Цикли: while і for

При написанні скриптів часто постає завдання зробити однотипні дії багато разів.

Наприклад, вивести товари зі списку один за одним чи просто запустити один і той же код для кожного числа від 1 до 10.

Цикл – це спосіб повторити один і той же код кілька разів.

Цикл “while”

Цикл `while` має такий синтаксис:

```
while (умова) {
  // код
  // так зване "тіло циклу"
}
```

Доки умова є `вірною`, виконується `код` із тіла циклу.

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

```
let i = 0;
while (i < 3) { // показується 0, далі 1, потім 2
  alert( i );
  i++;
}
```

Одне виконання циклу називається *ітерацією*. Цикл в зразку вище робить три ітерації.

Якщо `i++` пропустити в коді вище, то цикл виконувався б (в теорії) вічно. На практикі, браузери надають способи зупинити такі цикли, і на серверному JavaScript(Node.js), ми можемо знищити цей процес

Будь-який вираз або змінна може бути умовою циклу, а не тільки порівняння (`a < 5` чи `b !== 10`). Умова виконується і конвертується у логічне значення.

Наприклад, коротший спосіб написання `while (i != 0)` відповідає `while (i)`:

```
let i = 3;
while (i) { // коли i буде 0, умова стане невірною, і цикл зупиниться
  alert( i );
  i--;
}
```

Curly braces are not required for a single-line body

Якщо тіло цикла має тільки одну операцію, ми можемо опустити фігурні дужки `{…}`:

```
let i = 3;
while (i) alert(i--);
```

Цикл “do…while”

Перевірка умови може бути переміщена *нижче* тіла циклу використовуючи `do..while` синтаксис:

```
do {
  // тіло циклу
} while (умова);
```

Цикл спочатку виконує тіло, а потім перевіряє умову, і поки умова є `true`, цикл виконується знову і знову.

Наприклад:

```
let i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
```

Цю форму синтаксису слід використовувати лише тоді, коли ви хочете, щоб тіло циклу виконалось **хоча б один раз**, незалежно від умови. Зазвичай, інша форма є більш бажаною `while(…) {…}`

Цикл “for”

Цикл `for` є більш складним, але також є часто використовуваним циклом.

Виглядає він так:

```
for (початок; умова; крок) {
  // ... тіло циклу ...
}
```

Давайте дізнаємось про значення цих трьох частин за зразком. Цикл нижче виконує `alert(i)` для `i` від `0` до `3` (але не включаючи це число `3`)

```
for (let i = 0; i < 3; i++) { // показується 0, далі 1, потім 2
  alert(i);
}
```

Давайте розглянемо цикл `for` по частинах:

Назва частини		
початок	`let i = 0`	Виконується один раз, при вході в цикл.
умова	`i < 3`	Перевіряється перед кожною ітерацією циклу. Якщо умова невірна, цикл зупиняється.
тіло	`alert(i)`	Виконується знову і знову, поки умова є правдивою (`true`).
крок	`i++`	Виконується після тіла на кожній ітерації, але перед перевіркою умови.

Загальний алгоритм циклу працює так:

```
*Початок* виконання
→ (Якщо *умова* == true → виконати тіло і виконати крок)
→ (Якщо *умова* == true → виконати тіло і виконати крок)
→ (Якщо *умова* == true → виконати тіло і виконати крок)
→ ...
```

Спочатку один раз виконується `початок`, потім при кожній ітерації: перевіряється `умова`, виконується `тіло` циклу та `крок`.

Якщо ви новачок у циклах, вам може допомогти покрокове виконання цього прикладу на аркуші паперу.

Ось що відбувається в нашому випадку:

```
// for (let i = 0; i < 3; i++) alert(i)

// Початок виконання
let i = 0
// Якщо умова == true → виконати тіло і виконати крок
if (i < 3) { alert(i); i++ }
// Якщо умова == true → виконати тіло і виконати крок
if (i < 3) { alert(i); i++ }
// Якщо умова == true → виконати тіло і виконати крок
if (i < 3) { alert(i); i++ }
// ...кінець, тому що зараз i == 3
```

Вбудоване оголошення змінної

В цьому прикладі всередині циклу оголошена змінна `i`, яка виконує функцію лічильника. Це так зване «вбудоване» оголошення змінної. Такі змінні доступні лише всередині циклу.

```
for (let i = 0; i < 3; i++) {
  alert(i); // 0, 1, 2
}
alert(i); // помилка, немає такої змінної
```

Замість оголошення нової змінної, ми можемо використовувати існуючу:

```
let i = 0;

for (i = 0; i < 3; i++) { // використовуємо існуючу змінну
  alert(i); // 0, 1, 2
}

alert(i); // 3, змінна доступна, тому що вона оголошена поза циклом
```

Пропуск частин в “for”

Будь-яку частину `for` можна пропустити.

Наприклад, ми можемо опустити `початок`, якщо нам не потрібно нічого робити перед стартом циклу.

Ось так:

```
let i = 0; // ми вже маємо оголошену змінну і присвоєне значення

for (; i < 3; i++) { // немає необхідності в "початку"
  alert( i ); // 0, 1, 2
}
```

Ми також можемо видалити частину `крок`:

```
let i = 0;

for (; i < 3;) {
  alert( i++ );
}
```

Це робить цикл ідентичним до `while (i < 3)`.

Можна взагалі забрати все, отримавши нескінченний цикл:

```
for (;;) {
  // буде вічно повторюватися
}
```

Зауважте, що ці двокрапки `;` повинні бути, інакше виникне синтаксична помилка.

Переривання циклу: “break”

Зазвичай, цикл завершується, коли умова стає `false`.

Але ми можемо в будь-який момент вийти з циклу, використавши спеціальну директиву `break`.

Наприклад, наступний код запитує в користувача число доти, поки користувач їх вводить. Після того, як користувач не ввів число — цикл завершується (директивою “break”) і виводить суму чисел:

```
let sum = 0;

while (true) {

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

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

  sum += value;

}
alert( 'Сума: ' + sum );
```

Директива `break` в рядку `(*)` спрацьовує тоді, коли користувач вводить порожній рядок або скасовує введення. Ця директива негайно завершує виконання циклу і передає контроль наступному рядку за циклом, тобто на `alert`.

Комбінація «нескінченний цикл + `break`» — чудова річ для тих ситуацій, коли умова для переривання знаходиться не на початку або кінці циклу, а всередині (або навіть в декількох місцях) тіла циклу.

Продовження з наступної ітерації

Директива `continue` — це “полегшена версія” `break`. Вона не зупиняє весь цикл. Натомість, вона зупиняє поточну ітерацію і починає виконання циклу спочатку з наступної ітерації (якщо умова циклу досі вірна).

Її зручно використовувати коли закінчили з поточною ітерацією і хочемо продовжити з наступної.

Цикл нижче використовує `continue` щоб вивести лише непарні значення:

```
for (let i = 0; i < 10; i++) {

  // якщо умова справджується, тоді пропускаємо решту тіла циклу і починаємо з наступної ітераці
  if (i % 2 == 0) continue;

  alert(i); // 1, потім 3, 5, 7, 9
}
```

Для парних значень змінної `i`, директива `continue` зупиняє виконання тіла циклу і передає контроль наступній ітерації в `for` (в цьому випадку це буде наступне число). Таким чином функція `alert` викликається лише для непарних значень змінної `i`.

Директива `continue` допомагає зменшити рівень вкладеності

Цикл, який показує непарні значення може виглядати так:

```
for (let i = 0; i < 10; i++) {

  if (i % 2) {
    alert( i );
  }

}
```

З технічної точки зору, цей приклад ідентичний тому що вище. Звичайно, ми можемо просто обгорнути код в блок `if` замість використання `continue`.

Але побічним ефектом цього буде створення ще одного рівня вкладеності (виклик `alert` всередині фігурних дужок). Якщо код всередині `if` буде більшим за декілька рядків, то це може ускладнити загальну читабельність коду.

Директиви `break/continue` праворуч від '?' не працюють

Майте на увазі, що такі синтаксичні конструкції, які не є виразами, не можуть використовуватися з тернарним оператором `?`. Власне, такі директиви як `break/continue` там не дозволені.

Наприклад, якщо взяти код:

```
if (i > 5) {
  alert(i);
} else {
  continue;
}
```

…і переробити його з використанням знака питання:

```
(i > 5) ? alert(i) : continue; // використання continue в недозволеному місці
```

…то такий код перестане працювати: виникне синтаксична помилка.

Це ще одна причина не використовувати для умов оператор знака питання `?`, замість повноцінного `if`.

Мітки для break/continue

Деколи нам потрібно вийти з кількох вкладених циклів.

Наприклад, в коді нижче є 2 цикли, які проходяться по змінних `i` та `j`, і запитують в користувача координати `(i, j)` від `(0,0)` до `(2,2)`:

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

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

    let input = prompt(`Значення в координатах (${i},${j})`, '');

    // що якщо ми хочемо вийти звідси відразу до 'Готово!' (в функцію alert нижче)?
  }
}

alert('Готово!');
```

Нам потрібен спосіб зупинити ці два цикли, якщо користувач скасує введення.

Звичайний `break` після `input` перерве лише внутрішній цикл, а нам цього недостатньо. Ось тут нам стануть в пригоді мітки для циклів!

Мітка складається з ідентифікатора та двокрапки перед циклом:

```
labelName: for (...) {
  ...
}
```

Вираз `break <labelName>` в циклі нижче шукає найближчий цикл з заданою міткою і переходить в його кінець:

```
outer: for (let i = 0; i < 3; i++) {

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

    let input = prompt(`Значення в координатах (${i},${j})`, '');

    // якщо порожній рядок або Скасувати, тоді вихід з обох циклів
    if (!input) break outer; // (*)

    // зробити щось із значенням...
  }
}
alert('Готово!');
```

В коді вище, вираз `break outer` шукає зверху мітку `outer` і перериває цикл, позначений цією міткою.

Тож виконання коду перейде з місця переривання циклу (позначене `(*)`) до функції `alert('Готово!')`.

Мітку можна перемістити в новий рядок:

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

Також мітками можна використовувати з директивою `continue`. В такому разі, виконання коду перестрибне на наступну ітерацію поміченого циклу.

Міткам не дозволено “стрибати” будь-де

Ми не можемо використовувати мітки, щоб стрибати в довільне місце в коді.

Наприклад, ось таке неможливо зробити:

```
break label; // стрибнути в мітку label нижче (не спрацює)

label: for (...)
```

Директива `break` повинна бути всередині блоку з кодом. Технічно, підійде навіть іменований блок. Наприклад:

```
label: {
  // ...
  break label; // працює
  // ...
}
```

…Однак, 99.9% часу `break` використовується всередині циклів, як ми бачили в прикладах вище.

Виклик `continue` можливий лише всередині циклу.

Підсумки

Ми вивчили 3 типи циклів:

- `while` – умова перевіряється перед кожною ітерацією.
- `do..while` – умова перевіряється після кожної ітерації.
- `for (;;)` – умова перевіряється перед кожною ітерацією. Можна додатково налаштовувати.

Щоб зробити “нескінченний” цикл, використовують вираз `while(true)`. Такі цикли, як і будь-які інші, можна зупинити директивою `break`.

Якщо нам не потрібно нічого виконувати в поточній ітерації циклу, ми можемо пропустити цю ітерацію за допомогою директиви `continue`.

`break/continue` підтримують мітки перед циклом. Лише за допомогою міток `break/continue` ми можемо перервати вкладений цикл.

Завдання

Challenge 1 **Останнє значення циклу**

Яке останнє значення буде виведено на екран? Чому?

```
let i = 3;

while (i) {
  alert( i-- );
}
```

Challenge 2 **Яке значення виведе цикл "while"?**

Запишіть для кожного циклу значення, які він виведе. Потім порівняйте з відповіддю.

Чи виводять обидва цикли однакові значення?

1.

Префіксна форма `++i`:

```
let i = 0;
while (++i < 5) alert( i );
```

2.

Постфіксна форма `i++`

```
let i = 0;
while (i++ < 5) alert( i );
```

Challenge 3 **Яке значення виведе цикл "for"?**

Для кожного циклу запишіть, які значення він виведе. Потім порівняйте з відповіддю.

Обидва цикли виведуть одинакові числа чи ні?

1.

Постфіксна форма:

```
for (let i = 0; i < 5; i++) alert( i );
```

2.

Префіксна форма:

```
for (let i = 0; i < 5; ++i) alert( i );
```

Challenge 4 Виведіть парні числа

Виведіть парні числа від `2` до `10`, використовуючи цикл `for`.

Challenge 5 Замініть цикл "for" на "while"

Замініть цикл `for` на `while` так, щоб поведінка не змінилася (щоб вивід залишився той самий).

```
for (let i = 0; i < 3; i++) {
  alert( `число ${i}!` );
}
```

Challenge 6 Повторяти цикл, доки ввід невірний

Напишіть цикл, який пропонує `prompt` ввести число більше за `100`. Якщо відвідувач введе інше число – попросити ввести ще раз, і так далі.

Цикл повинен запитувати число доти, доки відвідувач не введе число, більше за `100`, або не скасує ввід/введе порожній рядок.

Ми припускаємо, що відвідувач вводитиме лише числа. В цьому завданні не обов'язково реалізовувати оброблення не-числового введення.

Challenge 7 Вивести прості числа

Просте число – це натуральне число, яке має два дільники (`1` і саме число).

Інакше кажучи, `n > 1` – просте, якщо воно більше за `1` і ділиться без остачі на `1` та `n`.

Наприклад, число `5` – просте, тому що воно не ділиться без остачі на `2`, `3` і `4`. Воно ділиться без остачі лише на `1` і на `5`.

Напишіть код, який виводить всі прості числа в діапазоні від `2` до `n`.

Для `n = 10` результат повинен бути `2,3,5,7`.

P.S. Код також повинен легко модифікуватися для будь-якого числа `n`.