ÉS (&&) és VAGY (||) Operátorok Működése és a Rövidzár

JavaScriptben az ÉS (&&) és a VAGY (||) operátorok logikai műveletek végrehajtására szolgálnak. Ezek az operátorok az úgynevezett "rövidzáras kiértékelést" (short-circuit evaluation) használják, ami azt jelenti, hogy a kifejezések kiértékelése során a lehető legkevesebb műveletet végzik el, és a kifejezés értékelését megállítják, amint a végeredmény egyértelművé válik.

ÉS Operátor (&&)

Az ÉS operátor két kifejezés közötti logikai ÉS kapcsolatot valósít meg. Csak akkor ad vissza truthy értéket, ha mindkét operandus igaz lenne logikai (boolean) értékké konvertálva.

Működése:

1. Kiértékeli az első operandust.
2. Ha az első operandus falsy, akkor az első operandussal tér vissza, és a második operandus kiértékelése nem történik meg (rövidzár).
3. Ha az első operandus truthy, akkor kiértékeli a második operandust, és annak értékét adja vissza.
4. Ha egyik operandus sem lett truthy akkor az utolsó operandust adja vissza

Példák:

```
1 && 1; // 1
1 && 0; // 0
null && "szöveg"; // null
undefined && null; // undefined
1 && null && undefined; // null
```

Rövidzáras Kiértékelés Példa:

```
let a = 10;
let b = 0;
a > 5 && (b = b + 1); // a > 5 igaz, tehát b = b + 1 kiértékelésre kerül, b értéke 1 lesz

let c = 0;
a < 5 && (c = c + 1); // a < 5 hamis, tehát c = c + 1 nem kerül kiértékelésre, c értéke marad 0
```

VAGY Operátor (||)

A VAGY operátor két kifejezés közötti logikai VAGY kapcsolatot valósít meg. Csak akkor ad vissza falsy értéket, ha mindkét operandus hamis falsy.

Működése:

1. Kiértékeli az első operandust.
2. Ha az első operandus trutgh, akkor az első operandussal tér vissza, és a második operandus kiértékelése nem történik meg (rövidzár).
3. Ha az első operandus falsy, akkor kiértékeli a második operandust, és annak értékét adja vissza.
4. Ha mindegyik operandus falsy volt akkor az utolsót adja vissza

Példák:

```
1 || 2; // 1
2 || null; // 2
undefined || "szöveg"; // "szöveg"
undefined || null; // null
```

Rövidzáras Kiértékelés Példa:

```
let x = 10;
let y = 0;
x > 5 || (y = y + 1); // x > 5 igaz, tehát y = y + 1 nem kerül kiértékelésre, y értéke marad 0

let z = 0;
x < 5 || (z = z + 1); // x < 5 hamis, tehát z = z + 1 kiértékelésre kerül, z értéke 1 lesz
```

Miért és Hogyan Használjuk a Rövidzárat?

Miért használjuk?

1. **Hatékonyság**: Rövidzáras kiértékelés javítja a kód hatékonyságát, mivel felesleges műveletek kiértékelését elkerüli.
2. **Biztonság**: Bizonyos műveletek elkerülhetők, amelyek hibaüzeneteket eredményeznének, ha előzőleg nem vizsgáljuk meg őket. Például objektum tulajdonságainak ellenőrzése előtt megvizsgálhatjuk, hogy az objektum létezik-e.

Hogyan használjuk?

1. **Értékadás csak szükség esetén**:

```
let isLoggedIn = true;
isLoggedIn && showWelcomeMessage(); // A showWelcomeMessage csak akkor kerül meghívásra, ha az isLoggedIn igaz
```

2. **Biztonságos hozzáférés objektum tulajdonságokhoz**: (erről majd később beszélünk)

```
let user = null;
let userName = user && user.name; // userName csak akkor kap értéket, ha user nem null
```

3. **Alapértelmezett érték megadása**:

```
let name = providedName || "Default Name"; // Ha providedName hamis érték (null, undefined, ""), akkor a name "Default Name" le
```

ÉS (&&) és VAGY (||) Operátorok Műveleti Sorrendje JavaScriptben

A logikai operátoroknak, mint az ÉS (&&) és a VAGY (||), meghatározott műveleti sorrendjük (precedenciájuk) van JavaScriptben. Ez azt jelenti, hogy amikor egy kifejezés több logikai operátort tartalmaz, a JavaScript egy adott sorrendben értékeli ki azokat.

Műveleti sorrend (precedencia)

1. **Logikai NOT (!)**: A legmagasabb precedenciával rendelkezik, tehát először kerül kiértékelésre.
2. **Logikai ÉS (&&)**: Közepes precedenciával rendelkezik, az operátorok között az ÉS kerül előbb kiértékelésre, mint a VAGY.
3. **Logikai VAGY (||)**: A legalacsonyabb precedenciával rendelkezik a három közül.

Működés és Példák

Logikai NOT (!)

A logikai NOT operátor minden más logikai operátor előtt kerül kiértékelésre.

```
let a = true;
let b = false;

!a; // false
!b; // true
```

Logikai ÉS (&&)

Az ÉS operátor precedenciája magasabb, mint a VAGY operátoré, tehát az ÉS műveletek előbb kerülnek kiértékelésre, mint a VAGY műveletek.

```
let a = true;
let b = false;
let c = true;

(a && b) || c; // (a && b) || c
// a && b -> false (mert b false)
// false || c -> true (mert c true)
```

Logikai VAGY (||)

A VAGY operátor precedenciája alacsonyabb, mint az ÉS operátoré, tehát a VAGY műveletek a legutolsók, amelyek kiértékelésre kerülnek, ha nincs zárójel használva.

```
let a = false;
let b = true;
let c = true;

a || (b && c); // a || (b && c)
// b && c -> true (mert mindkettő true)
// a || true -> true (mert a VAGY művelet eredménye true, ha bármelyik operandus true)
```

Zárójelek használata a műveleti sorrend módosításához

Zárójeleket használhatunk a műveleti sorrend megváltoztatásához és a kifejezés olvashatóságának javításához.

Példák zárójelekkel:

```
let a = false;
let b = true;
let c = false;

(a || b) && c; // Először a (a || b) kifejezés kerül kiértékelésre
// a || b -> true (mert b true)
// true && c -> false (mert c false)

a || (b && c); // Először a (b && c) kifejezés kerül kiértékelésre
// b && c -> false (mert c false)
// a || false -> false (mert mindkettő false)
```

Összefoglalás

- **Logikai NOT (!)**: A legmagasabb precedenciával rendelkezik, tehát először kerül kiértékelésre.
- **Logikai ÉS (&&)**: Közepes precedenciával rendelkezik, és előbb kerül kiértékelésre, mint a VAGY (||).
- **Logikai VAGY (||)**: A legalacsonyabb precedenciával rendelkezik, tehát utoljára kerül kiértékelésre, ha nincs zárójel használva.

A zárójelek használatával módosíthatjuk a kifejezések kiértékelésének sorrendjét, ami lehetővé teszi a logikai műveletek sorrendjének egyértelműbbé és olvashatóbbá tételét.