**­ Лекція 2. Взаємодія користувача з веб-сторінкою. Основні оператори та типи перетворень в JavaScript.**

1. Взаємодія користувача з веб-сторінкою. Умовні оператори *if, else.*
2. Логічні оператори.
3. Перетворення типів для примітивів.

**1.Взаємодія з користувачем: alert, prompt, confirm**

У цьому питанні ми розглянемо базові UI операції: *alert, prompt і confirm*, які дозволяють працювати з даними, отриманими від користувача.

***alert***

синтаксис:

alert(повідомлення);

*alert* виводить на екран вікно з повідомленням і призупиняє виконання скрипта, поки користувач не натисне «ОК».

alert( " Привіт " );

Вікно повідомлення, яке виводиться, є модальним вікном. Слово «модальне» означає, що користувач не може взаємодіяти зі сторінкою, натискати інші кнопки, тощо, поки не розбереться з вікном. В даному випадку - поки не натисне на «OK».

***prompt***

Функція *prompt* приймає два аргументи:

result = prompt(title, default);

Вона виводить модальне вікно з заголовком *title*, полем для введення тексту, заповненим рядком за замовчуванням *default* і кнопками OK / CANCEL.

Користувач повинен або щось ввести і натиснути OK, або скасувати введення кліком на CANCEL або натисканням Esc на клавіатурі.

Виклик prompt повертає те, що ввів користувач - рядок або спеціальне значення *null*, якщо введення скасоване.

Як і у випадку з *alert*, вікно *prompt* модальне.

var years = prompt('Скільки вам років?', 100);

alert('Вам ' + years + ' років!')

**Завжди вказуйте *default***

Другий параметр може бути відсутнім. Однак при цьому IE вставить в діалог значення за замовчуванням "*undefined*".

Запустіть цей код в IE, щоб зрозуміти про що мова:

var test = prompt("Тест");

Тому рекомендується завжди вказувати другий аргумент:

var test = prompt("Тест", ''); // <-- так краще

***confirm***

синтаксис:

result = confirm(question);

*confirm* виводить вікно з питанням question з двома кнопками: OK і CANCEL.

Результатом буде *true* при натисканні OK і *false* - при CANCEL (Esc).

Наприклад:

var isAdmin = confirm("Ви - адміністратор?");

alert( isAdmin );

*Особливості вбудованих функцій*

Конкретне місце, де виводиться модальне вікно з питанням - зазвичай це центр браузера, і зовнішній вигляд вікна обирає браузер. Розробник не може на це впливати.

З одного боку - це недолік, так як не можна вивести вікно в своєму, особливо красивому, дизайні.

З іншого боку, перевага цих функцій в порівнянні з іншими, більш складними методами взаємодії, які ми вивчимо в подальшому - як раз в тому, що вони дуже прості.

Це найпростіший спосіб вивести повідомлення або отримати інформацію від відвідувача. Тому їх використовують в тих випадках, коли простота важлива, а всякі «красивості» особливої ​​ролі не грають.

**Підсумок**

• *alert* виводить повідомлення.

• *prompt* виводить повідомлення і чекає, поки користувач введе текст, а потім повертає введене значення або null, якщо введення скасоване (CANCEL / Esc).

• *confirm* виводить повідомлення і чекає, поки користувач натисне «OK» або «CANCEL» і повертає true / false.

**Умовні оператори: *if*, *'?'***

Іноді, в залежності від умови, потрібно виконати різні дії. Для цього використовується оператор *if.*

Наприклад:

var year = prompt(' 'В якому році з'явилася специфікація ECMA-262 5.1?', '');

if (year != 2011) alert( 'А ось і не вірно' );

**Оператор *if***

Оператор *if* ( «якщо») отримує умову, в прикладі вище це year! = 2011. Він обчислює її, і якщо результат - *true*, то виконує команду.

Якщо потрібно виконати більше однієї команди - вони оформляються блоком коду в фігурних дужках:

if (year != 2011) {

alert( 'А ось..' );

alert( '..і не вірно!' );

}

Рекомендується використовувати фігурні дужки завжди, навіть коли команда одна. Це покращує читабельність коду.

Оператор *if* (...) обчислює і перетворює вираз в дужках до логічного типу.

У логічному контексті:

Число 0, порожній рядок "", *null* і *undefined*, а також *NaN* є *false*, Решта значення - *true*.

Наприклад, така умова ніколи не виконається:

if (0) { // 0 приведеться до false

...

}

... А таке - виконається завжди:

if (1) { // 1 приведеться до true

...

}

Можна і просто передати вже готове логічне значення, наприклад, заздалегідь розраховане в змінній:

var cond = (year != 2011); // true/false

if (cond) {

...

}

**Неправильна умова, *else***

Необов'язковий блок *else* ( «інакше») виконується, якщо умова невірна:

var year = prompt('Введіть рік появи стандарту ECMA-262 5.1', '');

if (year == 2011) {

alert( 'Так, ви знавець!' );

} else {

alert( 'А ось і неправильно!' ); // будь-яке значення, крім 2011

}

**Кілька умов, *else* *if***

Буває потрібно перевірити кілька варіантів умови. Для цього використовується блок *else* *if* .... Наприклад:

*var year = prompt('В якому році з’явилась специфікація ECMA-262 5.1?', '');*

*if (year < 2011) {*

*alert( Це дуже рано..' );*

*} else if (year > 2011) {*

*alert( 'Це запізно..' );*

*} else {*

*alert( 'Так, точно в цьому році!' );*

*}*

В наведеному вище прикладі JavaScript спочатку перевірить першу умову, якщо вона помилкова - перейде до другої - і так далі, до останнього *else*.

**Оператор знак питання "*?*"**

Іноді потрібно в залежності від умови привласнити змінну. Наприклад:

var access;

var age = prompt('Скільки вам років?', '');

if (age > 14) {

access = true;

} else {

access = false;

}

alert(access);

Оператор знак *'?'* дозволяє робити це швидше і простіше. Він складається з трьох частин: умова? значення1: значення2

Перевіряється умова, потім якщо воно вірно - повертається значення1, якщо невірно - значення2, наприклад:

access = (age > 14) ? true : false;

Оператор '?' виконується пізніше більшості інших, зокрема - пізніше порівнянь, тому дужки можна не ставити:

access = age > 14 ? true : false;

... Але коли дужки є - код краще читається. Так що рекомендується їх писати.

*На замітку:*

В даному випадку можна було б обійтися і без оператора '?', Тому що порівняння само по собі вже повертає true / false:

access = age > 14;

«Тернарний оператор»

Знак питання - єдиний оператор, у якого є аж три аргументи, в той час як у звичайних операторів їх один-два. Тому його називають «тернарний оператор».

Кілька операторів *"?"*

Послідовність операторів *'?'* дозволяє повернути значення в залежності не від однієї умови, а від кількох.

Наприклад:

var age = prompt('вік?', 18);

var message = (age < 3) ? ' Привіт, малюк!' :

(age < 18) ? 'Привіт!' :

(age < 100) ? 'Доброго дня!' :

'Який незвичайний вік!';

alert( message );

Спочатку може складно зрозуміти, що відбувається. Однак, уважно придивившись, ми помічаємо, що це звичайна послідовна перевірка!

Знак питання перевіряє спочатку age <3, якщо вірно - повертає 'Привіт, малюк!', Якщо ні - йде за двокрапкою і перевіряє age <18. Якщо це вірно - повертає 'Привіт!', Інакше перевірка age <100 і 'Привіт! '... І нарешті, якщо нічого з цього не вірно, то' Який незвичайний вік! '.

Те ж саме через if..else:

if (age < 3) {

message = ' Привіт, малюк!';

} else if (age < 18) {

message = 'Привіт!';

} else if (age < 100) {

message = 'Доброго дня!';

} else {

message = 'Який незвичайний вік!';

Нетрадиційне використання *"?"*

Іноді оператор знак '*?*' використовують як заміну *if*:

 var company = prompt('Яка компанія створила JavaScript?', '');

(company == 'Netscape') ?

alert('Так, вірно') : alert('Невірно');

Працює це так: в залежності від умови, буде виконана або перша, або друга частина після '*?*'. Результат виконання не записується в змінну, так що він пропаде (втім, *alert* нічого не повертає).

Рекомендується не використовувати знак питання таким чином. Незважаючи на те, що на вигляд такий запис коротший *if*, він є значно менш читабельним.

Ось, для порівняння, те ж саме з *if*:

var company = prompt('Яка компанія створила JavaScript?', '');

if (company == 'Netscape') {

alert('Так, вірно');

} else {

alert('Невірно');

}

При читанні коду очі слідують вертикально за текстом і конструкції, що займають кілька рядків, зі зрозумілою вкладеністю, сприймаються набагато легше. Можливо, ви і самі відчуєте, пробігши очима, що синтаксис з *if* більш простий і очевидний ніж з оператором '?'.

Сенс оператора '*?*' - повернути те чи інше значення, в залежності від умови. Будь ласка, використовуйте його за призначенням, а для виконання різних гілок коду є *if*.

**2.Логічні оператори**

Для операцій над логічними значеннями в JavaScript є || (АБО), && (І) і ! (НI).

Хоч вони і називаються «логічними», але в JavaScript можуть застосовуватися до значень будь-якого типу і повертають також значення будь-якого типу.

**|| (АБО)**

Оператор АБО виглядає як подвійний символ вертикальної риски:

result = a || b;

Логічне АБО в класичному програмуванні працює наступним чином: "якщо хоча б один з аргументів *true*, то повертає true, інакше - *false*". В JavaScript, як ми побачимо далі, це не зовсім так, але для початку розглянемо тільки логічні значення.

Виходить наступна «таблиця результатів»:

alert( true || true ); // true

alert( false || true ); // true

alert( true || false ); // true

alert( false || false ); // false

Якщо значення не логічного типу - то воно до нього наводиться в цілях обчислень. Наприклад, число 1 буде сприйнято як true, а 0 - як false:

if (1 || 0) { // сработает как if( true || false )

alert( 'вірно' );

}

Зазвичай оператор АБО використовується в *if,* щоб перевірити, чи виконується хоча б одна з умов, наприклад:

var hour = 9;

if (hour < 10 || hour > 18) {

alert( 'Офіс до 10 або після 18 зачинено ');

}

Можна передати і більше умов:

var hour = 12, isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {

alert( 'Офіс до 10 або після 18 або у вихідний закритий');

}

**Короткий цикл обчислень**

JavaScript обчислює кілька АБО зліва направо. При цьому, щоб економити ресурси, використовується так званий «короткий цикл обчислення».

Припустимо, обчислюються кілька АБО поспіль: a || b || c || .... Якщо перший аргумент - true, то результат свідомо буде true (хоча б одне зі значень - true), і інші значення ігноруються.

Це особливо помітно, коли вираз, переданий в якості другого аргументу, має сторонній ефект - наприклад, привласнює змінну.

При запуску прикладу нижче привласнення *x* не відбудеться:

var x;

true || (x = 1);

alert(x); // undefined, x не присвоєно

... А в прикладі нижче перший аргумент - *false*, так що АБО спробує обчислити другий, запустивши тим самим присвоювання:

var x;

false || (x = 1);

alert(x); // 1

**значення АБО**

Отже, як ми бачимо, оператор АБО обчислює рівно стільки значень, скільки необхідно - до першого true. При цьому оператор АБО повертає то значення, на якому зупинилися обчислення. Причому, воно не перетворене до логічного типу.

Наприклад:

alert( 1 || 0 ); // 1

alert( true || 'неважливо що' ); // true

alert( null || 1 ); // 1

alert( undefined || 0 ); // 0

Це використовують, зокрема, щоб вибрати перше «справжнє» значення зі списку:

var undef; // змінна не привласнена, тобто рівна undefined

var zero = 0;

var emptyStr = "";

var msg = "Привіт!";

var result = undef || zero || emptyStr || msg || 0;

alert( result ); // виведе "Привіт!" – перше значення, яке є true

Якщо всі значення «помилкові», то || поверне останнє з них:

alert( undefined || '' || false || 0 ); // 0

Отже, оператор || обчислює операнди зліва направо до першого «справжнього» і повертає його, а якщо усі помилкові - то останнє значення. Інакше можна сказати, що "|| зупиняється на правді".

**&& (І)**

Оператор І пишеться як два амперсанда &&:

result = a && b;

У класичному програмуванні І повертає true, якщо обидва аргументи істинні, а інакше - false:

alert( true && true ); // true

alert( false && true ); // false

alert( true && false ); // false

alert( false && false ); // false

Приклад з if:

var hour = 12, minute = 30;

if (hour == 12 && minute == 30) {

alert( 'Час 12:30' );

}

Як і в АБО, в І допустимі будь-які значення:

if (1 && 0) { // вычислится как true && false

alert( 'не спрацює, тому що умова помилкова');

}

До І застосуємо той же принцип «короткого циклу обчислень», але трохи по-іншому, ніж до АБО. Якщо лівий аргумент - false, оператор І повертає його і закінчує обчислення. Інакше - обчислює і повертає правий аргумент.

Наприклад:

// Перший аргумент - true,

// Тому повертається другий аргумент

alert( 1 && 0 ); // 0

alert( 1 && 5 ); // 5

// Перший аргумент - false,

// Він повертається, а другий аргумент ігнорується

alert( null && 5 ); // null

alert( 0 && "не важливо" ); // 0

Можна передати і кілька значень поспіль, при цьому повернеться перше «хибне» (на якому зупинилися обчислення), а якщо його немає - то останнє:

alert( 1 && 2 && null && 3 ); // null

alert( 1 && 2 && 3 ); // 3

Отже, оператор && обчислює операнди зліва направо до першого «помилкового» і повертає його, а якщо усі правдиві - то останнє значення. Інакше можна сказати, що "&& зупиняється на брехні".

**Пріоритет у && більше, ніж у ||**

Пріоритет оператора І && більше, ніж АБО ||, так що він виконується раніше. Тому в наступному коді спочатку буде обчислено праве І: 1 && 0 = 0, а вже потім - АБО.

alert( 5 || 1 && 0 ); // 5

Не використовуйте && замість *if* Оператор && в простих випадках можна використовувати замість *if*, наприклад:

var x = 1;

(x > 0) && alert( 'Більше' );

Дія в правій частині && виконається тільки в тому випадку, якщо до нього дійдуть обчислення. Тобто, alert спрацює, якщо в лівій частині буде true.

Вийшов аналог:

var x = 1;

if (x > 0) {

alert( 'Більше' );

}

Однак, як правило, варіант з *if* краще читається і сприймається. Він більш очевидний, тому краще використовувати його. Це, втім, стосується й інших неочевидним застосуванням можливостей мови.

**! (НІ)**

Оператор НІ - найпростіший. Він отримує один аргумент. Синтаксис:

var result = !value;

Дії!:

1. Спочатку наводить аргумент до логічного типу *true / false*.

2. Потім повертає протилежне значення.

Наприклад:

alert( !true ); // false

alert( !0 ); // true

Зокрема, подвійне НІ використовують для перетворення значень до логічного типу:

alert( !!"строка" ); // true

alert( !!null ); // false

**3.** **Перетворення типів для примітивів**

Система перетворення типів в JavaScript дуже проста, але відрізняється від інших мов. Тому вона часто служить «каменем спотикання» для програмістів, що пишуть на інших мовах.

Всього є три перетворення:

1. Строкове перетворення.
2. Чисельне перетворення.
3. Перетворення до логічного значення.

Ця глава описує перетворення тільки примітивних значень, об'єкти розбираються далі.

**Строкове перетворення**

Строкове перетворення відбувається, коли потрібне представлення чого-небудь у вигляді рядка. Наприклад, його виконує функція *alert*.

var a = true;

alert( a ); // "true"

Можна також здійснити перетворення явним викликом *String (val):*

alert( String(null) === "null" ); // true

Як видно з прикладів вище, перетворення відбувається найбільш очевидним способом, «як є»: *false* стає "*false*", *null* - "*null*", *undefined* - "*undefined*" і т.п.

Також для явного перетворення застосовується оператор "+", у якого один з аргументів рядок. У цьому випадку він приводить до рядка і інший аргумент, наприклад:

alert( true + "test" ); // "truetest"

alert( "123" + undefined ); // "123undefined"

**Чисельне перетворення**

Чисельне перетворення відбувається в математичних функціях і виразах, а також при порівнянні даних різних типів (крім порівнянь ===,! ==).

Для перетворення до числа в явному вигляді можна викликати *Number* (*val*), або, що коротше, поставити перед виразом унарний плюс "+":

var a = +"123"; // 123

var a = Number("123"); // 123, той же ефект

| **Значення** | **Перетворюється у...** |
| --- | --- |
| undefined | NaN |
| null | 0 |
| true / false | 1 / 0 |
| Строка | Пробільні символи по краях відкидаються.  Далі, якщо залишається порожній рядок, то 0, інакше з непорожнього рядка "зчитується" число, при помилці результат NaN. |
|  |  |

Наприклад:

// після відкидання пробільних символів залишиться "123"

alert( +" \n 123 \n \n" ); // 123

Ще приклади:

*Логічні значення:*

alert( +true ); // 1

alert( +false ); // 0

*Порівняння різних типів – має на увазі чисельне перетворення:*

alert( "\n0 " == 0 ); // true

При цьому рядок "\ n0" перетвориться до числа, як зазначено вище: початкові і кінцеві пробіли обрізаються, виходить рядок "0", який дорівнює 0.

*З логічними значеннями:*

 alert( "\n" == false );

alert( "1" == true );

Тут порівняння "==" знову приводить обидві частини до числа. У першому рядку ліворуч і праворуч виходить 0, у другому 1.

**Спеціальні значення**

Подивимося на поведінку спеціальних значень більш уважно. Інтуїтивно, значення null / undefined асоціюються з нулем, але при перетвореннях поводяться інакше.

Спеціальні значення перетворюються до числа так:

|  |  |
| --- | --- |
| **Значення** | **Перетворюється в...** |
| undefined | NaN |
| null | 0 |

Це перетворення здійснюється при арифметичних операціях і порівняннях >> = <<=, але не під час перевірки рівності ==. Алгоритм перевірки рівності для цих значень в специфікації прописаний окремо. У ньому вважається, що null і undefined рівні "==" між собою, але ці значення не рівні ніякому іншому значенню. Це веде до кумедних наслідків.

Наприклад, null не підкоряється законам математики - він «більше або дорівнює нулю»: null> = 0, але не більше і не дорівнює:

 alert( null >= 0 ); // true, оскільки. null перетвориться у 0

alert( null > 0 ); // false (не більше), оскільки null перетвориться у 0

alert( null == 0 ); // false (і не рівен!), оскільки == розглядає null особливо.

Значення undefined взагалі «непорівняне»:

alert( undefined > 0 ); // false, оскільки undefined -> NaN

alert( undefined == 0 ); // false, оскільки це undefined (без перетворення)

alert( undefined < 0 ); // false, оскільки undefined -> NaN

Для більш очевидної роботи коду і щоб уникнути помилок краще не давати спеціальним значенням брати участь в порівняннях >> = <<=.

Використовуйте в таких випадках змінні-числа або перетворюйте до числа явно.

**Логічне перетворення**

Перетворення до true / false відбувається в логічному контексті, такому як *if* (*value*), і при застосуванні логічних операторів. Всі значення, які інтуїтивно «порожні», стають *false*. Їх кілька: 0, порожній рядок, *null, undefined* і *NaN.*

Усе інше, в тому числі і будь-які об'єкти - true.

Повна таблиця перетворень:

|  |  |
| --- | --- |
| **Значення** | **Перетворюється у...** |
| undefined, null | false |
| Числа | Усі true, крім 0, NaN -- false. |
| Строки | Усі true, крім пустої строки "" -- false |
| Об’єкти | Завжди true |

Для явного перетворення використовується подвійне логічне заперечення !! *value* або виклик *Boolean (value)*.

Зверніть увагу: рядок "0" стає *true*

На відміну від багатьох мов програмування (наприклад PHP), "0" в *JavaScript* є *true*, як і рядок з прогалин:

alert( !!"0" ); // true

alert( !!" " ); // будь-які непусті рядки, навіть з пропусками - true!

Логічне перетворення цікаво тим, як воно поєднується з чисельним перетворенням.

Два значення можуть бути рівні, але одне з них в логічному контексті true, інше - false.

Наприклад, рівність в наступному прикладі вірна, так як відбувається чисельне перетворення:

alert( 0 == "\n0\n" ); // true

... А в логічному контексті ліва частина (0) дасть false, права ( "\ n0 \ n") - true, так як будь-який не порожній рядок в логічному контексті дорівнює true:

if ("\n0\n") {

alert( "true, зовсім не як 0!" );

}

З точки зору перетворення типів в JavaScript це абсолютно нормально. При порівнянні за допомогою «==» - чисельне перетворення, а в *if* - логічне, тільки і всього.

**Висновок**

В JavaScript є три перетворення:

Строкове: *String (value)* - в строковому контексті або при додаванні з рядком. Працює очевидним чином.

Чисельне: *Number (value)* - в чисельному контексті, включаючи унарний плюс + *value*. Відбувається при порівнянні різних типів, крім строгої рівності.

Логічне: *Boolean (value)* - в логічному контексті, можна також зробити подвійним НІ: !! value.

Особливим випадком є ​​перевірка рівності з *null* і *undefined*. Вони дорівнюють один одному, але не рівні чому б то не було ще, цей випадок прописаний особливо в специфікації.