Část z textu: *Michal Nykl - Moderní JavaScript v příkladech (ECMAScript 2009-2021)*

1.1 Deklarování proměnných využitím *VAR*, *LET* nebo *CONST* a blokový rozsah platnosti proměnných a funkcí

Abychom mohli omezit rozsah platnosti proměnné byla nově umožněno deklarovat proměnné klíčovými slůvky *let* a *const*. Nyní je tedy proměnná deklarovaná s využitím *var* stále dostupná i uvnitř bloků (dány zápisem {..}) a je umožněno její opětovné deklarování využitím *var*. Deklarace s *let* naopak zajišťuj blokový rozsah platnosti a zamezuje opětovné deklaraci téže proměnné v rámci daného bloku. Využít lze také deklaraci konstant s *const*, která poskytuje totéž, co *let*, ale navíc zamezuj přepsání hodnoty. Ukázka rozdílných vlastností a rozsahů platnosti proměnných deklarovaných využitím *var*, *let* nebo *const*, ukazují následující příklady. Poslední dva příklady ukazují, jaký rozsah platnosti mají funkce.

Ukázka rozsahu platnosti proměnné při deklaraci proměnné prostřednictvím *VAR*:

```
var x = 1; // nová proměnná
console.log(x);  // vypíše: 1

// funkce má vlastní rozsah platnosti
function varTest() {
   console.log(x);  // vypíše: undefined
   var x = 2; // nová proměnná

   { // nový samostatný blok kódu
      console.log(x);  // vypíše: 2
      var x = 3;  // přepsání předchozí proměnné (2)
      console.log(x);  // vypíše: 3
   }

   console.log(x);  // vypíše: 3

   var x = 4;  // přepsání předchozí proměnné (2)
   console.log(x);  // vypíše: 4

   // vnitřní funkce s vlastním rozsahem platnosti
   function interVarTest() {
      console.log(x);  // vypíše: undefined
      var x = 5;  // nová proměnná
      console.log(x);  // vypíše: 5

      { // nový samostatný blok kódu
         console.log(x);  // vypíše: 5
         var x = 6;  // // přepsání předchozí proměnné (5)
         console.log(x);  // vypíše: 6
      }

      console.log(x);  // vypíše: 6
   }
   interVarTest(); // volání funkce
   console.log(x);  // vypíše: 4
}

varTest(); // volání funkce
console.log(x);  // vypíše: 1
```

Ukázka rozsahu platnosti proměnné při deklaraci proměnné prostřednictvím *LET* nebo *CONST*:

```
let x = 1; // nová proměnná
const c = "Conct-1"; // nová proměnná
console.log(x, c);  // vypíše: 1, Conct-1

// funkce má vlastní rozsah platnosti, ale c deklarované s CONST je uplatněno
function letTestA() {
   console.log(x, c);  // vypíše: undefined, Const-1
}
letTestA(); // volání funkce

// funkce má vlastní rozsah platnosti
function letTestB() {
   // console.log(x,c);  // vyvolá výjimku: SyntaxError: Cannot access 'x' and 'c' before initialization
   let x = 2; // nová proměnná
   const c = "Const-2"; // nová proměnná
   console.log(x, c); // vypíše: 2, Const-2

   { // nový samostatný blok kódu
      // protože zde není nová deklarace, tak jsou použity předchozí hodnoty
      console.log(x, c); // vypíše: 2, Const-2
   }

   { // nový samostatný blok kódu
      // console.log(x,c);  // vyvolá výjimku: SyntaxError: Cannot access 'x' and 'c' before initialization
      let x = 3; // nová proměnná
      const c = "Const-3"; // nová proměnná
      console.log(x, c); // vypíše: 3, Const-3
   }
   console.log(x, c); // vypíše: 2, Const-2

   // vnitřní funkce s vlastním rozsahem platnosti
   function interVarTestA() {
      // protože zde není nová deklarace, tak jsou použity předchozí hodnoty
      console.log(x,c);  // vypíše: 2, Const-2
   }
   interVarTestA(); // volání funkce

   // vnitřní funkce s vlastním rozsahem platnosti
   function interVarTestB() {
      // console.log(x,c);  // vyvolá výjimku: SyntaxError: Cannot access 'x' and 'c' before initialization
      let x = 4;  // nová proměnná
      const c = "Const-4";
      console.log(x, c); // vypíše: 4, Const-4
   }
   interVarTestB(); // volání funkce
   console.log(x,c);  // vypíše: 2, Const-2

   // let x = 4;  // nelze znovu deklarovat, výjimka: SyntaxError: Identifier 'x' has already been declared
   // const c = 4; // nelze znovu deklarovat, výjimka: SyntaxError: Identifier 'c' has already been declared

   x = 5; // proměnné deklarované s let lze přepsat
   // c = "Const-5"; // pokus o přepsání konstanty vypíše výjimku: TypeError: Assignment to constant variable.
}
letTestB(); // volání funkce
console.log(x, c); // vypíše: 1, Const-1
```

Ukázka rozdílu mezi *CONST*, *LET* a *VAR* při použití v cyklu:

```
// VAR
for(var a of [2,4]){
   a = 7; // lze provést
   console.log(a);  // vypíše postupně: 7, 7
}
console.log(a); // proměnná je platná i po ukončení cyklu, vypíše: 7

// LET
for(let b of [2,4]){
   b = 6; // lze provést
   console.log(b);  // vypíše postupně: 6, 6
}
// console.log(b); // nelze vypsat, výjimka: ReferenceError: a is not defined

// CONST
for(const c of [2,4]){
   // c = 5; // nelze přepsat konstantu, výjimka: TypeError: Assignment to constant variable.
}
// console.log(c); // nelze vypsat, výjimka: ReferenceError: a is not defined
```

Přiřazení do objetku či pole není považováno za změnu konstanty:

```
// přiřazení do objektu v konstantě
const c = { x: 4, y: 0}
// c = {x: 7}; // nelze přepsat konstadntu, výjimka: TypeError: Assignment to constant variable.
c.x = 7;
console.log(c); // vypíše: { x: 7, y: 0 }

// přiřazení do objektu v konstantě
for(const c of [{x: 5, y: 11},{x: 6, y: 22}]){
   // c = {x: 8}; // nelze přepsat konstadntu, výjimka: TypeError: Assignment to constant variable.
   c.x = 8; // lze provést
   console.log(c);  // vypíše postupně: { x: 8, y: 11 }, { x: 8, y: 22 }
}

// přiřazení do pole v konstantě
for(const c of [[10,11],[20,22]]){
   // c = [40,44]; // nelze přepsat konstantu, výjimka: TypeError: Assignment to constant variable.
   c[0] = 9; // lze provést
   console.log(c);  // vypíše postupně: [ 9, 11 ], [ 9, 22 ]
}
```

Ukázka blokového rozsahu platnosti při deklarování funkce:

```
console.log( fceA() ); // vypíše: 7
function fceA() { // deklarace funkce f()
   return 7;
}
console.log( fceA() ); // vypíše: 7

{ // nový blok - nový rozsah platnosti
   console.log( fceA() ); // vypíše: 8
   function fceA() { // deklarace nové funkce f() uvnitř bloku
      return 8;
   }
   console.log( fceA() ); // vypíše: 8
} // ukončení bloku
console.log( fceA() ); // vypíše: 8
```

Funkce deklarovoná uvnitř jiné funkce má rozsah platnosti pouze uvnitř nadřazené funkce:

```
console.log( fce() ); // vypíše: true
function fce(){

   console.log( fce() ); // vypíše: 22
   function fce(){
      return 22;
   }
   console.log( fce() ); // vypíše: 22

   function test(){
      return 33;
   }
   console.log( test() ); // vypíše: 33

   return 10;
}
console.log( fce() ); // vypíše: 10
// console.log( test() ); // funkce v tomto kontestu není deklarována, výjimka: ReferenceError: test is not defined
```