JavaScript

- slabo tipiziran sa elementima objektno orjentisanih i funkcionalnih paradigmi
- promenljive se deklarisu pomocu **var** kljucne reci i mogu biti lokalne i globalne
-

Primitivni tipovi:

brojevi: sve vrednosti su predstavljene kao realni brojevi (64 bita, IEEE 754 standard)
u slucaju prekoracenja (overflow) generise se +/- **Infinity**
u slucaju potkoracenja (underflow) generise se 0/"negative zero"
deljenje nulom ne rezultira greskom osim u slucaju 0/0 – tada se generise **NaN**
Infinity i **NaN** su read-only globalne promenljive

stringovi: nepromenljive sekvence 16bitnih vrednosti (obicno Unicode karaktera)
indeksacija pocinje od 0
mogu se koristiti bilo dvostruki bilo jednostruki navodnici
sa ECMAScript 5 moze im se pristupati I preko notacije nizova
npr. s.charAt(10) i s[10]

logicki tip (boolean): true/false
zbog konverzije tipova *false* je isto sto I undefined, null, 0, -0, NaN, ""

Specijalne vrednosti:

null – predstavlja nedostajucu vrednost (language keyword)

undefined – predstavlja nedostajucu vrednost (read-only global variable)
dobija se kao rezultat pristupa svojstvima ili elementima niza koji ne postoje ili kao rezultat funkcija koje nemaju return

ni null ni undefined nemaju svojstava I metoda pa zato treba biti oprezan prilikom pristupa

Objekti: sve preostalo

- neuredjena kolekcija svojstava tj. imenovanih vrednosti
  objekti su dinamicke prirode i promenljivi/prosirivi
  njima se manipulise po referenci, a ne po vrednosti
  mogu se definisati sopstvene klase

- razlikujemo nesto sto se zove **globalni objekat**: svaki put kada se interpreter pokrene ili se ucita nova strana u pregledacu, kreira se ovaj globalni objekat I na nivou njega se definisu I inicijalizuju svojstva tipa undefined, NaN, globalne funkcije tipa isNaN(), parseInt(), konstruktori kao sto su Date(), Array(), String()..., globalni objekti kao sto su Math, JSON.. **this** je rezervisana rec kojom se u globalnom opsegu moze referisati na ovaj objekat; na nivou pregledaca to je Window objekat koji se moze referisati sa **window**

- I medju podrzanim objektima: **funkcije** (objekti koji sadrze kod koji treba izvrsiti), nizove (uredjena kolekcija numerisanih vrednosti) koji su imenovani sa **Array**, **Date** objekat za rad sa datumima I vremenom, **RegExp** objekat za rad sa regularnim izrazima i **Error** objekat koji predstavlja sintaksne greske i greske u izvrsavanju

Kreiranje objekata:

1. koriscenje literala
   var point={x:3, y:4};

2. koriscenjem **new** kljucne reci
   var a=new Array();

3. koriscenjem Object.create() funkcije
   var o=Object.create({x:3, y:4});

   svojstva imaju svoja **imena** i svoje **vrednosti**
   pored ovoga svako svojstvo ima I dodatne atribute kao sto su
   writable – da li se vrednost svojstva moze menjati
   enumerable – da li se svojstvo pojavljuje u for/in petlji
   configurable – da li se svojstvo moze obrisati I da li se njegovi atributi mogu menjati

   npr. sa Object.**getOwnPropertyDescriptor**(point, "x) se dobijaju svi atributi pridruzena ovom svojstvu

   pored svojstava svaki objekat ima i tri atributa:
   prototype – referenca na objekat cija se svojstva nasledjuju
   class – string koji kategorise tip objekata
   extensible – svojstvo koje odredjuje da li je objekat prosiriv

Dohvatanje svojstava:

. notacija:
   npr. point.x
   u ovoj notaciji svojstvo mora biti identifikator pa nedostaje dinamicnosti

[] notacija:
   npr. point[x]
   svojstvo moze biti odredjeno u toku ozvrsavanja skripta

pristup nepostojecem svojstvu rezultira sa *undefined*

Ispitivanje da li neko svojstvo postoji na nivou nekog objekta:

operator **in** - proveravaju se i licna i nasledjena svojstva
   "x" in point
   "toString" in o

Object.**hasOwnProperty** – za proveru samo licnih svojstava

Brisanje svojstava:

delete operator
npr. delete point.x

Prototipi:

o.x – ako objekat o ima svojstvo x rezultat je njegova vrednost
ako objekat o nema svojstvo x, razmatra se njegov prototip P(o)
ako P(o) ima svojstvo x, onda je rezultat njegova vrednost
ako P(o) nema svojstvo x, razmatra se njegov protptip P(P(o))
ovaj algoritam se nastavlja sve dok se ne pronadje svojstvo o ili dok ne nestane prototipova u lancu – Object nema prototip

o.x=4 – ako objekat o ima svojstvo x, menja se njegova vrednost
ako objekat o nema svojstvo x, prosiruje se I sada x postaje njegovo svojstvo
ako je u prototipu objekta o bilo nasledjeno x – onda ono postaje zaklonjeno tekucim svojstvom x (izuzetak ako je x nasledjeno svojstvo sa osobinom read-only)

=> nasledjivanje se razmatra prilikom citanja vrednosti, ali ne i kada se pridruzuje vrednost svojstvu

Pojam klase:
ako dva objekta nasledjuju svojstva istog prototipa kazemo da su instance iste klase
klase su dinamicke i prosirive

Nacin da se utvrdi tip klase:
instanceof operator :
o instanceof c ce dati *true* ko o nasledjuje prototip od c (ne obavezno direktno)
constructor svojstvo
if(o.constructor==f)....

Nasledjivanje:

ako klasa **B** nasledjuje klasu **A** onda onda **B.prototype** mora da bude naslednik **A.prototype**

```
function inherit(p) {
if (p == null) throw TypeError(); // p must be a non-null object
if (Object.create)
        // If Object.create() is defined then just use it.
        return Object.create(p);
// Otherwise do some more type checking
var t = typeof p;
if (t !== "object" && t !== "function") throw TypeError();
// Define a dummy constructor function.
function f() {};

f.prototype = p;
// Set its prototype property to p.
// Use f() to create an "heir" of p.
return new f();
}
```

prevencija nasledjivanja:

Object.preventExtensions(obj) zabranjuje dodavanje novih svojstava

Object.seal() zabranjuje dodavanje novih svojstava ali i brisanje postojecih

npr. Object.seal(obj.prototype)

Overload metoda:

funkcije koje se obicno preklapaju su:

toString(), toJSON(), valueOf() - za numericke konverzije

ostalo preklapanje zavisi od broja argumenata ali je moguce

Pristup roditeljskim metodama i svojstvima:

direktno preko pristupa roditeljskom prototipu