JavaScript alapok összefoglaló

Változók definiálása

Három kulcsszó

- var (régebbi projektekben van jelentősége)
- **const**
- **let**

const és let közötti különbség:

```
let myFirstVar = 'Hello';
myFirstVar = 5; // Akárhányszor adhatunk értéket

const onlyOnce = 'World'; // Pontosan egyszer adhatunk értéket
onlyOnce = 'Hello'; // Hibát kapunk a böngészőben
```

Érték típusok

Teljes lista: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures

Primitív értékek

```
// String
const s1 = 'I am a string';
const s2 = "Also a string";
const s3 = `String between backticks`; // template string
```

```
const age = 27;
const text = `I am ${age} years old`; // egyszerű behelyettesítés
const text2 = 'I am ' + age + 'years old';
```

```
// Number
const n1 = 5;
const n2 = 3.4343;
const n3 = Number('2');
const n4 = Number('not a number'); // NaN
// bool
let isValid = true;
isValid = false;
```

Referencia értékek

Object és leszármazottai

```
// Object
const o = {
   name: 'Zizi',
   favoriteNumber: 5
};
// Array
const a1 = [1, 2, 3];
const a2 = [1, 'Zizi', true];
// Function
const f = (a, b) => { return a + b; } // arrow function szintakszis
```

Értékátadás

JavaScriptben az értékátadás (pl.: függvényhívás, változó értékadás) másolással történik, primitív típus esetén az átadott érték másolódik, referencia típus esetén az adatszerkezetre mutató referencia kerül duplikálásra. Ebből a működésből adódóan az alábbiakra kell figyelnünk:

```
let num = 5;
const foo = (n) => {
   n = n * 2; // Másolaton dolgozunk, num értéke ettől még változatlan
   return n; // Vissza is kell térni a módosított értékkel
}

num = foo(num);
```

```
let  pet = {
   name: 'Kitty',
   type: 'cat'
};

const bar = (p) => {
   p = {
       name: 'Doggy',
       type: 'dog'
   } // Másolat referencián dolgozunk
   return p; // Vissza kell térni a módosított referenciával
};

pet = bar(pet);
```

A másolat referencia az eredeti adatszerkezetre mutat, így, ha az adatszerkezetben módosítunk, akkor az a “pet” változó értékét is módosítja:

```
const bar2 = (p) => {
   p.name = 'Snakey';
   p.type = 'snake';
};

bar2(pet);
```

Null és undefined

```
let v; // inicializálatlan változó értéke undefined
console.log(v);
v = null;
console.log(v);
v = undefined;
```

Dinamikus típusosság

JavaScriptben típusa csak az értékeknek van, a változóknak nincsen, ennek következménye, hogy egy változónak több féle érték és értékül adható:

```
let d = 5;
d = '5';
d = [1, 2, 3];
d = (i) => {console.log(i)};

d('JavaScript');
```

Closure

```
const makeClosure = () => {
    const name = 'Closure';
    const displayName = () => {
      alert(name);
    }
    return displayName;
  }

const testFn = makeClosure();
testFn();
```

Elágazások és ciklusok

Teljes lista: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements

If-else elágazás

```
const isPositive = n => {
    if (n >= 0) {
        console.log('positive');
    } else {
        console.log('negative');
    }
}

isPositive(5);
isPositive(-5);
```

JavaScriptben mindenféle érték és kifejezés leképezhető logikai értékre, így az if ágban tetszőleges kifejezés és érték szerepelhet:

```
const isTruthy = v => {
    if (v) {
        console.log('Truthy');
    } else {
        console.log('Falsy');
    }
}

isTruthy({});
isTruthy('');
```

- A “true”-ra értékelődő ún. truthy értékek listája: https://developer.mozilla.org/en-US/docs/Glossary/Truthy
- A “false”-ra értékelődő ún. falsy értékek listája: https://developer.mozilla.org/en-US/docs/Glossary/Falsy

Switch-case

```
const mapToStars = n => {
    let res;
    switch (n) {
        case 1: res = '*'; break;
        case 2: res = '**'; break;
        case 3: res = '***'; break;
        case 4: res = '****'; break;
```

```
        case 5: res = '*****'; break;
        default: res = '-'
    }

    return res;
}

console.log(mapToStars(2));
console.log(mapToStars(0));
```

Try-catch-throw

Ha egy olyan kódrészletet futtatunk, ahol futási időben fenn áll a hiba lehetősége, akkor az érintett kódrészletet egy try blokkba ágyazva tudjuk lefuttatni, egy kapcsolódó catch blokkal, pedig meg tudjuk mondani, hogy hiba esetén mi történjen

```
const throwIfFalsy = v => {
    if (!v) {
        throw new Error('Falsy value');
    }
}

try {
    throwIfFalsy('Truthy');
    console.log('success');
} catch (err) {
    console.error('Caught error', err);
}

try {
    throwIfFalsy(false);
    console.log('success2');
} catch (err) {
    console.error('Caught error2', err);
}
```

For ciklus

```
for(let i = 0; i <= 4; i++) {
    console.log('For loop', i + 1);
}
```

While ciklus

```
let i = 0;
while (i <= 4) {
    console.log('While loop', i + 1);
    i++;
}
```

Operátorok

Teljes lista:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators

Egyenlőség operátor

JavaScriptben két féle egyenlőség operátor létezik, a == nem várja el, hogy az operandusok megegyező típusúak legyenek, eltérés esetén közös típusra hozza őket, és így végzi el az összehasonlítást. A === csak azonos típusú operandusok esetén jelez egyenlőséget. Referencia típusoknál nincs “deep equal” típusú egyezőség figyelés, csak a referencia egyezőségét figyeljük.

```
const a = '5';
const b = 5;
console.log(a == b);
```

```
console.log(a === b);
const p1 = {name: 'Zizi'};
const p2 = {name: 'Zizi'};
console.log(p1 === p2); // Referencia típusoknál nincs deep equal
const p3 = p1;
console.log(p1 === p3); // Csak a referencia
```

Aritmetikai és értékadási operátorok

```
let x = 5;
let y = 4;

console.log(x + y); // 9
console.log(x - y); // 1
console.log(x * y); // 20
console.log(x / y); // 1.25
```

```
console.log(x % y); // 1

x += y; // x = x + y
console.log(x); // 9

x++ // x = x + 1
console.log(x); // 10
```

Logikai operátorok

A logikai operátorokat nem csak bool típusú értékekkel használhatjuk:

```
console.log(!true);

const e1 = 'Cat' && 'Dog';    // t && t returns Dog
const e2 = false && 'Cat';    // f && t returns false
const e3 = 'Cat' && false;    // t && f returns false

console.log(e1, e2, e3);

const e4 = 'Cat' || 'Dog';    // t || t returns Cat
const e5 = false || 'Cat';    // f || t returns Cat
const e6 = 'Cat' || false;    // t || f returns Cat
const e7 = false || false;    // f || f returns false

console.log(e4, e5, e6, e7);
```

?: operátor

A ?: operátor vár egy logikai kifejezést, amely eredményétől függően a neki megadott első, vagy második érték lesz az operátor értéke:

```
const isAdult = age => {
    let res;
    res = age <= 18 ? false : true;
    return res;
}

console.log(isAdult(16)); // false
console.log(isAdult(22)); // true
```

Delete operátor

A delete operátor segítségével objektumok kulcsait, és a hozzá tartozó értéket tudjuk törölni:

```
const person = {
    name: 'John',
    email: 'john@example.com'
}

delete person.email;

console.log(person);
```

Typeof és instanceof operátorok

A “typeof” operátor segítségével primitív értékek típusát tudjuk lekérdezni, referencia típusok esetén ez azonban mindig Objectet ad vissza, kivéve a függvényeket, ahol Function lesz a típus. A referencia típusokról pontosabb típus információt az “instanceof” operátor segítségével kaphatunk:

```
console.log(typeof 5); // number
console.log(typeof 'a string'); // string
console.log(typeof []); // object
console.log(typeof {}); // object
console.log(typeof new Date()); // object
console.log(typeof (() => {})); // function
console.log([] instanceof Array); // true
console.log([] instanceof Object); // true
console.log({} instanceof Object); // true
console.log(new Date() instanceof Date); // true
```