**Лекція 3. Цикли в JS**

1. **Цикли *while, do ... while* та *for***
2. **Конструкція *switch***
3. **Цикли *while***

При написанні скриптів часто постає завдання зробити однотипну дію багато разів. Наприклад, вивести товари зі списку один за іншим. Або просто перебрати всі числа від 1 до 10 і для кожного виконати однаковий код. Для багаторазового повторення однієї ділянки коду - передбачені цикли.

Цикл *while* має вигляд:

while (умова) {

// код, тіло циклу

}

Поки умова вірна - виконується код з тіла циклу.

Наприклад, цикл нижче виводить *i* поки *i <3:*

 var i = 0;

while (i < 3) {

alert( i );

i++;

}

Повторення циклу по-науковому називається «ітерація». Цикл в прикладі вище робить три ітерації.

Якби *i ++* в коді вище не було, то цикл виконувався б (в теорії) вічно. На практиці, браузер виведе повідомлення про «зависший» скрипт і користувач його зупинить.

Нескінченний цикл можна зробити і простіше:

while (true) {

// ...

}

Умова в дужках інтерпретується як логічне значення, тому замість *while* (*i! = 0*) зазвичай пишуть *while (i)*:

var i = 3;

while (i) { // при i, рівному 0, значення в дужках буде false і цикл зупиниться

alert( i );

i--;

}

**Цикл do ... while**

Перевірку умови можна поставити під тілом циклу, використовуючи спеціальний синтаксис do..while:

do {

// тіло циклу

} while (умова);

Цикл, описаний таким чином, спочатку виконує тіло, а потім перевіряє умову. Наприклад:

 var i = 0;

do {

alert( i );

i++;

} while (i < 3);

Синтаксис *do..while* рідко використовується, тому що звичайний *while* наочніше - в ньому не доводиться шукати очима умову і ламати голову, чому вона перевіряється саме в кінці.

**Цикл *for***

Найчастіше застосовується цикл *for*. Виглядає він так:

for (початок; умова; крок) {

// ... тіло циклу ...

}

Приклад циклу, який виконує *alert (i)* для *i* від 0 до 2 включно (до 3):

var i;

for (i = 0; i < 3; i++) {

alert( i );

}

Тут: початок - *i = 0*, умова - *i <3*, крок - *i ++,* тіло - *alert (i)*, тобто код всередині фігурних дужок (вони не обов'язкові, якщо тільки одна операція).

Цикл виконується так:

1. Початок: *i = 0* виконується один-єдиний раз, під час заходу в цикл.

2. Умова: *i <3* перевіряється перед кожною ітерацією і при вході в цикл, якщо вона порушена, то відбувається вихід.

3. Тіло: *alert (i).*

4. Крок: *i ++* виконується після тіла на кожній ітерації, але перед перевіркою умови.

5. Йти на крок 2.

Іншими словами, потік виконання: початок → (якщо умова → тіло → крок) → (якщо умова → тіло → крок) → ... і так далі, поки вірна умова.

*На замітку:*

У циклі також можна визначити змінну:

for (var i = 0; i < 3; i++) {

alert(i); // 0, 1, 2

}

Цю змінну буде видно і за межами циклу, зокрема, після закінчення циклу *i* буде дорівнювати 3.

Будь-яка частина *for* може бути пропущена. Наприклад, можна прибрати початок. Цикл в прикладі нижче повністю ідентичний наведеному вище:

var i = 0;

for (; i < 3; i++) {

alert( i ); // 0, 1, 2

}

Можна прибрати і крок:

var i = 0;

for (; i < 3;) {

alert( i );

// цикл перетворився на аналог while (i<3)

}

А можна і взагалі прибрати все, отримавши нескінченний цикл:

for (;;) {

// буде виконуватися вічно

}

При цьому самі крапки з комою; обов'язково повинні бути присутніми, інакше буде помилка синтаксису.

**Переривання циклу: *break***

Вийти з циклу можна не тільки під час перевірки умови а й, взагалі, в будь-який момент. Цю можливість забезпечує директива *break*.

Наприклад, наступний код підраховує суму чисел, що вводяться до тих пір, поки користувач їх вводить, а потім - видає:

var sum = 0;

while (true) {

var value = +prompt("Введіть число", '');

if (!value) break; // (\*)

sum += value;

}

alert( 'Сума: ' + sum );

Директива *break* в рядку (\*), якщо відвідувач нічого не ввів, повністю припиняє виконання циклу і передає управління рядку за його тілом, тобто на *alert*.

Взагалі, поєднання «нескінченний цикл + break» - відмінна штука для тих ситуацій, коли умова, за якою потрібно перерватися, знаходиться не на початку/наприкінці циклу, а посередині.

**Наступна ітерація: *continue***

Директива *continue* припиняє виконання поточної ітерації циклу.

Вона - в деякому роді «молодша сестра» директиви break: перериває не весь цикл, а тільки поточне виконання його тіла, як ніби воно закінчилося.

Її використовують, якщо зрозуміло, що на поточному повторі циклу робити більше нічого.

Наприклад, цикл нижче використовує *continue*, щоб не виводити парні значення:

for (var i = 0; i < 10; i++) {

if (i % 2 == 0) continue;

alert(i);

}

Для парних *i* спрацьовує *continue*, виконання тіла припиняється і керування передається на наступний прохід *for*. Директива *continue* дозволяє обійтися без дужок.

Цикл, який обробляє тільки непарні значення, міг би виглядати так:

for (var i = 0; i < 10; i++) {

if (i % 2) {

alert( i );

}

}

З технічної точки зору він повністю ідентичний. Дійсно, замість *continue* можна просто загорнути дії в блок *if.* Однак, ми отримали додатковий рівень вкладеності фігурних дужок. Якщо код всередині *if* довший, то це погіршує читаність, на відміну від варіанту з *continue*.

Не можна використовувати *break / continue* праворуч від оператора *"?"*

Зазвичай ми можемо замінити *if* на оператор знак '*?*'.

Тобто, запис:

if (умова) {

a();

} else {

b();

}

... аналогічний запису:

умова ? a() : b();

В обох випадках в залежності від умови виконується або *a ()* або *b ().* Але різниця полягає в тому, що оператор знак '*?*', використаний у другому записі, повертає значення. Синтаксичні конструкції, які не повертають значень, можна використовувати в операторі '*?*'.

До таких відносяться більшість конструкцій і, зокрема, *break / continue*.

Тому такий код призведе до помилки:

(i > 5) ? alert(i) : continue;

Втім, як уже говорилося раніше, оператор знак '*?*' не варто використовувати таким чином. Це - всього лише ще одна причина, чому для перевірки умови кращий *if.*

**Мітки для break / continue**

Буває потрібно вийти одночасно з декількох рівнів циклу. Наприклад, всередині циклу по *i* знаходиться цикл по *j*, і при виконанні деякої умови ми б хотіли вийти з обох циклів відразу:

 outer: for (var i = 0; i < 3; i++) {

for (var j = 0; j < 3; j++) {

var input = prompt('Значення в координатах'+i+','+j, '');

// якщо скасування введення або порожній рядок -

// завершити обидва циклу

if (!input) break outer; // (\*)

}

}

alert('Готово!');

У коді вище для цього використана мітка. Мітка має вигляд "*ім'я:*", ім'я має бути унікальним. Вона ставиться перед циклом, ось так:

outer: for (var i = 0; i < 3; i++) { ... }

Можна також виносити її в окремий рядок:

outer:

for (var i = 0; i < 3; i++) { ... }

Виклик *break …outer* шукає найближчий зовнішній цикл з такою міткою і переходить в його кінець.

В наведеному вище прикладі це означає, що буде розірваний самий зовнішній цикл і управління перейде на *alert*.

Директива *continue* також може бути використана з міткою, в цьому випадку управління перейде на наступну ітерацію циклу з міткою.

**Висновки**

JavaScript підтримує три види циклів:

*while* - перевірка умови перед кожним виконанням.

*do..while* - перевірка умови після кожного виконання.

*for* - перевірка умови перед кожним виконанням, а також додаткові налаштування.

Щоб організувати нескінченний цикл, використовують конструкцію *while* (*true*). При цьому він, як і будь-який інший цикл, може бути перерваний директивою *break*.

Якщо на даній ітерації циклу робити більше нічого не треба, але повністю припиняти цикл не слід - використовують директиву *continue*.

Обидві ці директиви підтримують «мітки», які ставляться перед циклом. Мітки - єдиний спосіб для *break / continue* вплинути на виконання зовнішнього циклу.

Зауважимо, що мітки не дозволяють перейти в довільне місце коду, в JavaScript немає такої можливості.

**2**. **Конструкція *switch***

Конструкція *switch* замінює собою відразу кілька *if*.

Вона являє собою більш наочний спосіб порівняти вираз відразу з декількома варіантами. Виглядає вона так:

switch(x) {

case 'value1': // if (x === 'value1')

...

[break]

case 'value2': // if (x === 'value2')

...

[break]

default:

...

[break]

}

Змінна *x* перевіряється на сувору рівність першому значенню *value1*, потім другому *value2* і так далі.

Якщо відповідність встановлено - *switch* починає виконуватися від відповідної директиви *case* і далі, до найближчого *break* (або до кінця *switch*). Якщо жоден *case* не співпав - виконується (якщо є) варіант *default*. При цьому *case* називають варіантами *switch*.

Приклад використання *switch*:

var a = 2 + 2;

switch (a) {

case 3:

alert( 'Мало' );

break;

case 4:

alert( 'В точку!' );

break;

case 5:

alert( 'Перебільшення' );

break;

default:

alert( 'Я таких значень не знаю' );

}

Тут оператор *switch* послідовно порівняє *a* з усіма варіантами з *case*.

Спочатку 3, потім - так як немає збігу - 4. Збіг знайдено, буде виконаний цей варіант, з рядка *alert ( 'В точку!')* і далі, до найближчого break, який перерве виконання.

Якщо *break* немає, то виконання піде нижче за наступними *case*, при цьому інші перевірки ігноруються.

Приклад без *break*:

var a = 2 + 2;

switch (a) {

case 3:

alert( 'Мало' );

case 4:

alert( 'В точку!' );

case 5:

alert( 'Перебільшення' );

default:

alert( 'Я таких значень не знаю' );

}

В наведеному вище прикладі послідовно виконуються три *alert*:

alert( 'В точку!' );

alert( 'Перебільшення' );

alert( 'Я таких значень не знаю' );

У *case* можуть бути будь-які вирази, в тому числі включати в себе змінні і функції.

Наприклад:

var a = 1;

var b = 0;

switch (a) {

case b + 1:

alert( 1 );

break;

default:

alert('ні, виконається варіант вище')

}

**Групування *case***

Кілька значень *case* можна групувати.

У прикладі нижче *case 3* і *case 5* виконують один і той же код:

var a = 2+2;

switch (a) {

case 4:

alert('Вірно!');

break;

case 3: // (\*)

case 5: // (\*\*)

alert('Невірно!');

alert('Трохи помилились, буває.');

break;

default:

alert('Дивний результат, дуже дивний');

При *case 3* виконання йде з рядка (\*), при *case 5* - з рядка (\*\*).

Наступний приклад приймає значення від користувача.

var arg = prompt("Введіть arg?")

switch (arg) {

case '0':

case '1':

alert( 'Один або нуль' );

case '2':

alert( 'Два' );

break;

case 3:

alert( 'Ніколи не виконається' );

default:

alert('Невідоме значення: ' + arg)

}

Що буде виведено при введенні числа 0? Числа 1? 2? 3?

* При введенні *0* виконається перший *alert*, далі виконання продовжиться вниз до першого *break* і виведе другий *alert* ( *'Два'*). Тобто двічі виведеться *alert*.
* При введенні *1* відбудеться те ж саме.
* При введенні *2*, *switch* перейде до *case '2'*, і спрацює єдиний *alert* *( 'Два').*
* При введенні *3*, *switch* перейде на *default*. Це тому, що *prompt* повертає рядок '*3*', а не число. Типи різні. Оператор *switch* передбачає сувору рівність ===, так що збігу не буде.