# Webové technológie 6

## Aplikácie Webových technológií

Peter Bednár

# Polia

- Premennú poľa je možné vytvoriť a inicializovať uvedením prvkov poľa v [] zátvorkách, napr.:

```
var dni = ["pondelok", "utorok", "streda"];
```

- Podobne je možné inicializovať viacrozmerné polia, prvky poľa nemusia mať ten istý typ
- K prvkom sa pristupuje indexovaním od 0, napr. pre zmenu prvého prvku:

```
dni[0] = "nedeľa";
```

- Hodnoty pre indexy mimo aktuálnu veľkosť poľa sú `undefined`
- Dĺžka poľa

```
dni.length
```

# Metódy polí

- K poliam je možné pristupovať ako k objektom s metódami
- Základné metódy
  - `pole.push(obj)` – Pridá na koniec poľa nový prvok `obj`
  - `pole.pop()` – Vráti a odstráni posledný prvok poľa
  - `pole.shift()` – Vráti a odstráni prvý prvok poľa
  - `pole.splice(from, count)` – Vráti a odstráni prvky poľa od indexu `from`, `count` udáva počet odstránených prvkov
  - `pole.splice(from, 0, obj1, obj2, ...)` – Pridá do zoznamu zadané prvky od indexu `from`
  - `pole.toString()` – Prevedie pole na reťazec (zoznam hodnôt oddelených `,`

# Funkcie

- Podprogramy, ktoré môžu mať parametre (argumenty) a môžu vracať jednu hodnotu

```
function sum(a, b) {
    var c = a + b;   // c je lokálna premenná
    return c;
}
// mimo bloku funkcie, c nie je definovaná
```

- Príkaz `return` preruší vykonávanie funkcie a vráti hodnotu výrazu ako návratovú hodnotu funkcie
- Volanie funkcie:

```
var s = sum(1, 4);
```

- Ak sa pri volaní predá menší počet parametrov, ostatné budú nedefinované, napr.:

```
s = sum(2);   // a = 2, b = undefined
```

# Premenlivý počet parametrov

- Hodnoty všetkých parametrov, ktoré boli predané funkcii pri volaní sú prístupné v lokálnej premennej `arguments`, ku ktorej sa pristupuje ako k poľu, napr.:

```
function sum() {
    var sum = 0;
    for (var x in arguments) {
        sum += x;
    }
    return sum;
}
var s1 = sum();             // = 0
var s2 = sum(10, 20, 30); // = 60
```

# Objekty

- Objekty v JavaScripte sú kolekcie dvojíc kľúč:hodnota
  - Vlastnosti – dátové hodnoty (primitívne typy, polia alebo odkazy na ďalšie objekty
  - Metódy – funkcie priradené objektu
- Definovanie objektu:

```
var person = {
    name: "Peter",
    surname: "Bednár"
}
```

- Prístup k vlastnostiam, napr.:

```
person.age = 41;
person["age"] = 41;    // alternatívny zápis cez []
```

# Konštruktory

- Konštruktory a prototypy slúžia na vytváranie a inicializáciu rôznych objektov toho istého typu
- Konštruktor – vytvorí a inicializuje nový objekt:

```
function Person(name, surname, age) {
    this.name = name;
    this.surname = surname;
    this.age = age;
    this.nationality = "sk";
}
```

- Vytvorenie nového objektu:

```
var person1 = new Person("Peter", "Bednár", 40);
var person2 = new Person("John", "Doe", 150);
```

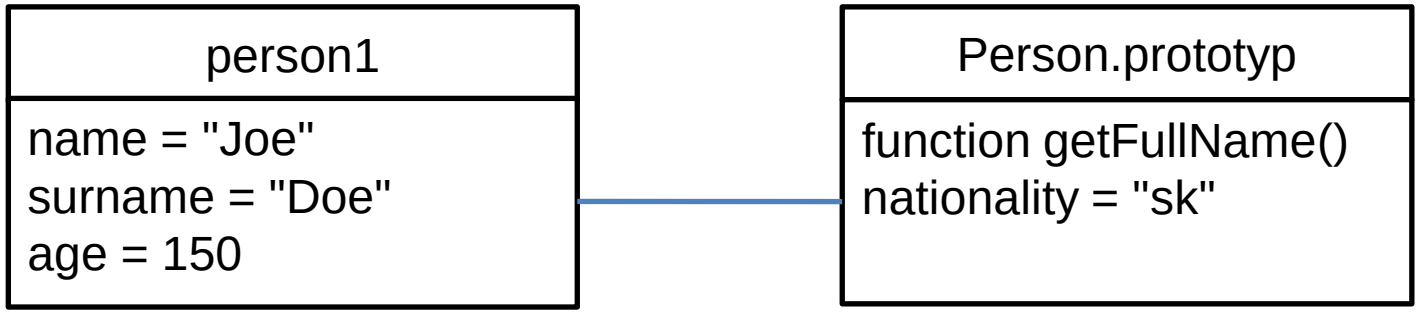
# Prototypy

- Prototypy umožňujú rozšíriť objekt vytvorený konštruktorom o nové vlastnosti, alebo metódy, napr.:

```
function Person(name, surname, age) {
    this.name = name;
    this.surname = surname;
    this.age = age;
}
Person.prototype.nationality = "sk";
Person.prototype.getFullName = function() {
    return this.name + " " + this.surname;
}
var person = new Person("Peter", "Bednár", 40);
var fullName = person.getFullName();
```

# Prototyp a objekt (1)

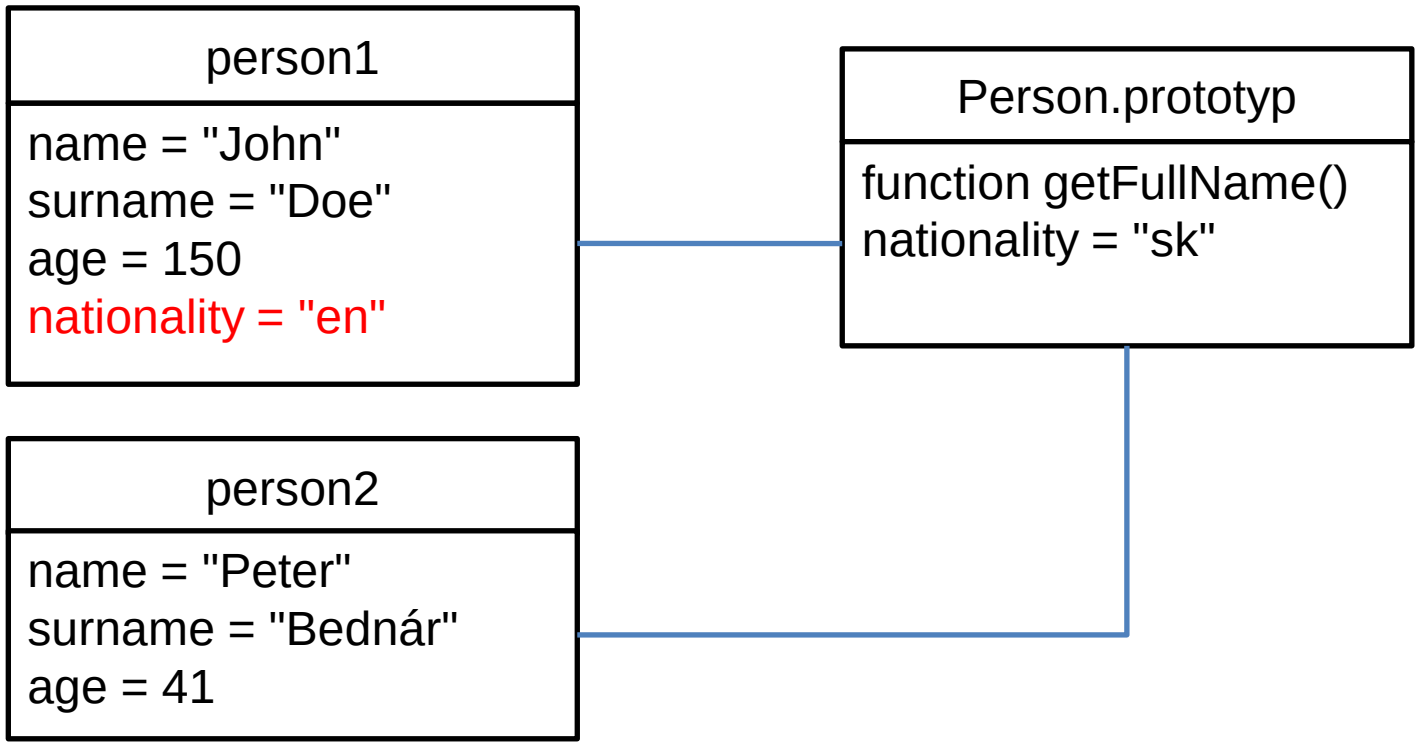
```
var person1 = new Person("John", "Doe", 150);
person1.name                    // = "John"
person1.getFullName()           // = "John Doe"
person1.nationality             // = "sk"
```




Ak objekt priamo nedefinuje premennú, alebo metódu, použije sa definícia z prototypu

# Prototyp a objekt (2)

```
person1.nationality = "en";
person1.nationality   // = "en"
var person2 = new Person("Peter", "Bednár", 41);
```



# Testovanie typov

- Typ premennej môžeme testovať operátorom `typeof`, ktorý vráti reťazec s názvom typu, napr.:

`typeof(42) = "number"` – číslo
`typeof("str") = "string"` – reťazec
`typeof(true) = "boolean"` – Boolovská hodnota
`typeof([]) = "array"` – pole
`typeof(x) = "undefined"` – ak je premenná x deklarovaná, ale nemá priradenú hodnotu
`typeof(function(){}) = "function"` – funkcia
`typeof(new Person()) = "object"` – objekt
`typeof(null) = "object"`

# Vetvenia (1)

- Základný príkaz `if-then-else` podobne ako v C
- Časť `else` je nepovinná, je možné zreťaziť viacero vetvení napr.:

```
if (hodina > 18) {
    pozdrav = "Dobrý večer"
} else if (hodina > 11) {
    pozdrav = "Dobrý deň"
} else {
    pozdrav = "Dobré ráno"
}
```

- Pre podmienené priradenie je možné použiť ternárny operátor:

```
var vysledok = body >= 51 ? "prešiel" : "neprešiel";
```

# Vetvenia (2)

- Viacnásobné vetvenie podľa kľúča – príkaz `switch-case`
- Výraz pre výpočet kľúča sa vyhodnotí iba raz, kľúč môže byť číslo, alebo reťazec
- Porovnávanie je ako pri operátore ===, tzn. nerobí sa žiadna typová konverzia

```
switch (today.getDay()) {
    case 6:  text = "Dnes je nedeľa";
             break;
    case 0:  text = "Dnes je sobota";
             break;
    default: text = "Pracovný deň";
}
```

# Cykly (1)

- Základný aritmetický cyklus:

```
var sum = 0;
for (var i = 0; i < pole.length; i++) {
    sum += pole[i]
}
// premenná i je definovaná aj mimo bloku for
```

- Cez prvky poľa je možné priamo iterovať skráteným zápisom:

```
for (var x in pole) {
    sum += x
}
```

- Cyklus je možné kedykoľvek ukončiť príkazom `break`

# Cykly (2)

- Prefixový cyklus:

```
var i = 0;
while (i < pole.length) {
    sum += pole[i];
    i++;
}
```

- Postfixový cyklus (telo sa vykoná aspoň raz):

```
var i = 0;
do {
    sum += pole[i];
    i++;
} while (i <= pole.length)
```

# Programovanie dynamických stránok

# JavaScript v prehliadači

- Chceme dynamicky zmeniť obsah HTML stránky podľa toho, aké sú vstupy od používateľa
  - Aké hodnoty zadal do formulára
  - Na ktorý element v stránke klikol, alebo presunul myš
  - Akú klávesu stlačil
  - a pod.
- Chceme dynamicky zmeniť obsah HTML stránky podľa dát, ktoré získame zo servera – webovej služby
- Chceme odoslať dáta na server/webovú službu

# Vloženie JavaScript kódu do HTML

- Priamo ako obsah značky `<script>`:

```
<script>
    // kód JavaScript-u
    function myFunction() {
        alert("Ahoj!");
    }
    myFunction();
</script>
```

- Do samostatného súboru na ktorý sa odkazuje cez URL (preferovaný spôsob):

```
<script src="skript.js" />
```

# Atribúty elementu `<script>`

- `async` – binárny atribút, ak je uvedený, skript bude vykonaný asynchrónne s načítavaním stránky (tzn. ak sa kód odkazuje na niektoré elementy HTML, ešte nemusia byť načítané), platí iba pre externé skripty načítané zo súboru
- `defer` – binárny atribút, ak je uvedený, skript bude vykonaný až po úplnom načítaní stránky, platí iba pre externé skripty načítané zo súboru
- Ak nie je uvedený ani atribút `sync` ani `defer`, skript sa načíta a vykoná okamžite a až po jeho ukončení sa pokračuje v načítavaní nasledujúcej časti stránky

# Externé knižnice

- JavaScript knižnice pre programovanie webových aplikácií sú distribuované ako .js súbory
  - Je možné ich stiahnuť a uložiť lokálne ako časť projektu
  - Odkazovať sa priamo na server cez externú URL – *Content Delivery Network* (sieť pre doručenie obsahu) – rýchle načítanie podľa polohy používateľa
- Väčšina knižníc existuje v dvoch verziách:
  - Úplná verzia .js – dobre čitateľná pre vývojára
  - Minimalizovaná verzia .min.js – skrátené názvy a vynechané formátovanie pre čo najmenšiu veľkosť súboru a rýchle načítanie
  - Je vhodné minimalizovať aj vlastný kód, napr.: http://jscompress.com/

# Príklad

- Jednoduchá aplikácia pre správu úloh (TODO list)

# Štruktúra projektu

- Archív súborov: todo.zip

- `css`
  - `todo.css` - CSS súbory
- `js`
  - `todo.js` - JavaScript súbory
- `index.html` - hlavná HTML stránka aplikácie

# Štruktúra HTML stránky (1)

- Hlavička stránky obsahuje textové pole pre zadanie novej úlohy
- Tlačidlo pre pridanie novej úlohy je reprezentované ako `<span>` element, atribút `onlick` obsahuje JavaScript kód, ktorý sa zavolá po kliknutí na daný element (volanie funkcie `newToDoItem()` definovanej v súbore `todo.js`)

```
<div class="header">
  <h2>Moje úlohy</h2>
  <input type="text" id="new-task"
         placeholder="Úloha...">
  <span onclick="newToDoItem();" class="add-btn">
    Pridaj
  </span>
</div>
```

# Štruktúra HTML stránky (2)

- Hlavná časť stránky obsahuje zoznam úloh formátovaný ako nečíslovaný zoznam `<ul>`
- Nové úlohy sú reprezentované ako elementy `<li>`, ktoré sú do HTML pridané dynamicky JavaScript kódom
- Po kliknutí je možné označiť úlohu za splnenú zmeneným formátovaním (preškrtnutý text) pridaním triedy CSS `checked`
- Každá položka má priradené tlačidlo pre odstránenie reprezentované ako `<span>` element
- Príklad:

```
<li class="checked">
  Text úlohy <span class="close">x</span>
</li>
```

# CSS

- Súbor CSS definuje triedy pre formátovanie textového poľa a tlačidla pre pridávanie úloh a pre formátovanie jednotlivých položiek
- Pomocou pseudotriedy `:hover` sa mení zobrazenie položiek a tlačidiel ak naň presunie používateľ myš, napr.:

```
.close:hover {
    background-color: red;
    color: white;
}
```

- Farebné striedanie položiek je formátované pseudotriedou `:nth-child(odd)`

# JavaScript

1. Nastavenie funkcie (tzv. *handler-a*), ktorú prehliadač zavolá po kliknutí na položky zoznamu (funkcia prepína preškrtnutie textu položky)
2. `function newToDoItem()` – funkcia volaná po kliknutí na tlačidlo "Pridaj"
3. `function addToDoItem(text, checked)` – funkcia, ktorá vytvorí element novej položky s textom `text` a stavom `checked` (ak `checked = true`, položka bude preškrtnutá) vrátane tlačidla "x" na odstránenie položky