Логічні оператори

В JavaScript існує чотири логічні оператори: `||` (АБО), `&&` (І), `!` (НЕ), `??` (оператор null-об’єднання). В цьому розділі ми розглянемо перші три оператори, а оператор `??` — в наступному розділі.

Хоча вони називаються “логічними”, вони можуть бути застосовані до значень будь-якого типу, не тільки булевих. Їх результати також можуть бути будь-якого типу.

Подивимось більш детально.

|| (АБО)

Оператор “АБО” представлений двома символами вертикальної лінії:

```
result = a || b;
```

У класичному програмуванні логічний оператор АБО призначений для маніпулювання лише булевими значеннями. Якщо будь-який з його аргументів означає `true`, повертається `true`, інакше повертається `false`.

У JavaScript цей оператор складніший і потужніший. Але спочатку подивимося, що відбувається з булевими значеннями.

Є чотири можливі логічні комбінації:

```
alert( true || true );   // true
alert( false || true );  // true
alert( true || false );  // true
alert( false || false ); // false
```

Як бачимо, результат завжди `true`, за винятком випадку, коли обидва операнди `false`.

Якщо операнд не є булевим, він перетворюється на булевий для обчислення.

Наприклад, число `1` розглядається як `true`, число `0` — як `false`:

```
if (1 || 0) { // працює так само, як ( true || false )
  alert( 'правдиво!' );
}
```

У більшості випадків АБО `||` використовується в інструкціях `if`, щоб перевірити, чи є *будь-яка* із заданих умов `true`.

Наприклад:

```
let hour = 9;

if (hour < 10 || hour > 18) {
  alert( 'Офіс зачинений.' );
}
```

Ми можемо передавати більше умов:

```
let hour = 12;
let isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {
  alert( 'Офіс зачинений.' ); // це вихідні
}
```

АБО "||" знаходить перше правдиве значення

Описана вище логіка дещо класична. Тепер введімо “додаткові” особливості JavaScript.

Розширений алгоритм працює наступним чином.

Дано кілька значень, розділених оператором АБО:

```
result = value1 || value2 || value3;
```

Оператор АБО `||` робить наступне:

- Обчислює операнди зліва направо.
- Перетворює значення кожного операнда на булеве. Якщо результат `true`, зупиняється і повертає початкове значення цього операнда.
- Якщо всі операнди були обчисленні (тобто усі були `false`), повертає останній операнд.

Значення повертається у первісному вигляді без конвертації.

Іншими словами, ланцюжок з АБО `||` повертає перше правдиве значення або останнє, якщо правдивого значення не знайдено.

Наприклад:

```
alert( 1 || 0 ); // 1 (1 є правдивим)

alert( null || 1 ); // 1 (1 є першим правдивим значенням)
alert( null || 0 || 1 ); // 1 (перше правдиве значення)

alert( undefined || null || 0 ); // 0 (усі хибні, повертається останнє значення)
```

Це призводить до цікавого використання, у порівнянні з “чистим, класичним, виключно-булевим АБО”.

1. **Отримання першого істинного значення зі списку змінних або виразів.**

   Наприклад, маємо змінні `firstName`, `lastName` та `nickName`, усі необов’язкові (тобто вони можуть бути невизначеними або мати хибні значення).

   Використаємо АБО `||`, щоб вибрати ту змінну, яка має дані, і виведемо її (або рядок `"Анонім"`, якщо жодна змінна не має даних):

   ```
   let firstName = "";
   let lastName = "";
   let nickName = "СуперКодер";

   alert( firstName || lastName || nickName || "Анонім"); // СуперКодер
   ```

   Якщо всі змінні мали б порожні рядки, тоді показалося слово `"Анонім"`.

2. **Обчислення короткого замикання.**

   Іншою особливістю оператора АБО `||` є так зване “обчислення короткого замикання”.

   Це означає, що оператор `||` опрацьовує аргументи доти, доки не досягається перше правдиве значення, після чого це значення негайно повертається, без подальшого опрацювання решти аргументів.

   Важливість такої особливості стає очевидною, якщо операнд є не просто змінною, а виразом із побічним ефектом, як-от присвоєння змінної або виклик функції.

   У наведеному нижче прикладі виведеться лише друге повідомлення:

```
true || alert("не виведеться");
false || alert("виведеться");
```

В першому рядку оператор АБО `||` зупиняє виконання відразу після того, як “побачить” що лівий вираз є `true`, тож `alert` не виконається.

Деколи таку конструкцію використовують, щоб виконувати команди лише при хибності умови ліворуч від оператора.

&& (І)

Оператор І представлений двома амперсандами `&&`:

```
result = a && b;
```

У класичному програмуванні І повертає `true`, якщо обидва оператори є правдивими, і `false` в іншому випадку:

```
alert( true && true );   // true
alert( false && true );  // false
alert( true && false );  // false
alert( false && false ); // false
```

Приклад з `if`:

```
let hour = 12;
let minute = 30;

if (hour == 12 && minute == 30) {
  alert( 'Час: 12:30' );
}
```

Так само, як з АБО, будь-яке значення дозволено як операнд І:

```
if (1 && 0) { // обчислюється як true && false
  alert( "не буде працювати, тому що результат хибний" );
}
```

І “&&” шукає перше хибне значення

Дано декілька значень, об’єднаних кількома І:

```
result = value1 && value2 && value3;
```

Оператор І `&&` робить наступне:

- Обчислює операнди зліва направо.
- Перетворює кожен операнд на булевий. Якщо результат `false`, зупиняється і повертає оригінальне значення того операнда.
- Якщо всі операнди були обчисленні (тобто усі були правдиві), повертає останній операнд.

Іншими словами, І повертає перше хибне значення, або останнє значення, якщо жодного хибного не було знайдено.

Правила, наведені вище, подібні до правил АБО. Різниця полягає в тому, що І повертає перше *хибне* значення, тоді як АБО повертає перше *правдиве*.

Приклади:

```
// якщо перший операнд правдивий,
// І повертає другий операнд:
alert( 1 && 0 ); // 0
alert( 1 && 5 ); // 5

// якщо перший операнд хибний,
// І повертає саме його. Другий операнд ігнорується
alert( null && 5 ); // null
alert( 0 && "неважливо" ); // 0
```

Ми також можемо передавати декілька значень поспіль. Подивіться, як повертається перше хибне:

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

Коли всі значення є правдивими, повертається останнє значення:

```
alert( 1 && 2 && 3 ); // 3, останнє
```

Пріоритет І `&&` вище за АБО `||`

Оператор І `&&` має вищий пріоритет за АБО `||`.

Отже, код `a && b || c && d` по суті є таким самим, як код з виразами `&&` у дужках: `(a && b) || (c && d)`.

Не міняйте `if` на `||` чи `&&`

Деколи оператор І `&&` використовують як "скорочений варіант `if`".

Наприклад:

```
let x = 1;

(x > 0) && alert( 'Більше за нуль!' );
```

Дія у правій частині `&&` буде виконуватися, тільки якщо обчислення дійде до неї. Тобто тільки якщо `(x > 0)` є істинним.

Тому, власне, ми маємо аналог для:

```
let x = 1;

if (x > 0) alert( 'Більше за нуль!' );
```

Хоча варіант з `&&` видається коротшим, конструкція з `if` є більш очевидною і зазвичай більш читабельною. Тому ми рекомендуємо використовувати кожну конструкцію за своїм призначенням: використовуємо `if`, якщо нам потрібна інструкція `if`, і використовуємо `&&`, якщо нам потрібен оператор І.

! (НЕ)

Булевий оператор НЕ представлений знаком оклику `!`.

Синтаксис дуже простий:

```
result = !value;
```

Оператор приймає один аргумент і виконує наступне:

1. Перетворює операнд на булевий тип: `true/false`.
2. Повертає зворотне значення.

Наприклад:

```
alert( !true ); // false
alert( !0 ); // true
```

Подвійний НЕ `!!` іноді використовується для перетворення значення на булевий тип:

```
alert( !!"не пустий рядок" ); // true
alert( !!null ); // false
```

Тобто, перший НЕ перетворює значення на булеве і повертає зворотне, а другий НЕ інвертує його знову. Зрештою ми маємо просте перетворення значень на булевий тип.

Є трохи довший спосіб зробити те ж саме – вбудована функція `Boolean`:

```
alert( Boolean("не пустий рядок") ); // true
alert( Boolean(null) ); // false
```

Пріоритет НЕ `!` є найвищим серед усіх логічних операторів, тому він завжди виконується першим, перед `&&` або `||`.

Завдання

Challenge 1 **Який результат АБО?**

Що виведе код нижче?

```
alert( null || 2 || undefined );
```

Challenge 2 **Який результат alerts, об'єднаних АБО?**

Що виведе код нижче?

```
alert( alert(1) || 2 || alert(3) );
```

Challenge 3 **Який результат І?**

Що виведе код нижче?

```
alert( 1 && null && 2 );
```

Challenge 4 **Який результат alerts, об'єднаних за допомогою І?**

Що виведе код нижче?

```
```

```
alert( alert(1) && alert(2) );
```

Challenge 5 **Результат АБО І АБО**

Який буде результат?

```
alert( null || 2 && 3 || 4 );
```

Challenge 6 **Перевірте діапазон**

Напишіть умову `if`, щоб перевірити, чи `age` знаходиться між `14` та `90` включно.

“Включно” означає, що `age` може досягати країв `14` або `90`.

Challenge 7 **Перевірте значення поза діапазоном**

Напишіть умову `if`, щоб перевірити, чи значення `age` НЕ знаходиться між `14` та `90` включно.

Створіть два варіанти: перший з оператором НЕ `!`, другий — без нього.

Challenge 8 **Питання про "if"**

Які з цих `alert` буде виконано?

Якими будуть результати виразів у `if(...)`?

```
if (-1 || 0) alert( 'перший' );
if (-1 && 0) alert( 'другий' );
if (null || -1 && 1) alert( 'третій' );
```

Challenge 9 **Перевірте логін**

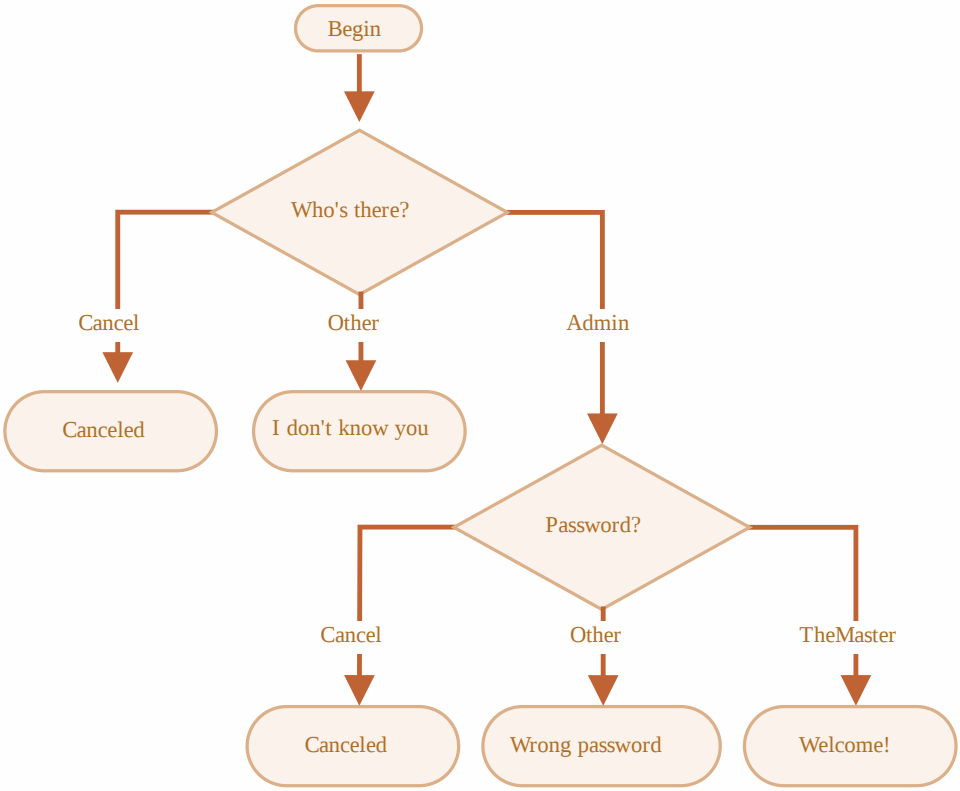
Напишіть код, який запитує логін за допомогою `prompt`.

Якщо відвідувач вводить `"Admin"`, тоді запропонуйте за допомогою `prompt` ввести пароль. Якщо введено порожній рядок або натиснуто `Esc` – показати “Скасовано”. Якщо введено інший рядок – тоді покажіть “Я вас не знаю”.

Пароль перевіряється наступним чином:

- Якщо він дорівнює “Господар”, тоді покажіть “Ласкаво просимо!”,
- Інший рядок – покажіть “Неправильний пароль”,
- Для порожнього рядка, або якщо введення було скасовано, покажіть “Скасовано”.

Схема:



Будь ласка, використовуйте вкладені `if` блоки. Потурбуйтесь про загальну читабельність коду.

Підказка: передача порожнього вводу до запиту повертає порожній рядок `''`. Натискання `ESC` протягом запиту повертає `null`.

Оператор об’єднання з null '??'

Нещодавнє доповнення

Це нещодавнє доповнення до мови. У старих браузерах може бути потрібен поліфіл.

Оператор об’єднання з null записується як два знаки питання `??`.

Оскільки `null` і `undefined` сприймаються однаково, ми введемо спеціальну домовленність. В цій статті ми будемо вважати, що значення виразу “визначене”, якщо воно відрізняється від `null` та `undefined`.

Результатом `a ?? b` буде:

- `a`, якщо `a` визначене,
- `b`, якщо `a` не визначене.

Інакше кажучи, `??` повертає перший аргумент, якщо він не `null/undefined`. Інакше, другий.

Оператор об’єднання з null не є абсолютно новим. Це просто хороший синтаксис, щоб отримати перше “визначене” значення з двох.

Ми можемо переписати вираз `result = a ?? b`, використовуючи оператори, які ми вже знаємо:

```
result = (a !== null && a !== undefined) ? a : b;
```

Тепер повинно бути абсолютно зрозуміло, що робить `??`. Подивімось, де це допомагає.

Наприклад, тут ми показуємо значення у змінній `user`, якщо її значення не `null/undefined`, інакше – показуємо `Анонімний`:

Ось приклад з `user`, якому присвоєне ім’я:

```
let user = "Іван";

alert(user ?? "Анонімний"); // Іван (user визначений)
```

Ми також можемо використовувати послідовність з `??`, щоб вибрати перше значення зі списку, яке не є `null/undefined`.

Скажімо, у нас є дані користувача в змінних `firstName`, `lastName` або `nickName`. Всі вони можуть бути не визначені, якщо користувач вирішив не вводити значення.

Ми хотіли б показати ім’я користувача, використовуючи одну з цих змінних, або показати “Анонімний”, якщо всі вони `null/undefined`.

Використаймо оператор `??` для цього:

```
let firstName = null;
let lastName = null;
let nickName = "Суперкодер";
```

```
// показує перше визначене значення:
alert(firstName ?? lastName ?? nickName ?? "Анонімний"); // Суперкодер
```

Порівняння з ||

Оператор АБО `||` може бути використаний таким же чином, як `??`, як це було описано вище.

Наприклад, у коді вище, ми могли б замінити `??` на `||` і все ще отримали б той самий результат:

```
let firstName = null;
let lastName = null;
let nickName = "Суперкодер";

// показує перше істинне значення:
alert(firstName || lastName || nickName || "Анонімний"); // Суперкодер
```

Історично, оператор АБО `||` був першим. Він існує з початку JavaScript, тому розробники використовували його для цих цілей протягом тривалого часу.

З іншого боку, оператор об'єднання з null `??` було нещодавно додано в JavaScript, і причиною того було те, що люди були не дуже задоволені `||`.

Важлива різниця між ними полягає в тому, що:

- `||` повертає перше *істинне* значення.
- `??` повертає перше *визначене* значення.

Інакше кажучи, оператор `||` не розрізняє, чи значення `false`, `0`, порожній рядок `""` чи `null/undefined`. Всі вони однакові – хибні значення. Якщо будь-яке з них є першим аргументом `||`, тоді ми отримаємо другий аргумент як результат.

Однак на практиці, ми хочемо використовувати типове значення лише тоді, коли змінна `null/undefined`. Тобто, коли значення дійсно невідоме/не встановлене.

Наприклад, розгляньте це:

```
let height = 0;

alert(height || 100); // 100
alert(height ?? 100); // 0
```

- `height || 100` перевіряє чи має змінна `height` має хибне значення, і `0` – це дійсно хибне значення.
  - отже, результатом `||` є другий аргумент, `100`.
- `height ?? 100` перевіряє змінну `height`, чи вона `null/undefined`, і це не так,
  - отже, результат – `height` "як є", тобто `0`.

На практиці нульова висота часто є дійсним значенням, яке не слід замінювати на типове значення. Отже, `??` робить саме те, що треба.

Пріоритет

Пріоритет оператора `??` такий самий, як у `||`. Він дорівнює `3` у таблиці MDN.

Це означає, що, як і `||`, оператор об'єднання з null `??` оцінюється до `=` та `?`, але після більшості інших операцій, таких як `+`, `*`.

// важливо: використовуйте дужки let area = (height ?? 100) * (width ?? 50);

alert(area); // 5000

```
В іншому випадку, якщо ми опускаємо дужки, то, оскільки `*` має вищий пріоритет, ніж `??`, то ві

```js
// без дужок
let area = height ?? 100 * width ?? 50;

// ...працює так само, як попередній вираз (мабуть, це не те, що ми хочемо):
let area = height ?? (100 * width) ?? 50;
```

Використання ?? разом з && або ||

З міркувань безпеки, JavaScript забороняє використання `??` разом з операторами `&&` та `||`, якщо пріоритет явно не вказаний дужками.

Код нижче викликає синтаксичну помилку:

```
let x = 1 && 2 ?? 3; // Синтаксична помилка
```

Обмеження є досить спірним, воно було додано до специфікації мови з метою уникнення помилок програмування, коли люди почнуть переходити з `||` до `??`.

Використовуйте явні дужки, щоб працювати з цим оператором:

```
let x = (1 && 2) ?? 3; // Працює

alert(x); // 2
```

Підсумки

- Оператор об'єднання з null `??` надає короткий спосіб вибору першого "визначеного" значення зі списку.

  Він використовується для присвоєння типових значень до змінних:

  ```
  // встановлює height=100, якщо height null чи undefined
  height = height ?? 100;
  ```

- Оператор `??` має дуже низький пріоритет – трохи вищий, ніж `?` та `=`, тому розглядайте додавання дужок при використанні його у виразах.
- Цей оператор заборонено використовувати з `||` або `&&` без явних дужок.
  ```