Javascript: funkce, objekty, prototypy, HTML DOM, kontext a rozsah platnosti proměnné, datové typy

Javascript

- Je to multiplatformí, OOP, událostmi řízený skriptovací jazyk
- Syntaxe patří do rodiny jazyků C/C++/Java
- Standardizuje ho společnost ECMA
- Javascript může pracovat na straně serveru i na straně klienta
- Na straně serveru je hlavně vhodný pro real-time aplikace
- V dnešní době díky mnoha frameworkům lze využit i na psaní desktop aplikací a mobilních aplikací

Funkce

- Funkce je blok kódu určený k vykonávání nějaké určité úlohy
- Funkce se vykoná když jí někde v kódu vyvoláme
- Deklarace funkce:
  - function keyword
  - jméno funkce
  - parametry
  - tvrzení funkce (function statement)
- V javascriptu jsou dvě metody zapisování funkcí buď pomocí klasické funkce nebo pomocí arrow funkce
- Klasická funkce:

- Arrow funkce:

Objekty

- Class je šablona pro vytváření objektů
- Zapouzdří data a metody
- V javascriptu jdou definovat pomocí dvou klíčových slov function a class
- Class:

- Function:

```
function Objekt(jmeno) {
    this.jmeno = jmeno;
}

const obj = new Objekt("Jmeno")

console.log(obj)
```

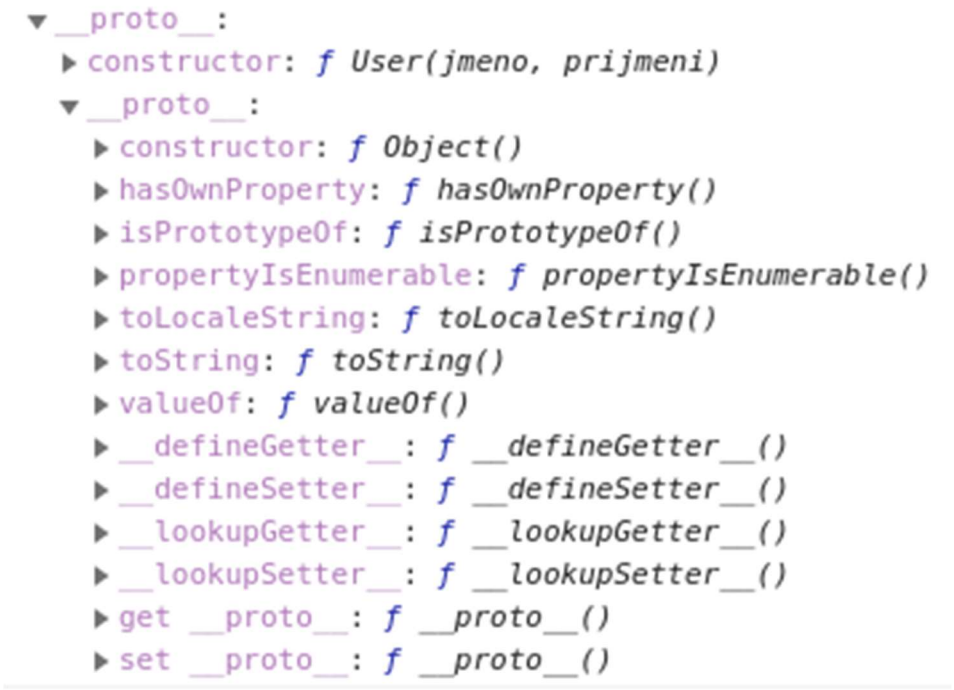
Prototypy

- Prototypy jsou mechanismus díky, kterým JS objekty dědí vlastnosti jeden od druhého
- Každý typ objektu má svůj prototyp
- Link na video pro lepší vysvětlení: https://www.youtube.com/watch?v=4jb4AYEyhRc&ab_channel=TheNetNinja
- Příklad:
    - Vytvoříme objekt User a dáme mu metodu login

```
function User(jmeno, prijmeni) {
    this.jmeno = jmeno;
    this.prijmeni = prijmeni;
    this.online = false;

    this.login = function() {
        this.online = true;
        console.log("uzivatel je prihlasen");
    };
}
```

    - Nyní když uděláme instanci této třídy a console logneme jí zjistíme, že v proměnné __proto__ se nenachází naše metoda



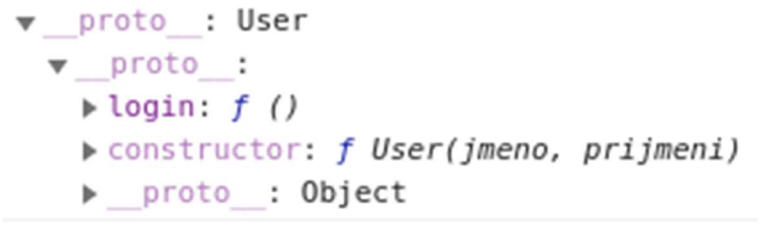
- Nyní vytvoříme novou třídu Admin

```
function Admin( ... args) {
    User.apply(this, args)
    this.role = "admin"
}
```

- Tato třída dědí atributy třídy User
- Teď zdědíme metodu z prototypu User

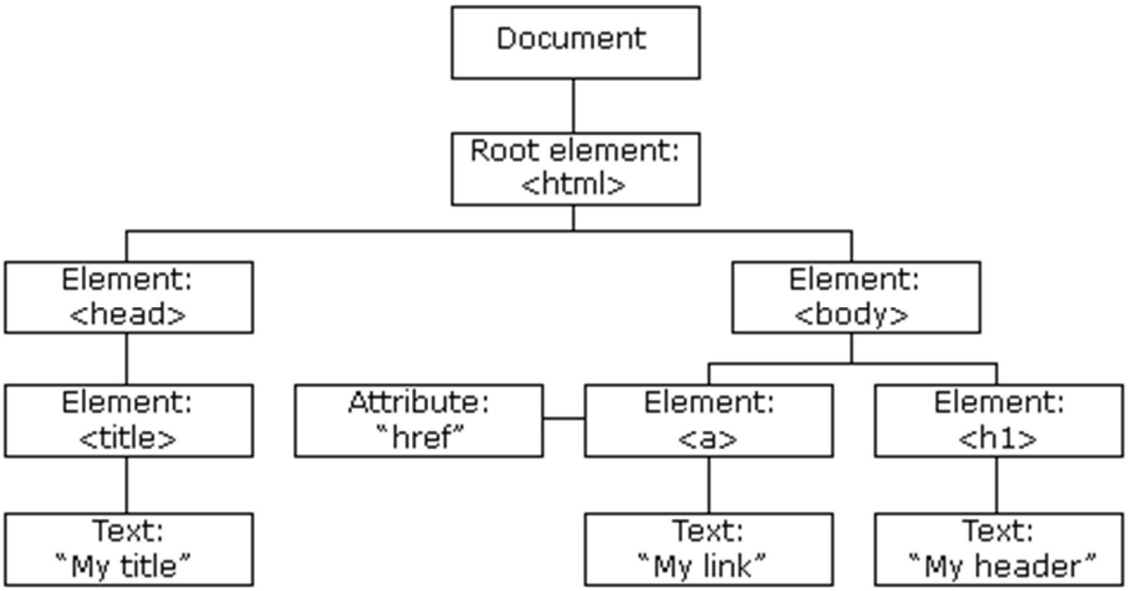
```
Admin.prototype = Object.create(User.prototype)
```

- Nyní když console logneme instanci admin uvidíme, že pomocí prototypu zdědila metodu třídy User



HTML DOM

- Document Object Model
- Je to programové rozhraní (API) pro HTML a XML dokumenty
- Reprezentuje stránku jako stromovou strukturu aby bylo možné měnit strukturu dokumentu, style a kontextu
- Díky tomu se mohou programovací jazyky jako třeba JS připojit ke stránce



- Všechny hodnoty jsou uloženy v objektu document
- Následně pomocí tohoto objektu a jeho metod můžeme různě upravovat HTML stránky
- Jedna z používaných metod je getElementById
- Často používaný atribut documentu je hodnota innerHTML ta umožňuje změny kontextu nějakého elementu

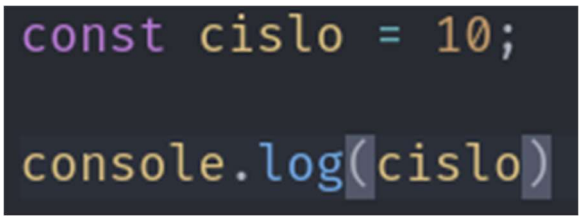
Kontext a rozsah platnosti proměnné

- U JS, které pracuje na straně klienta je globální scope document objekt
  - Každá proměnná, která je inicializovaná v nějaké funkce patří do local scopu
  - Proměnná vytvořená v global scopu lze použít po celém souboru
  - Proměnná v local scopu lze používat pouze v tom scopu kdy byla inicializovaná

- Příklad 1:

  - Proměnná globalVar půjde lognout, protože je v globálním scopu
  - Proměnná localVar nepůjde lognout protože patří pouze do lokálního scopu
  - Aby bylo možné localVar lognout musíme zavolat funkci, ve které je tato proměna a následně bude lognuta
- Příklad 2:
  - Globální scope funugje napříč soubory v klien-side JS
  - To znamená že pokud vytvoříme proměnou globalVar v souboru index.js a následně vytvoříme soubor index2.js tato proměnná půjde stále zavolat

- U JS, které pracuje na straně serveru funguje globální scope pouze v jednom modulu
  - Jeden modul je jeden soubor, takže pokud bychom proměnnou z jednoho souboru chtěli použít v jiném tak už to nejde
  - Příklad:



    - Vytvořením této proměnné v souboru scope-a.js a její následného lognutí funguje bez problému
    - Pokud ale vytvoříme soubor scope-b.js a budem tuto proměnou chtít lognout tak už to nefunguje, protože tato proměnná je pouze v modulu scope-a.js
    - Aby bylo možné tuto proměnou používat i v jiném module musíme jí jí exportovat pomocí příkazu **module.exports.cislo = cislo;**

- A následně pomocí příkaz **const scopea = require("./scope-a")** naimportovat

Datové typy

- Primitivní a ne-primitivní
- Ne-primitivní datové typy mohou ukládat kolekce data zatímco primitivní datové typy mohou skladovat pouze jednotlivá data
- Primitivní:
  - String
  - Number
  - BigInt
  - Boolean
  - undefined
  - null
  - Symbol
    - https://www.youtube.com/watch?v=4J5hnOCj69w&ab_channel=ColtSteele
- Ne-primitivní:
  - Objekt
    - Toto je speciální strukturální typ pomocí, kterého lze následně konstruovat:
      - Array
      - Map
      - Set
- Javascript je volně psaný dynamický jazyk, to znamená že proměnným v JS není přímo přidělován určitý datový typ
- Proměnné může být přidělen nějaký datový typ a následně té samé proměné přidělit jiný datový typ