственных инструкций.

Типичным примером записи оператора «`for`» является следующий:

```
for(i=0; i<5; i++)
    console.log(i);
```

В первом блоке инициализируется цикловая переменная «`i=0`», во втором указывается условие на эту переменную «`i<5`», в третьем — алгоритм ее изменения «`i++`» (увеличение на 1). В данном примере телом цикла является команда вывода значения цикловой переменной в консоль браузера. Традиционно цикловую переменную называют «`i`» (от англ. *iteration — итерация*), но это не является обязательным и для нее может быть выбрано произвольное имя.

Наберите или скопируйте приведенный выше фрагмент с описанием цикла в консоль и нажмите «`Enter`». Как результат работы цикла в консоли должны появиться цифры от нуля до четырех.



Рисунок 2

Сам оператор «`for`» возвращает значение последней операции в теле цикла, поэтому в консоли появляется ответ «`undefined`», возвращенный командой «`console.log(4)`».

Работа цикла начинается с выполнения первого блока оператора `for «i=0»` и проверки условия во втором блоке `«i<5»`. Если условие истинно, выполняется тело цикла. После этого активируется третий блок `«i++»`, увеличивающий цикловую переменную, и снова проверяется условие `«i<5»`.

Согласно с описанным механизмом работы, цикл `«for»` является разновидностью циклов с предусловием (условие проверяется до выполнения тела). Чтобы в этом убедиться заменим «верхний предел» изменения переменной на ноль, т.е. введем изначально ложное условие.

Ведите в консоль тот же цикл, но с условием `«i<0»` (повтор команды в консоли обеспечивается нажатием «стрелки вверх» на клавиатуре). Нажмите `«Enter»` и обратите внимание на отсутствие результатов работы цикла (только ответ `«undefined»` от самого оператора `«for»`). Если бы условие проверялось после выполнения тела, мы бы наблюдали, по крайней мере, один результат его срабатывания, и в консоль был бы выведен ноль.

Содержание всех трех блоков оператора `«for»` не ограничивается описанными выше правилами и может быть практически произвольным. Прежде всего, эти блоки вообще не являются обязательными и могут быть оставлены пустыми (разделители блоков `«;»` при этом должны оставаться). Например, если цикловая переменная инициализирована до цикла, первый блок можно не указывать (все дальнейшие примеры можно скопировать в консоль и проверить на работоспособность)

```
i=0;  
for( ; i<5; i++)  
    console.log(i);
```

Аналогично, если цикловая переменная меняет значение в теле цикла, то последний блок также можно не указывать

```
i=0;
for( ; i<5 ; ) {
    console.log(i);
    i++;
}
```

Обратите внимание, что в результате выполнения последнего примера в консоли появится две цифры «4». Вторая цифра соответствует значению, которое возвращает сам оператор «**for**» — результат последней команды в теле цикла (**i++**). При этом надпись «**undefined**» из консоли исчезает.

Если в теле цикла содержатся механизмы собственной остановки, то и второй блок, отвечающий за цикловое условие, становится ненужным (подробнее об операторе «**break**» будет рассказано в следующем разделе)

```
i=0;
for( ; ; ) {
    console.log(i);
    i++;
    if(i>4)
        break;
}
```

С учетом того, что проверка (**i>4**) перенесена в конец тела, данная реализация цикла приобретает форму цикла с постусловием.

Все блоки оператора «`for`» могут содержать по нескольку операций, разделенных запятой. Например, следующий цикл содержит две переменные «`i`» и «`j`», одна увеличивается, начинаясь с `0`; другая уменьшается, стартуя с `10`. Цикл продолжается, пока переменные не равны между собой (`i!=j`). Другими словами, цикл прекращается, когда значения переменных оказываются равными друг другу.

```
for(i=0, j=10; i!=j; i++, j--)
    console.log(i, j)
```

Тело цикла содержит вывод в консоль обеих переменных. Результат выполнения такого цикла будет подобен следующему (см. рис. 3).

The screenshot shows the 'Console' tab of a browser's developer tools. It displays the following code and its execution results:

```
> for(i=0, j=10; i!=j; i++, j--) console.log(i, j)
0 10
1 9
2 8
3 7
4 6
< undefined
>
```

Рисунок 3

Описанные манипуляции с блоками дают обширные возможности управления циклом «`for`». В частности, им можно заменить все рассмотренные ранее условные циклы. Однако при этом следует не забывать о читаемости кода и правилах «хорошего тона». Все-таки оператор «`for`» изначально предназначался для циклов-счетчиков и именно для этих целей его желательно использовать. В свою

очередь, условные циклы следует создавать специально для того введенными операторами «**while**» или «**do-while**».

**В качестве примера рассмотрим** реализацию оставшихся функций кредитного калькулятора. В простейшем случае мы рассчитываем ежемесячные платежи кредита, оформленного на год. Будем считать, что годовая ставка кредита составляет **20%** и она сразу начисляется на всю сумму, не меняясь в зависимости от ежемесячных платежей.

```
fund = +prompt("Credit sum");
creditBody = fund + 0.2*fund;
monthlyPayment = creditBody / 12;
for(i=1; i<=12; i++)
    console.log("month " + i +
                " payment "+monthlyPayment);
```

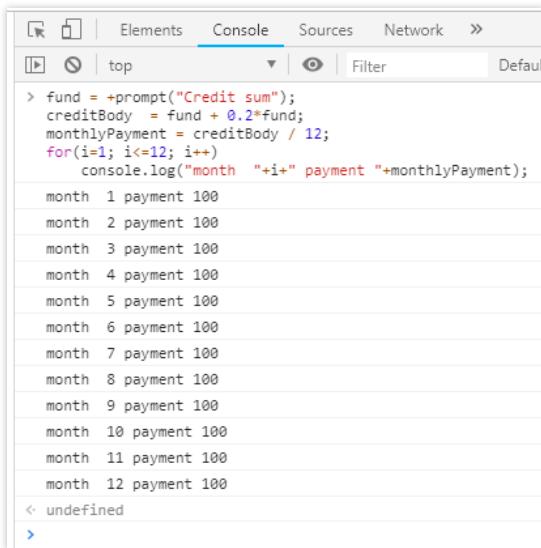
В первой строке кода мы запрашиваем у пользователя желаемую сумму кредита и сохраняем ее в переменной «**fund**». Используем преобразование введенных данных к числовому типу при помощи оператора «**+**», указанного перед вызовом диалогового окна «**prompt**».

Затем рассчитываем тело кредита «**creditBody**», добавляя к введенной сумме **20%**, что соответствует коэффициенту **0.2**, умноженному на введенную сумму. Затем вычисляем ежемесячный платеж «**monthlyPayment**» путем деления тела кредита на 12 месяцев.

Далее организовываем цикл. Поскольку срок кредитования нам задан условием задачи, используем цикл-счетчик «**for**», классически, цикловой переменной даем имя «**i**». Начальное значение устанавливаем равным 1 (первый месяц). Предельное значение — 12. С учетом того, что

само число 12 также должно быть обработано, используем нестрогое сравнение «`i<=12`». В теле цикла выводим в консоль номер месяца и сумму ежемесячного платежа.

Наберите или скопируйте в консоль браузера приведенный программный фрагмент, нажмите «`Enter`». Откроется диалоговое окно «`prompt`» в котором введите сумму, например, 1000. После ввода в консоли появится график платежей, подобный приведенному на рисунке 4.



The screenshot shows a browser's developer tools console tab selected. The code entered is:

```
> fund = +prompt("Credit sum");
creditBody = fund + 0.2*fund;
monthlyPayment = creditBody / 12;
for(i=1; i<=12; i++)
    console.log("month "+i+" payment "+monthlyPayment);
month 1 payment 100
month 2 payment 100
month 3 payment 100
month 4 payment 100
month 5 payment 100
month 6 payment 100
month 7 payment 100
month 8 payment 100
month 9 payment 100
month 10 payment 100
month 11 payment 100
month 12 payment 100
< undefined
```

Рисунок 4

Как мы уже знаем, оператор цикла «`for`» возвращает результат последней операции, поэтому в конце вывода появляется «`undefined`».

Второй вариант калькулятора будет запрашивать еще и срок кредитования. Будем считать, что срок должен вводиться в месяцах. Тогда в программе появится дополнительная переменная «`months`», хранящая в себе

указанный срок. Аналогично сумме кредита она будет запрошена у пользователя диалоговым окном «`prompt`».

```
fund = +prompt("Credit sum");
months = +prompt("Credit term (months)");
creditBody = fund + 0.2 * fund * months / 12;
monthlyPayment = creditBody / months;
for(i=1; i<= months; i++)
    console.log("month " + i + " payment " + monthlyPayment);
```

В расчетной части программы константа «`12`» заменяется на введенный пользователем срок, а также пересчитывается процентная ставка: добавляется множитель «`month / 12`», определяющий количество лет кредита. Операции вывода остаются такими же.

Наберите или скопируйте в консоль браузера приведенный программный фрагмент, нажмите «`Enter`». Откроется диалоговое окно «`prompt`» в котором введите сумму, например, `1000`. Появится второе диалоговое окно с запросом срока, введите `6`. После ввода в консоли появится график платежей с учетом введенных данных (рис. 5).

The screenshot shows a browser's developer tools console tab labeled 'Console'. The code entered is:

```
> fund = +prompt("Credit sum");
> months = +prompt("Credit term (months)");
> creditBody = fund + 0.2 * fund * months / 12;
> monthlyPayment = creditBody / months;
> for(i=1; i<= months; i++)
>     console.log("month " + i + " payment " + monthlyPayment);
>
```

The output in the console is:

```
month 1 payment 183.333333333334
month 2 payment 183.333333333334
month 3 payment 183.333333333334
month 4 payment 183.333333333334
month 5 payment 183.333333333334
month 6 payment 183.333333333334
< undefined
>
```

Рисунок 5

**Задание:** используйте функцию округления для того, чтобы полученные числа отображались с двумя цифрами после запятой — в «денежном формате».

## For in, for of

Наиболее популярное применение циклы-счетчики находят при работе с массивами или коллекциями для перебора их содержимого. Детальнее о таких программных конструкциях будет рассказано в следующем уроке, но для полноты рассмотрения оператора «**for**» приведем здесь его разновидности, адаптированные для указанных задач.

Циклы «**for in**» и «**for of**» позволяют перебирать свойства комплексных объектов (объединяющих в себе несколько других объектов), используя сокращенную запись оператора «**for**».

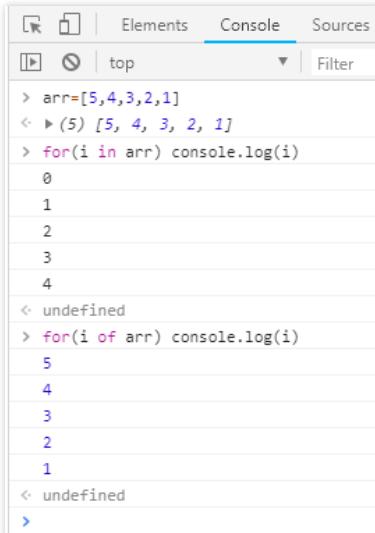
```
for (variable in object) statement  
      for (variable of object) statement
```

Здесь «**variable**» — цикловая переменная (точнее, ее имя), «**object**» — массив, коллекция или объект, содержащие некий набор внутренних элементов.

Проще всего пояснить работу этих циклов и различия между ними на следующем примере. Создадим массив из нескольких значений. Введем в консоль браузера команду «**arr=[5, 4, 3, 2, 1]**» и нажмем «**Enter**». Теперь организуем циклы, перебирающие эти массивы. Вводим последовательно в консоль: «**for(i in arr) console.log(i)**», нажимаем «**Enter**», затем «**for(i of arr) console.log(i)**» и снова нажимаем «**Enter**».

Результатом работы первого цикла являются числа **0–4**, отвечающие за индексы (номера) элементов массива,

тогда как второй цикл выводит само содержимое массива: числа **5–1**. И те, и другие данные могут потребоваться для различных задач, из-за чего и предусмотрены две формы оператора цикла.



```

Elements    Console    Sources
top
> arr=[5,4,3,2,1]
< ▶ (5) [5, 4, 3, 2, 1]
> for(i in arr) console.log(i)
0
1
2
3
4
< undefined
> for(i of arr) console.log(i)
5
4
3
2
1
< undefined
>

```

*Рисунок 6*

Сокращенные формы записи являются лучше читаемыми и экономят несколько символов программного кода при оптимизации программ. К особенностям их работы вернемся, когда более детально изучим массивы, коллекции и объекты.

## Break, continue

Для того чтобы иметь возможность дополнительного управления процессом выполнения цикла предусмотрены операторы «**break**» и «**continue**». Эти операторы могут применяться во всех рассмотренных выше циклах.

Оператор «`break`» полностью останавливает выполнение цикла, независимо от состояния циклового условия. Такое действие может понадобиться, если в процессе выполнения цикла появится некоторое недопустимое значение, например, в ожидаемых числовых данных появится не-цифра, или пользователь захочет остановить программу «вручную». Также прерывание может потребоваться при возникновении условий, не позволяющих продолжать цикл, например, пропадет подключение к сети или отключится устройство, с которого читаются данные.

Оператор «`continue`» останавливает выполнение данной итерации цикла. Если цикловое условие позволяет, то будет запущена следующая итерация. Такое поведение может быть полезно при игнорировании каких-либо данных, например, пробелов и дефисов при анализе номера телефона.

В некоторых случаях операторы «`break`» и «`continue`» применяются не для реакции на непредвиденные ситуации, а для нормальной организации логики работы циклов. Рассмотрим несколько примеров.

**Задача:** нужно генерировать не больше десяти случайных чисел из диапазона `1–7`. В случае если выпадает четверка, генерация прекращается.

Во-первых, адаптируем генератор случайных чисел JavaScript для работы в указанном диапазоне. Существующий генератор «`Math.random()`» выдает дробные числа в диапазоне от 0 до 1. Если мы умножим это число на 6 и округлим при помощи команды `Math.round()`, то получим диапазон `0–6`. Добавив единицу, получим диапазон, заданный условиями задачи. Итоговое выражение для получения случайного числа будет выглядеть следующим образом

```
rnd = Math.round(Math.random() * 6) +1
```

Далее организовываем цикл. Поскольку нам известно граничное количество чисел, используем цикл-счетчик «**for**». В теле цикла проверяем число на равенство четырем и, в случае равенства, останавливаем цикл оператором «**break**»

```
for(i=0; i<10; i++){
    rnd = Math.round(Math.random() * 6) +1;
    if(rnd==4) break;
    console.log(rnd);
}
```

Этот код можно набрать в консоли браузера и запустить его нажатием «**Enter**». Для повторного выполнения нажмите стрелку вверх на клавиатуре и снова «**Enter**». Убедитесь, что каждый раз случайные числа выводятся по-разному, останавливаясь при первом выпадении четверки.



Рисунок 7

Поменяем условие задачи — нужно сгенерировать ровно 10 случайных чисел, но без четверок.

В данном случае появление четверки должно прерывать текущую итерацию цикла и запускать следующую, пока не наберется 10 чисел. Т.к. числа случайные и появление четверок непредсказуемо, количество итераций нам заведомо неизвестно. Значит, стоит отдать предпочтение условному циклу. Код тела цикла, в принципе, остается таким же, только в отличие от предыдущего условия задачи, остановка цикла заменяется на остановку итерации, то есть оператор «**break**» заменяется на «**continue**». Найдите ошибку в приведенном коде и исправьте ее перед запуском

```
i=0;
while(i<10) {
    rnd = Math.round(Math.random()*6)+1;
    if(rnd==4) continue;
    console.log(rnd);
}
```

(Здесь снова пропущены команды, влияющие на цикловое условие. В теле цикла необходимо добавить инструкцию «**i++**»).

## Понятие метки

Операторы управления выполнения циклом «**break**» и «**continue**» действуют на тот цикл, в теле которого они применяются. Если анализируемые данные являются многомерными или имеют глубокую степень вложенности, то их перебор, скорее всего, будет организован

при помощи нескольких циклов, вложенных друг в друга. В такой ситуации может появиться необходимость прервать итерацию или выполнение «верхнего» цикла, а не только данного. Для реализации такой возможности в JavaScript предусмотрены метки.

Метка представляет собой идентификатор (имя), сформированный согласно общим правилам именования переменных, после которого указывается двоеточие «`:`». Метка служит для указания определенного места в коде, как правило — цикла. В следующем примере использованы две метки: «`loopI`» и «`loopJ`», относящиеся к циклам с соответствующей цикловой переменной (`i` или `j`).

```
loopI: for(i=0;i<5;i++)
    loopJ: for(j=0;j<5;j++) {
        console.log(i,j);
        if(j==3) break loopI;
    }
```

После оператора «`break`» указывается метка цикла, который нужно прервать. Если метку не указывать или указать «`loopJ`», то при условии (`j==3`) будет прекращаться вложенный цикл, при этом внешний цикл будет продолжать работать. Если же указать метку «`loopI`», то прерываться будет первый цикл, полностью останавливая работу приведенного блока.

Аналогичным образом можно использовать метки и с оператором «`continue`», прерывая итерацию заданного цикла.

## Задание для самостоятельной работы

(в следующих задачах вывод данных может быть реализован как с помощью диалоговых окон, так и средствами консоли разработчика)

1. Напишите скрипт, который запрашивает у пользователя число **N** и выводит все четные числа от **2** до **N** или **N-1**, если **N** нечетное. Например: ввод **10**, вывод **2 4 6 8 10**; ввод **7**, вывод **2 4 6**.
2. Напишите скрипт, который запрашивает у пользователя число **N** и выводит все нечетные числа от **N** (или **N-1**, если **N** четное) до **1** в порядке убывания. Например, ввод **7**, вывод **7 5 3 1**; ввод **10**, вывод **9 7 5 3 1**.
3. Напишите скрипт, который запрашивает у пользователя число **N** и выводит все числа, на которые делится **N**, включая число **1** (примеры: ввод **N=10**, вывод **1, 2, 5; 11**, вывод **1**).
4. Напишите скрипт, который принимает от пользователя величину годовой депозитной ставки (в процентах) и выводит количество лет, по прошествии которых вклад увеличится вдвое.
5. Напишите скрипт, который выводит ровно **10** случайных чисел из диапазона **1–20**, кроме тех, которые делятся на **4**.
6. Из-за утечки из бака охлаждения ежедневно вытекает **10%** налитой воды. При объеме воды менее **10** литров возникает аварийная ситуация. Составьте программу, которая запрашивает у пользователя первоначальный объем воды и рассчитывает, на сколько дней работы этого хватит.



## Unit 3. Циклы

© Денис Самойленко.

© Компьютерная Академия «Шаг», [www.itstep.org](http://www.itstep.org).

Все права на охраняемые авторским правом фото-, аудио- и видеопротивления, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.