УМОВИ І ЦИКЛИ

План:

1. Умови і цикли в JavaScript
    1.2. Конструкція if-else
    1.3. Робота з логічними змінними
    1.4. Конструкція switch-case
2. Робота з циклами for і while
    1.2. Цикл while
    1.3. Цикл for
    1.4. Цикл без тіла
    1.5. Кілька команд в циклі for
    1.6. Цикл for для масивів
    1.7. Цикл for-in
    1.8. Інструкція break
    1.9. Інструкция continue

Умови і цикли в JavaScript

Конструкція if-else

Для того, щоб написати корисну програму, необхідно більше, ніж лише змінні. Нам потрібен механізм, що дозволяє виконувати код залежно від певних умов. Наприклад: якщо значення змінної менше нуля, вивести 'негативно', якщо більше нуля - вивести 'позитивно'. У JavaScript для таких завдань використовується конструкція if, що дозволяє виконати певний код, якщо виконується певна умова:

if (логічний вираз) {

```
  Цей код виконатися,
  якщо логічне вираження вірно (тобто дорівнює true)
} else {
  Цей код виконатися,
  якщо логічне вираження невірно (тобто дорівнює false)
}
```

Зверніть увагу на те, що блок else не обов'язковий.

Логічний вираз - це запитання, яке задається в JavaScript. Наприклад, щоб запитати "чи змінна a більше нуля", ми можемо написати так: a > 0.

Приклади роботи:

```
var a = 3;
/*
  Якщо змінна a більше нуля, то виведи 'правильно',
  інакше (якщо менше або дорівнює нулю) виведи 'невірно'
*/
if (a > 0) {alert('Вірно!');} else {alert('Невірно!');} //виведе 'Вірно!'
var a = -3;
/*
  Якщо змінна a більше або дорівнює нулю, то виведи ‘вірно’, інакше (якщо менше нуля) виведи 'невірно'
*/
if (a >= 0) {alert('Вірно!');} else {alert('Невірно!');} //виведе 'Невірно!'
```

Скорочений синтаксис

У разі, якщо в фігурних дужках if або else буде тільки один вираз, можна ці фігурні дужки не писати:

```
// Повний варіант:
if (a == 0) {alert('Вірно!');} else {alert('Невірно!');}
```

```
// Видалимо дужки після if:
if (a == 0) alert('Вірно!'); else {alert('Невірно!');}
// Видалимо дужки після else:
if (a == 0) {alert('Вірно!');} else alert('Невірно!');

/*
  Видалимо дужки і після if, і після else
  (зверніть увагу на точку з комою - вона залишилася):
*/
if (a == 0) alert('Вірно!'); else alert('Невірно!');
```

Рівність за значенню і типу

Для того, щоб порівняти на рівність слід використовувати оператор подвійне дорівнює ==, а не одиночне, тому що одиночне одно зарезервовано за присвоюванням.

```
var a = 0;
/*
  Якщо змінна a дорівнює нулю, то виведи вірно,
  інакше (якщо не дорівнює нулю) виведи 'невірно'
*/
if (a == 0) alert('Вірно!'); else alert('Невірно!'); //виведе 'Верно!'
```

Наступний приклад:

```
var a = 0;
/*
  Ми думаємо воно працює так: якщо змінна a дорівнює нулю, то виведи вірно, інакше (якщо не дорівнює нулю) виведи 'невірно'.
  Насправді воно працює так: змінної a присвоїти 1, якщо вдалося привласнити-то виведи 'вірно', інакше (якщо не вдалося привласнити) виведи 'невірно'.
*/
```

```
if (a = 1) alert('Вірно!'); else alert('Невірно!'); //завжди буде виводити 'Вірно!'
```

Крім оператора == існує ще й оператор===. Їх відмінність в тому, що = = = порівнює не тільки за значенням, але і за типом, а = = порівнює тільки за значенням.

Приклади:

```
var a = '0'; // змінна a являє собою рядок, а не число 0
if (a == 0) alert('Вірно!'); else alert('Невірно!');
/*
  Виведе 'Вірно!', так як перевіряється тільки значення, але не тип.
  Тому '0' дорівнює 0.
*/
var a = '0'; //змінна a являє собою рядок, а не число 0
if (a === 0) alert('Вірно!'); else alert('Невірно!');
/*
  Виведе 'Невірно!', так як рядок '0' не дорівнює числу 0 при порівнянні за типом.
*/
```

Не дорівнює

Для того, щоб використовувати 'не дорівнює', існують оператори != і !==. Перший ігнорує різницю в типах, а другий – ні.

```
var a = 0;
/*
  Якщо змінна a НЕ дорівнює нулю, то виведи вірно, інакше (якщо дорівнює нулю) виведи 'невірно'
*/
if (a != 0) alert('Вірно!'); else alert('Невірно!'); //виведе 'Невірно!', так як a дорівнює 0
a = 1;
/*
```

```
Якщо змінна a НЕ дорівнює нулю, то виведи вірно, інакше (якщо дорівнює нулю) виведи 'невірно'
*/
if (a != 0) alert('Вірно!'); else alert('Невірно!'); //виведе 'Вірно!', так як a дорівнює 1
var a = '0';
/*
  Якщо змінна a НЕ дорівнює нулю, то виведи вірно, інакше (якщо дорівнює нулю) виведи 'невірно'
*/
if (a != 0) alert('Вірно!'); else alert('Невірно!');
/*
  Виведе 'Невірно!', так як a дорівнює '0',
  а відмінність в типах ігнорується.
*/
var a = '0';
/*
  Якщо змінна a НЕ дорівнює нулю, то виведи вірно, інакше (якщо дорівнює нулю) виведи 'невірно'
*/
if (a !== 0) alert('Вірно!'); else alert('Невірно!');
/*
  Виведе 'Вірно!', так як a дорівнює '0',
  а це не дорівнює 0 при порівнянні за типом.
*/
```

Всі операції порівняння

Можливі операції порівняння, які можна використовувати всередині if:

a == b	a дорівнює b

a === b	a дорівнює b і вони однакові за типом
a != b	a не дорівнює b
a !== b	a не дорівнює b або a дорівнює b, але вони різні за типом
a < b	a менше b
a > b	a більше b
a <= b	a менше або дорівнює b
a >= b	a більше або дорівнює b

Іноді вам може знадобитись створити складну умову, наприклад, якщо ви просите користувача ввести місяць свого народження, то вам потрібно перевірити, що введене число є більшим або рівним 1 і меншим або рівним 12 (так як рік містить у собі 12 місяців).

Для цього використовуються логічні оператори && (логічне і) і || (логічне АБО).

```
var a = 3;
var b = -3;
// Якщо a більше нуля і b одночасно менше нуля, то...
if (a > 0 && b < 0) alert('Вірно!'); else alert('Невірно!'); //виведе 'Вірно!'
var a = 3;
// Якщо a більше або дорівнює 1 і менше або дорівнює 12 то...
if (a >= 1 && a <= 12) alert('Вірно!'); else alert('Невірно!'); //виведе 'Вірно!'
var a = -3;
var b = -3;
```

```
/*
  Якщо a більше нуля або b менше нуля-хоча б один з них, то...
  виведе 'Вірно!', так як хоча a і не більше нуля,
  але одна з умов - b < 0 - виконається!
*/
if (a > 0 || b < 0) alert('Верно!'); else alert('Невірно!');
```

Робота з логічними змінними

Багато функцій у JavaScript повертають значення true (істина) або false (неправда) під час своєї роботи. Ці значення є досить зручними для програмістів, але можуть бути складними для новачків. Давайте уявимо, що змінна a має значення true. У такому випадку, конструкцію if можна записати так:

```
var a = true;
//Если a равно true, то...
if (a == true) alert('Вірно!'); else alert('Невірно!');
/*
  Виведе 'Верно!', так як a дорівнює true.
*/
```

Так як такі порівняння досить поширені в JavaScript, існує спеціальний прийом, який полегшує роботу. Цим прийомом є заміна конструкції a == true на більш просту a в конструкції if.

Наприклад, замість if (a == true), можна написати if (a) і вона буде працювати аналогічно.

Слід користуватися другою конструкцією, так як вона простіше.

```
/*
  Замінимо a == true на більш просту:
```

```
  замість if (a == true) напишемо if (a):
*/
var a = true;
// Якщо A дорівнює true, то...
if (a) alert('Вірно!'); else alert('Невірно!'); //виведе 'Вірно!', так як a дорівнює true
var a = true;
//Якщо a НЕ true (тобто false!), то...
if (!a) alert('Вірно!'); else alert('Невірно!'); //виведе 'Невірно!', так як a дорівнює true
```

Також зверніть увагу на наступні приклади:

```
//Даний вираз завжди буде виводити 'Вірно'
if (true) alert('Верно!'); else alert('Невірно!');
// Даний вираз завжди буде виводити 'Невірно'
if (false) alert('Вірно!'); else alert('Невірно!');
// Даний вираз завжди буде виводити 'Невірно'
if (!true) alert('Вірно!'); else alert('Невірно!');
 // Даний вираз завжди буде виводити 'Вірно'
if (!false) alert('Верно!'); else alert('Невірно!');
```

Укладені if

Представимо, що нам потрібно зробити так: якщо змінна a не вказана, то вивести "Введіть a", а якщо вона вказана, то перевірити, чи є a більшою за 0. Якщо так, то вивести "Більше нуля!", якщо ні – вивести "Менше нуля".

Для цього нам знадобиться дві конструкції if, одна всередині іншої. Перша перевіряє, чи визначена змінна a. Якщо вона не визначена, ми отримаємо повідомлення "Введіть a". Якщо ж вона визначена, то ми

переходимо до другої конструкції if, що перевіряє, чи є a більшою за 0. Якщо так, то виводиться "Більше нуля!", якщо ні – "Менше нуля!".

Ось як це можна зробити:

```
if (a === undefined) { // перевірка, чи визначена змінна a
  alert('Введіть a!');
} else { // якщо a визначена
  if (a > 0) { // перевірка, чи a більше за 0
        alert('Більше нуля!');
  } else {
        alert('Менше нуля!');
  }
}
```

Конструкція else if

Попередній приклад мав недолік – багато фігурних дужок. Але можна скористатися конструкцією else if, яка зменшує їх кількість. Вона поєднує в собі else і наступний if:

```
// Використання else if для розв'язання попереднього завдання:

// Рішення попереднього завдання через конструкцію else if:
if (a === undefined) {
  alert('Введіть a!');
} else if (a > 0) {
  alert('Більше нуля!');
 } else {
  alert('Менше нуля!');
}
```

Можна використовувати кілька else if, але зловживати цим не варто (краще буде скористатися конструкцією switch-case).

Кілька if

Нехай у нас є таке завдання: *сайт підтримує 3 мови-російська, англійська, німецька. Змінна lang може приймати 3 значення – 'ua', 'en' і 'de'. Залежно від значення змінної lang слід вивести фразу на одній з мов.*

Рішення: можна було б скористатися вкладеними if або else if. Виглядало б це приблизно так:

```
//Рішення задачі через else if – не найвдаліше:
if (lang == 'ua') { // фраза українською
  alert('Український текст');
} else if (lang == 'en') { // фраза англійською
  alert('Англійський текст');
} else if (lang == 'de') { // фраза німецькою
  alert('Німецький текст');
}
```

Таке рішення не дуже красиве – блок else тут не потрібен! Найпростіше буде написати не один довгий if з декількома else, а кілька if взагалі без else:

```
//Решение задачи через несколько if – оно намного лучше:
if (lang == 'ua') {
  alert('Український текст');
}
if (lang == 'en') {
  alert('Англійський текст');
}
if (lang == 'de') {
  alert('Німецький текст');
}
```

В даному коді спрацює тільки один з if, так як змінна lang може мати тільки одне зі значень.

Однак це рішення теж не дуже зручно. Уявіть, що у вас буде не три мови, а 10 – доведеться написати 10 конструкцій if.

Для таких випадків існує конструкція switch-case.

Конструкція switch-case

Дана конструкція являє собою альтернативу if-else, її рекомендується використовувати у разі множинного вибору (наприклад, 10 різних мов).

Вивчіть її синтаксис:

```
switch (змінна) {
case 'значення1':
        тут код, який виконується в разі, якщо змінна має значення1;
break;
case 'значення2':
    тут код, який виконується в разі, якщо змінна має значення2;
break;
case 'значение3':
    тут код, який виконується у випадку, якщо змінна має значеніе3;
break;
default:
    цей код виконається в разі, якщо змінна не збіглася ні з одним значенням;
break;
}
```

Вирішимо завдання з трьома мовами за допомогою даної конструкції:

```
switch (lang) {
  case 'ua':
        alert('Український текст');
  break;
  case 'en':
        alert('Англійський текст');
  break;
  case 'de':
```

```
        alert('Німецький текст');
    break;
    default:
        alert('Дана мова ні підтримується');
    break;
}
```

РОБОТА З ЦИКЛАМИ FOR І WHILE

Цикли використовуються для того, щоб деяку ділянку коду виконати кілька разів поспіль.

Цикл while

Цикл while - це конструкція, яка дозволяє виконувати певний код, допоки вказаний умовний вираз буде істинним.

Ось синтаксис цього циклу:

while (умова) {// код, який потрібно виконувати}

Цикл while перевірятиме умову на кожній ітерації. Якщо умова буде істинною, то код всередині тіла циклу виконуватиметься знову. Якщо умова стає хибною, то виконання циклу закінчується.

Наприклад, можна використовувати цикл while для виведення чисел від 1 до 5:

var i = 1; // лічильник циклу while (i <= 5) {

console.log(i); i++; // збільшуємо лічильник на 1}

В цьому прикладі на кожній ітерації циклу змінна i збільшується на 1, і виводиться на консоль значення цієї змінної. Цикл виконається 5 разів, поки змінна i не стане більше за 5. Також, важливо пам'ятати, що якщо умова вказана невірно, то цикл взагалі не виконається. Тому важливо

перевіряти умову на правильність перед запуском циклу, щоб уникнути непотрібних помилок.

Цикл for

Цикл for – це інший спосіб зробити те саме, що і з циклом while. Синтаксис циклу for складається з трьох частин: початкових команд, умови закінчення циклу та команд після проходу циклу.

1. Початкові команди виконуються тільки один раз на початку циклу і зазвичай містять початкові значення лічильників. Наприклад, можна почати з лічильника i = 0.
2. Умова закінчення циклу визначає, коли цикл повинен припинити свою роботу. Цикл буде виконуватися, поки ця умова є істинною. Наприклад, можна зупинити цикл, коли лічильник досягне значення 10, наприклад, якщо i < 10.
3. Команди після проходу циклу виконуються кожен раз після того, як виконується тіло циклу. Зазвичай тут збільшують лічильник на одиницю, наприклад, за допомогою оператора i++.

Наприклад, для того, щоб вивести числа від 0 до 9 за допомогою циклу for, можна використовувати наступний код:

```
for (var i = 0; i < 10; i++) {
alert(i); // виводить 0, 1, 2, ..., 9}
```

Тут змінна i є лічильником циклу. Починаємо з i = 0, і цикл продовжується, доки i < 10. Після кожного проходу циклу лічильник і збільшується на одиницю за допомогою оператора i++.

Цикл без тіла

У циклі for можна не вказувати фігурні дужки, якщо під ним повинен бути виконаний лише один рядок. Наприклад, в циклі for можна вивести числа від 0 до 9 таким чином:

```
for (var i = 0; i < 10; i++) alert(i);
```

У цьому випадку цикл виконає тільки один рядок під ним, а саме – виведе числа від 0 до 9.

Проте, якщо поставити точку з комою після умови циклу, цикл буде виконуватися, але нічого не робитиме. Наступний рядок коду буде виконаний після завершення циклу, а не в тілі циклу. Наприклад:

```
for (var i = 0; i < 10; i++); alert(i);
```

У цьому випадку цикл буде прокручуватись 10 разів, але не робитиме нічого. Після завершення циклу буде виведене число 9, оскільки після останньої ітерації змінна i дорівнюватиме 10, але відображатиметься останнє значення змінної до закінчення циклу, тобто 9.

Кілька команд в циклі for

Цикл for може виконувати декілька команд, якщо вони вказані через кому в круглих дужках:

```
for (var i = 0, j = 2; i < 10; i++, j++, i = i + j) {
//тіло циклу
}
```

Перед першим проходом циклу виконуються дві команди: var i = 0, j = 2 (зверніть увагу на те, що var тут пишеться один раз). А після кожної ітерації циклу виконується цілих три команди: i++, j++, i = i + j.

Цикл for для масивів

Цикл for дозволяє послідовно обійти всі елементи масиву. Для цього ми створюємо змінну i, присвоюємо їй початкове значення 0 та збільшуємо на 1 після кожної ітерації, поки не досягнемо довжини масиву.

Наприклад, у нас є масив arr = [1, 2, 3, 4, 5]. Щоб вивести всі його елементи, ми використовуємо цикл for:

```
var arr = [1, 2, 3, 4, 5];
for (var i = 0; i < arr.length; i++) {
alert(arr[i]); //виведе 1, 2, 3, 4, 5
}
```

Важливим моментом є те, що ми робимо перебір від нуля до довжини масиву, але оскільки індекс елементів масиву починається з 0, ми віднімаємо одиницю від довжини масиву, щоб не вийти за межі масиву. Таким чином, ми можемо перебрати всі елементи масиву та вивести їх на екран.

Також можна використовувати інший спосіб перебору масиву, де ми не віднімаємо одиницю від довжини масиву, але замість <= використовуємо < в умові циклу:

```
var arr = [1, 2, 3, 4, 5];
for (var i = 0; i <= arr.length-1; i++) {
```

```
alert(arr[i]); //виведе 1, 2, 3, 4, 5
}
```

Тут ми також перебираємо масив від нуля до довжини масиву мінус одиниця. Обидва способи еквівалентні та дають один і той самий результат.

Цикл for-in

Для перебору ключів об'єкта використовують цикл for-in. Наприклад, у нас є об'єкт з ключами та їх значеннями:

```
var obj = { Коля: 200, Вася: 300, Петя: 400 };
```

Щоб вивести всі ключі, можна використати цикл:

```
for (var key in obj) {
alert(key); //виведе 'Коля', 'Вася', 'Петя'
}
```

Тут ключі об'єкта будуть виведені по черзі, а змінна key - назва змінної, в яку будуть записуватися ключі об'єкта. В цьому циклі не потрібно вказувати умову закінчення - цикл перебере всі ключі, поки їх не залишиться.

Якщо потрібно вивести значення ключів, потрібно звернутися до об'єкта по ключу - obj[key]. Наприклад:

```
for (var key in obj) {
alert(obj[key]); //виведе 200, 300, 400
}
```

Тут у змінну key будуть записані ключі об'єкта по черзі, і за допомогою obj[key] можна вивести їх значення.

Інструкція **break**

Іноді потрібно зупинити виконання циклу раніше, ніж він перебере всі елементи масиву. Наприклад, якщо ми хочемо вивести всі елементи масиву до певного значення і зупинитися, коли досягли цього значення. Для цього ми використовуємо ключове слово "break".

Якщо програма дійде до цього оператора, то цикл буде завершений і програма вийде з нього. Наприклад, якщо ми маємо масив [1, 2, 3, 4, 5] і хочемо вивести всі елементи до числа 3, то ми можемо написати такий код:

```
var arr = [1, 2, 3, 4, 5];
for (var i = 0; i < arr.length; i++) {
if (arr[i] === 3) {
        break; //вихід з циклу
} else {
        alert(arr[i]);
}
}
```

У цьому прикладі ми створюємо масив arr і проходимо по всіх його елементах за допомогою циклу for. У тілі циклу перевіряється, чи дорівнює поточний елемент масиву 3. Якщо так, то ми викликаємо break і цикл завершується, якщо ні, то ми виводимо цей елемент за допомогою alert().

Інструкция **continue**

У програмуванні існують цикли, які дозволяють виконувати певні дії декілька разів. Але іноді можуть виникнути ситуації, коли потрібно зупинити виконання циклу до того, як він закінчиться.

Наприклад, у нас є масив з числами і ми хочемо вивести всі числа, крім числа 3. Для цього ми можемо скористатися інструкцією continue. Коли програма доходить до числа 3, цикл пропускає його і переходить до наступної ітерації.

Ось як це можна зробити у циклі for з масивом чисел:

```
var arr = [1, 2, 3, 4, 5];
for (var i = 0; i < arr.length; i++) {
if (arr[i] === 3) {
        continue; // пропускаем элемент с числом 3
} else {
        console.log(arr[i]);
}
}
```

Цикл for проходить по кожному елементу масиву arr, і якщо зустрічає число 3, то він переходить до наступної ітерації, не виводячи число 3. У всіх інших випадках він виводить значення елементу масиву.