Instrukcje warunkowe w JS

Podstawą każdego programu czy skryptu jest możliwość podejmowania decyzji, czyli wykonania różnych działań w zależności od spełnienia określonych warunków logicznych lub matematycznych. Właśnie tym zajmuje się instrukcja warunkowa

`if`, którą niewątpliwie kojarzymy z zajęć szkolnych od w zasadzie pierwszych napisanych skryptów. Z decyzjami realizowanymi dzięki instrukcji `if` jest trochę jak z tzw. *questami* (misjami dla postaci) w grach RPG – przygoda zakończy się inaczej, w zależności od naszych wykonanych działań, a naszym celem jako programistów rozgrywki jest przewidzieć i okodować wszystkie możliwe scenariusze.

Dzięki instrukcji warunkowej możemy sprawić, że witryna poprawnie zareaguje na różne nieprzewidziane sytuacje, np. ostrzeże nas przed wpisaniem do koszyka w sklepie internetowym ujemnej lub niecałkowitej liczby zamówionych produktów, poinformuje o braku zaakceptowania regulaminu przez zaznaczenie checkboxa czy przypomni o konieczności podania jakieś wymaganej do zrealizowania wysyłki danej (typu nazwisko czy adres wysyłki). Bardzo też przyda się we wszystkich skryptach realizujących obliczenia matematyczne, przetwarzanie tekstów albo zmianę stylów elementów.

Poznajmy teraz dobrze składnię instrukcji, która wykonuje się “jeżeli” warunki logiczne zostaną spełnione.

Składnia instrukcji warunkowej if

Podstawowa konstrukcja instrukcji warunkowej wygląda tak:

```
if (warunek_logiczny) {
    // instrukcje do wykonania, jeśli warunek jest prawdziwy
} else {
    // instrukcje do wykonania, jeśli warunek jest fałszywy
}
```

Klauzula

`else` (*ang. w przeciwnym wypadku*) jest oczywiście opcjonalna – możemy ją pominąć, jeśli nie potrzebujemy reakcji na przypadek, kiedy warunek logiczny nie zostanie spełniony.

Przykład – sprawdzanie numeru PIN w banku

Załóżmy, że istnieje bankomat, który sprawdza poprawność wprowadzonego przez użytkownika kodu PIN. Jeśli ten czterocyfrowy kod jest poprawny – zakładamy, że jest to dla tego klienta wartość

`1945` – transakcja może zostać wykonana. Jeśli nie – użytkownik otrzyma komunikat o błędzie:

```
var pin = prompt("Podaj numer PIN:");
if (pin == 1945) {
    alert("Poprawny numer PIN. Dostęp do konta uzyskany.");
} else {
    alert("Niepoprawny numer PIN! Spróbuj ponownie.");
}
```

Zwróćmy uwagę na operator porównania

`==`, który sprawdza, czy dwie wartości są takie same. Nie należy go jednak pomylić z operatorem `=` (przypisania nowej wartości zmiennej). Takie przeoczenie to jeden z najczęściej popełnianych na egzaminach błędów:

```
var pin = prompt("Podaj numer PIN:");
if (pin = 1945) {   //BŁĄD!!! WIELBŁĄD!!!
    alert("Poprawny numer PIN. Dostęp do konta uzyskany.");
} else {
```

```
    alert("Niepoprawny numer PIN! Spróbuj ponownie.");
}
```

Mieliśmy przecież sprawdzić (porównać) wartość podaną przez użytkownika ze wzorcem

`1945`, a zamiast tego my tę wpisaną z klawiatury wartość pojedynczym operatorem `=` efektywnie nadpisujemy, wstawiając do zmiennej poprawny PIN. Totalnie zmienia to działanie skryptu i sprawia, że obojętnie co wpiszemy i tak zawsze się zalogujemy! A zwróćmy uwagę – to tylko **brak jednego znaku** tak bardzo zepsuł mechanizm autoryzacji – sytuacja niewątpliwie ucząca nas pokory. Więcej o słynnych przykładach błędów w oprogramowaniu (kosztujących czasem miliony dolarów, a nawet ludzkie życie!) opowiedziałem kiedyś w tym filmie.

Łączenie warunków – spójniki logiczne OR i AND

Stosunkowo często w warunkach logicznych (zapisywanych wewnątrz nawiasów okrągłych) potrzebujemy sprawdzić więcej niż jeden warunek jednocześnie. W tym celu używamy spójników logicznych:

- `||` OR (*ang. "lub"*) – warunek zostanie spełniony, jeśli **przynajmniej jeden** z warunków jest prawdziwy (a więc także w sytuacji gdy wszystkie okażą się prawdą).
- `&&` AND (*ang. "i"*) – warunek zostanie spełniony tylko wtedy, gdy **wszystkie** warunki składowe są jednocześnie prawdziwe.

Co ważne – w JavaScript nie można używać słownych wersji operatorów, czyli

`or`, `and` – w języku JavaScript trzeba się posłużyć operatorami `||`, `&&`.

Warunek A	Warunek B	OR (||)	AND (&&)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Jak to zrozumieć na praktycznych przykładach? Spójnik

`&&` wykorzystamy np. w mechanizmie logowania – trzeba wpisać zarówno poprawny login jak i hasło – jedynie obie właściwe wartości spowodują wpuszczenie nas do systemu, nie wystarczy podanie tylko właściwego loginu lub hasła.

A spójnik

`||` możemy zobrazować jako sterowanie w grze komputerowej typu *Counter Strike*, w której poruszamy się z użyciem klawiszy WSAD. Obojętnie czy naciśniemy małe "w" na klawiaturze, czy duże "W" (bo np. właśnie mamy włączony CAPS LOCK) – nasza postać powinna i tak poruszyć się do przodu. W matematyce nazywamy to *alternatywą* – istnieją w końcu dwie możliwości i każda z nich tak "poruszy" postać w świecie gry.

W matematyce używamy tych samych spójników logicznych do zapisów przedziałów – na przykład

`(3; 8)` oznacza przedział łączny – liczba należy do takiego przedziału tylko jeśli **jednocześnie** jest większa od 3 i mniejsza od 8. Stąd wartość 25 nie należy do przedziału, bo co prawda jest większa od 3, ale nie jest mniejsza od 8. Z kolei przedział tzw. rozłączny `(-∞; -5) ∪ (3; +∞)` to matematyczny przykład zastosowania spójnika OR – wystarczy, że liczba należy do jednego z tych przedziałów, np. -25 spełnia jedynie warunek bycia mniejszą od -5 a należy do przedziału pomimo nie bycia wartością większą od 3.

Spójniki logiczne w warunkach pozwalają nam uprościć zapisy warunków logicznych – nie trzeba ich zagnieżdżać w stylu:

```
if (imie == "Marek") {
```

```
    if (wiek >= 18) {
        alert("Poprawne logowanie. Witamy w systemie!");
    }
}
```

Tylko możemy zapisać łącznie:

```
if (imie == "Marek" && wiek >= 18) {
    alert("Poprawne logowanie. Witamy w systemie!");
}
```

Dzięki unikaniu wielokrotnych zagnieżdżeń poprawiamy znacząco czytelność źródłowego!

Spójnik OR – sprawdzanie pustych pól w formularzu

Załóżmy, że istnieją dwa pola tekstowe mające atrybuty kolejno:

`id="liczba1"` oraz `id="liczba2"`, w których użytkownik może wprowadzić liczby. Funkcja przypisana jest do obsługi zdarzenia kliknięcia przycisku. Po jej wywołaniu, jeżeli przynajmniej jedno z pól jest puste, to wyświetlimy komunikat o błędzie:

```
function check_values() {
    var a = document.getElementById("liczba1").value;
    var b = document.getElementById("liczba2").value;
    if (a == "" || b == "") {
        alert("Proszę uzupełnić obie liczby!");
    }
}
```

Spójnik AND – logowanie użytkownika

W przypadku systemu logowania, zarówno login, jak i hasło muszą być poprawne. Możemy to sprawdzić w następujący sposób:

```
var login = prompt("Podaj login:");
var haslo = prompt("Podaj hasło:");
if (login == "admin" && haslo == "5up3Rt4jn3h@sl0_4adm1N") {
    alert("Poprawne logowanie. Witamy w systemie!");
} else {
    alert("Niepoprawne dane logowania.");
}
```

Priorytet spójników – && ważniejszy niż ||

Podczas tworzenia warunków logicznych w JavaScript często korzystamy z operatorów

`||` i `&&`. Warto jednak wiedzieć, iż mają one różny priorytet (tak samo jak np. mnożenie jest “ważniejsze” w matematyce aniżeli dodawanie) – a zatem JavaScript wykona je w określonej kolejności. Warunki logiczne złączone operatorem `&&` (“and”) zawsze zostaną zatem ocenione wcześniej aniżeli te spięte spójnikiem `||` (“or”). Zobaczmy to na przykładzie:

```
if (imie == "Marek" || imie == "Adam" && wiek >= 18) {
    alert("Poprawne logowanie. Witamy w systemie!");
}
```

Na pierwszy rzut oka możemy odnieść wrażenie, że warunek zostanie oceniony od lewej do prawej – czyli najpierw sprawdzimy czy imię użytkownika to Marek lub Adam, a potem dodatkowo zweryfikujemy pełnoletność osoby. Jednak JavaScript najpierw oceni warunki złączone ważniejszym operatorem

`&&`, co oznacza że poprawnie zaloguje się tylko pełnoletni Adam, natomiast Marek zostanie wpuszczony do systemu niezależnie od wieku!

Aby zmienić działanie warunków tak, aby zarówno Marek jak i Adam musieli być osobami pełnoletnimi należy użyć nawiasów:

```
if ((imie == "Marek" || imie == "Adam") && wiek >= 18) {
    alert("Poprawne logowanie. Witamy w systemie!");
}
```

Jak widzimy, znajomość hierarchii rozpatrywania spójników logicznych jest kluczowa – to trochę jak w matematyce z działaniem

`2 + 2 * 2` – bez nawiasów zmieniających kolejność działań wynikiem jest 6, a nie 8 (mnożenie ma w końcu wyższy priorytet aniżeli dodawanie).

Zanegowanie warunku: NOT

Czasami w kodzie przydatna może okazać się operacja zanegowania wartości logicznej warunku – realizuje to następujący operator:

- `!` NOT (*ang. "nie"*) – tzw. negacja, która zmienia wartość logiczną po na przeciwną:

Warunek	NOT (!)
0	1
1	0

Podobnie jak przy spójnikach logicznych, w JavaScript nie można zapisać tekstowej wersji operatora, czyli

`not` zamiast `!` – taki zapis jest w JS niepoprawny składniowo i spowoduje błąd!

Załóżmy, że sprawdzamy czy użytkownik podał poprawną liczbę:

```
var input = prompt("Podaj liczbę:");
var number = Number(input);
if (!isNaN(number)) {
  console.log("To jest poprawna liczba");
} else {
  console.log("To nie jest poprawna liczba");
}
```

Jako, że funkcja

`isNaN()` (*ang. is Not a Number)* zwraca `true` dla wartości nie będącej liczbą, to zanegowanie jej operatorem `!` powoduje spełnienie warunku w przypadku odwrotnym – kiedy wartość jest poprawna.

Skrócony zapis instrukcji warunkowej

Czasami zdarza się, że chcemy sprawdzić jakiś warunek i w zależności od jego wyniku przypisać odpowiednią wartość do zmiennej lub wyświetlić komunikat. W takich prostych sytuacjach zamiast klasycznego

`if`, możemy użyć tzw. operatora warunkowego (*ang. ternary operator*) – czyli zapisu w skróconej formie:

```
warunek ? wartość_gdy_true : wartość_gdy_false;
```

Oczywiście najłatwiej zrozumieć to na konkretnym przykładzie – rozważmy ponownie sprawdzenie pełnoletności użytkownika:

```
var wiek = prompt("Podaj swój wiek:");
var komunikat = age >= 18 ? "Pełnoletni" : "Niepełnoletni";
alert(komunikat);
```

Jak widzimy, następuje tutaj przypisanie innej wartości w zależności od spełnienia (bądź nie) warunku.

Trzeci możliwy scenariusz: else if

Instrukcja warunkowa nie musi obsługiwać tylko dwóch scenariuszy – gdy warunek logiczny jest prawdziwy lub gdy zwraca wartość

`false`. Dzięki instrukcji `else if` możliwe jest dodawanie kolejnych obsługiwanych “zakończeń” naszej instrukcji:

```
var wiek = prompt("Podaj swój wiek:");
if (wiek >= 35) {
    alert("Możesz zostać prezydentem i jesteś osobą pełnoletnią");
} else if (wiek >= 18) {
    alert("Jesteś osobą pełnoletnią, ale nie możesz jeszcze zostać prezydentem");
} else {
    alert("Nie jesteś osobą pełnoletnią oraz nie możesz jeszcze zostać prezydentem");
}
```

Obsługujemy tutaj nie dwa, a trzy możliwe scenariusze – prezydentem możemy zostać w wieku co najmniej 35 lat, a pełnoletność osiągamy mając lat przynajmniej 18. Oczywiście możliwych scenariuszy możemy dodać jeszcze więcej, w zależności od liczby zaimplementowanych instrukcji

`else if`.

Podsumowanie – najważniejsze informacje

- Instrukcja warunkowa `if` pozwala sterować wykonaniem programu na podstawie spełnienia (bądź nie) warunków logicznych.
- Operator `==` porównuje wartości, natomiast `=` służy do przypisywania nowej wartości – nie możemy tego mylić!
- `else` pozwala obsłużyć sytuację, gdy warunek w `if` nie zostanie spełniony.
- Spójniki logiczne `||`, `&&` umożliwiają tworzenie bardziej złożonych warunków, dzięki czemu nie trzeba zagnieżdżać `if` tylko połączyć warunki.
- Instrukcja `else if` pozwala obsłużyć więcej niż dwa przypadki.
- Spójnik `&&` ma wyższy priorytet od `||`, podobnie jak w matematyce mnożenie jest w kolejności przed dodawaniem.
- Operator trójargumentowy `? :` tworzy skrócony zapis instrukcji `if` przypisujący inną wartość zależnie od spełnienia warunku.