Тема 6. Циклы.

хекслет колледж

Цель занятия:

Сформировать у студентов понимание циклических конструкций в JavaScript и научить использовать циклы для многократного выполнения кода.

Учебные вопросы:

1. Понятие цикла в программе
2. Цикл for
3. Цикл while
4. Цикл do ... while
5. Управление циклом
6. Вложенные циклы

1. Понятие цикла в программе

Цикл — это конструкция языка программирования, которая позволяет многократно выполнять один и тот же участок кода, пока выполняется заданное условие. Иными словами, цикл используется тогда, когда одно и то же действие нужно повторить несколько раз, но количество повторений заранее неизвестно или зависит от условия.

Зачем нужны циклы?

Без циклов для повторяющихся действий пришлось бы:

- копировать один и тот же код много раз;
- усложнять и увеличивать программу;
- снижать читаемость и сопровождаемость кода.

Циклы позволяют:

- сократить объём кода;
- избежать дублирования;
- упростить реализацию алгоритмов.

Примеры из практики:

- выводить сообщения по очереди;
- перебирать список элементов;
- выполнять проверку до получения нужного результата;
- считать сумму, количество или среднее значение.

Основные элементы цикла.

Большинство циклов состоит из:

- Начального значения (счётчик или условие начала),
- Условия продолжения (когда цикл выполняется),
- Действия, изменяющего условие или счётчик.

Вывод:

- Цикл — это способ повторного выполнения кода.
- Используется для автоматизации повторяющихся действий.
- Основан на условии.

2. Цикл for

Цикл for используется, когда:

- заранее известно количество повторений;
- нужен счётчик, который изменяется по шагам;
- требуется последовательное выполнение действий.

Это самый часто используемый цикл в JavaScript.

Синтаксис цикла for:

```
for (инициализация; условие; изменение) {
    // тело цикла
}
```

Цикл состоит из трёх частей:

1. Инициализация
Выполняется один раз перед началом цикла.
Обычно создаётся счётчик.

```
let i = 0;
```

2. Условие
Проверяется перед каждой итерацией.
Пока условие истинно (true), цикл продолжается.

3. Изменение счётчика
Выполняется после каждой итерации цикла.

```
i++;
```

Принцип работы:

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

Последовательность выполнения:

- **i = 0** — инициализация;
- проверка **i < 5**;
- выполнение тела цикла;
- **i++** — изменение счётчика;
- повтор шагов 2–4, пока условие ложно.

Использование шага цикла:

```
for (let i = 0; i <= 10; i += 2) {
    console.log(i); // выводит чётные числа
}
```

Область видимости счётчика

При использовании **let** переменная счётчика доступна только внутри цикла:

```
for (let i = 0; i < 3; i++) {
    console.log(i);
}
// console.log(i); // ошибка
```

Типичные ошибки

- неверное условие завершения;
- забытый i++ (бесконечный цикл);
- ошибка на единицу (< вместо <= и наоборот).

Пример ошибки:

```
for (let i = 1; i > 5; i++) {
    console.log(i); // цикл не выполнится ни разу
}
```

Вывод:

- Цикл for применяется при известном количестве повторений.
- Содержит инициализацию, условие и изменение счётчика.
- Требует аккуратной настройки условий.
- Является базой для перебора данных.

3. Цикл while

Цикл while используется, когда:

- количество повторений заранее неизвестно;
- выполнение зависит от условия;
- цикл должен продолжаться, пока условие истинно.

В отличие от **for**, здесь нет явного счётчика в синтаксисе цикла.
Синтаксис цикла **while**:

```
while (условие) {
    // тело цикла
}
```

- условие проверяется перед каждой итерацией;
- если условие ложно — тело цикла не выполнится ни разу.

Принцип работы:

```
let i = 0;

while (i < 5) {
    console.log(i);
    i++;
}
```

Последовательность:

- Проверка условия i < 5;
- Выполнение тела цикла;
- Изменение переменной, влияющей на условие;
- Повтор шагов, пока условие истинно.

Пример: счётчик.

```
let count = 3;

while (count > 0) {
    console.log(count);
    count--;
}
```

Отличие от цикла for

for	while
Подходит, когда известно число итераций	Подходит, когда число итераций неизвестно
Счётчик в заголовке цикла	Управление счётчиком вручную
Компактный синтаксис	Более гибкий

Бесконечный цикл

Если внутри цикла не изменять условие, он станет бесконечным:

```
let i = 0;

while (i < 3) {     // условие всегда верно
    console.log(i); // i не изменяется
}
```

Когда использовать while:

- ожидание определённого состояния;
- повтор действий до выполнения условия;
- работа с вводом данных (на базовом уровне).

Вывод:

- while проверяет условие перед выполнением.
- Может не выполниться ни разу.
- Требует аккуратного управления условием.
- Полезен при неопределённом количестве повторений.

4. Цикл do ... while

Цикл **do ... while** используется в тех случаях, когда необходимо, чтобы тело цикла выполнилось хотя бы один раз, независимо от условия.

В отличие от **while**, проверка условия происходит после выполнения тела цикла.

Основные

Синтаксис цикла **do ... while**:

```
do {
    // тело цикла
} while (условие);
```

Обратите внимание: после **while** (условие) ставится точка с запятой.

Принцип работы:

- Выполняется тело цикла.
- Затем проверяется условие.
- Если условие истинно — цикл повторяется.
- Если условие ложно — цикл завершается.

Пример простого цикла:

```
let i = 5;

do {
    console.log(i);
    i++;
} while (i < 3);
```

Несмотря на то что условие ложно с самого начала, код выполнится один раз.

Типичные случаи использования:

- отображение сообщений минимум один раз;
- повтор запроса данных;
- проверка состояния после выполнения действия.

Вывод:

- **do ... while** гарантирует минимум одно выполнение.
- Используется, когда действие важнее первоначального условия.
- Отличается порядком проверки условия.

5. Управление циклом: **break** и **continue**

Операторы **break** и **continue** позволяют управлять ходом выполнения цикла:

- досрочно завершать цикл;
- пропускать отдельные итерации.

Они используются внутри циклов **for**, **while** и **do ... while**.

Оператор break

break завершает выполнение цикла полностью, независимо от условия.

Пример:

```
for (let i = 1; i <= 10; i++) {
    if (i === 5) {
        break;
    }
    console.log(i);
}
```

После выполнения **break** цикл сразу прекращается.

Оператор continue

continue завершает текущую итерацию и переходит к следующей.

Пример:

```
for (let i = 1; i <= 5; i++) {
    if (i === 3) {
        continue;
    }
    console.log(i);
}
```

Число 3 пропущено, но цикл продолжается.

Вывод:

- break используется для выхода из цикла;
- continue — для пропуска итерации;
- оба оператора упрощают управление логикой циклов.

6. Вложенные циклы

Вложенные циклы — это циклы, расположенные внутри других циклов. При этом внутренний цикл полностью выполняется на каждой итерации внешнего цикла.

Вложенные циклы применяются для:

- работы с таблицами и матрицами;
- обработки двумерных данных;
- генерации повторяющихся структур;
- перебора комбинаций значений.

Пример простого вложенного цикла

```
for (let i = 1; i <= 3; i++) {
    for (let j = 1; j <= 2; j++) {
        console.log("i =", i, ", j =", j);
    }
}
```

Последовательность:

- i = 1 → цикл j выполняется полностью
- i = 2 → цикл j выполняется полностью
- i = 3 → цикл j выполняется полностью

Пример: таблица умножения

```
for (let i = 1; i <= 9; i++) {
    let row = "";
    for (let j = 1; j <= 9; j++) {
        row += i * j + "\t"
    }
    console.log(row);
    console.log();
}
```

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Вывод:

- Вложенные циклы — это циклы внутри циклов.
- Каждый уровень вложенности увеличивает число операций.
- Внутренний цикл выполняется полностью для каждой итерации внешнего.
- Используются для работы с многомерными структурами.
- Требуют аккуратного контроля условий.

Итоги лекции:

- Циклы используются для многократного выполнения одного и того же участка кода.
- Основными циклами в JavaScript являются for, while и do ... while.
- Цикл for применяется, когда количество повторений заранее известно.
- Цикл while используется, когда выполнение зависит от условия и число итераций неизвестно.
- Цикл do ... while гарантирует как минимум одно выполнение тела цикла.
- Каждый цикл основан на условии, которое контролирует его выполнение.
- Операторы break и continue позволяют управлять ходом цикла.
- break завершает цикл полностью, а continue пропускает текущую итерацию.
- Вложенные циклы применяются для обработки сложных и многомерных данных.
- Неправильные условия могут привести к бесконечным циклам, поэтому требуется контроль логики.

Контрольные вопросы:

- Что такое цикл в программе?
- Для чего используются циклы?
- Какие виды циклов существуют в JavaScript?
- В каких случаях целесообразно использовать цикл for?
- В чём отличие циклов for и while?
- Чем цикл do ... while отличается от while?
- Что такое бесконечный цикл и как он возникает?
- Какова роль условия в цикле?
- Для чего используется оператор break?
- В каких ситуациях применяется оператор continue?
- Что называется вложенным циклом?
- Как выполняется внутренний цикл по отношению к внешнему?
- Какие ошибки часто допускают при работе с циклами?
- Почему важно аккуратно задавать условия выхода из цикла?

Домашнее задание:

https://ru.hexlet.io/courses/js-basics

хекслет колледж

@HEXLY.KZ