Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

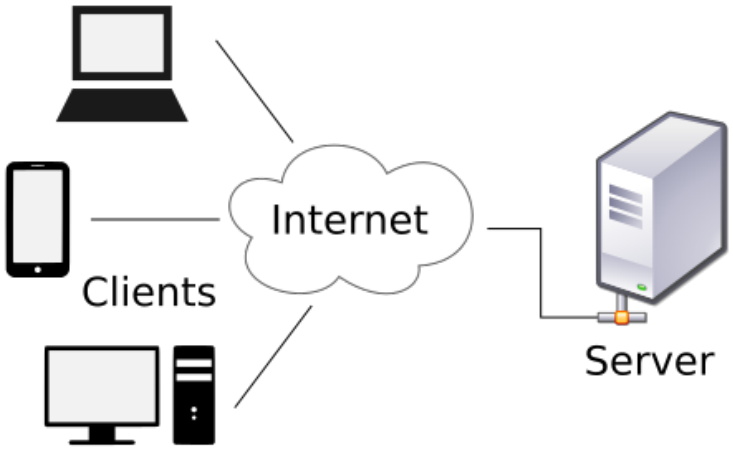
RAČUNARSKI PRAKTIKUM II

Predavanje 08 - Uvod u JavaScript

8. svibnja 2024.

Sastavio: Zvonimir Bujanović

Skriptiranje na strani klijenta



Aktivne web-stranice = skriptiranje na strani klijenta:

- Server pošalje klijentu tražene HTML, CSS i ostale datoteke. Te datoteke sadrže i neki program.
- Web-browser na klijentu prikaže “downloadane” datoteke, te izvršava program unutar web-browsera na klijentskom računalu.
- Zbog interakcije programa s korisnikom, korisnik ima dojam da je web-stranica “aktivna”.

Skriptiranje na strani klijenta

Osnovna varijanta:

- Program se u potpunosti izvršava na klijentu.
- Program nakon “downloadanja” nema više nikakvu interakciju sa serverom.

Složenija varijanta:

- Program na osnovu korisničkih akcija (klik/unos teksta) ima daljnju kombinaciju sa serverom.
- Obično se ta komunikacija “skriva” od korisnika (nema ponovnog učitavanja stranice, nego se odvija u pozadini).

Tipični programski jezici za klijentsko skriptiranje:

- JavaScript, ~~ActionScript (Adobe Flash), Java appleti~~

Uoči: svaki klijent treba imati podršku za programski jezik!

- Svi moderni browseri podržavaju JavaScript *out-of-the-box*.
- ~~Adobe Flash i Javu najčešće treba manualno instalirati.~~

Programski jezik JavaScript

- Višeplatformski, objektno orijentirani skriptni jezik.
- Jezgra jezika (sintaksa, tipovi, ključne riječi, operatori...) je tzv. ECMAScript, standard ECMA-262
  - Aktualna verzija je ECMAScript 2023.
  - Na ovom linku možete vidjeti podršku u browserima.
  - Jezik se vrlo brzo razvija i ima ekstremno puno biblioteka.
- Jezgra jezika je proširena objektima koji pružaju kontrolu nad:
  - web-browserom – Browser Object Model (BOM),
  - učitanim HTML dokumentom – Document Object Model (DOM).
- Na primjer:
  - Aplikacija može dodavati ili micati elemente na web-stranici, mijenjati njihov stil.
  - Aplikacija može odgovarati na događaje poput klika mišem, unosa podataka u formu ili navigacije unutar web-stranice.

Prva implementacija: Netscape Navigator, 1996. godine.
Dolaskom HTML5, JavaScript doživljava veliku ekspanziju.

Literatura

Knjige:

- M. Frisbie – *Professional JavaScript for Web Developers*, 2019.
- E. Freeman, E. Robson – *Head First HTML5 Programming*, 2011.
- E. Freeman, E. Robson – *Head First JavaScript Programming*, 2014.

Web-resursi:

- Mozilla Developer Network – JavaScript
- JavaScript.info
- Eloquent JavaScript
- w3schools – JavaScript Tutorial

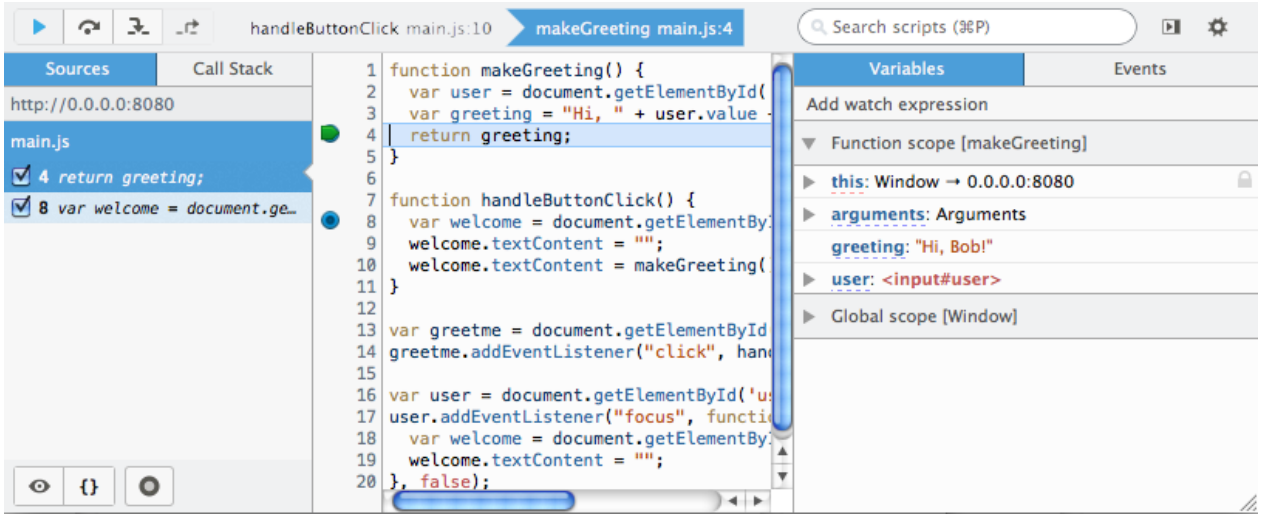
Programski alati

Mozilla Firefox

- F12 ili Tools → Browser Tools → Web Developer Tools

Editori:

- Visual Studio Code
- Brackets
- NetBeans
- SublimeText
- WebStorm



Uključivanje JavaScript koda u HTML dokumente

Direktno unutar HTML koda:

```
<script>
function sayHi() {
    alert("Hi!");
}
</script>
```

Link iz HTML dokumenta na eksterni file:

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

- Eksterni file se može nalaziti na drugoj domeni.
- Ranije je bilo uobičajeno stavljati `<script>` unutar `<head>`.
- Sada je običaj stavljati `<script>` na kraj `<body>`, kad se cijela web-stranica već učita i prikaže (no postoje iznimke).
- Kod se izvršava redom kojim je naveden/uključen u HTML-u.

Osnovna sintaksa

Sintaksa vrlo slična C-u:

- case-sensitive
- komentari `//...` i `/* ... */`
- naredbe završavaju sa `;`
- blokovi naredbi unutar `{...}`

JavaScript je *duck*, slabo i dinamički tipiziran jezik.
Varijable se deklariraju ključnom riječi `var`, `let` ili `const`:

```
let a;
console.log("The value of a is " + a); // "...undefined"
console.log("The value of b is " + b); // ReferenceError exceptio
```

- Doseg varijable je funkcijski ili globalni za `var` (nije blokovski!).
- Doseg varijable je blokovski za `let` i `const` (to želimo).
- Varijable mogu mijenjati tip tijekom izvršavanja (nije preporučeno).
- `typeof(x)` vraća string s tipom varijable `x`.

Tipovi podataka

Postoji 6 *primitivnih* tipova podataka:
`undefined`, `null`, `Boolean`, `Number`, `String`, `Symbol`.
Primitivni tipovi se alociraju na stogu.

- undefined
  - sadrži samo jednu vrijednost, `undefined`
  - to je tip deklarirane neinicijalizirane varijable

```
let message;
console.log( message === undefined ); // "true"
console.log( typeof( message ) ); // "undefined"
```

- null
  - sadrži samo jednu vrijednost, `null`
  - koristimo ga kad ne postoji smisleni objekt koji bismo pridružili varijabli

```
let obj = null;
console.log( typeof( obj ) ); // "object"
```

- Boolean

```
let found = true, lost = false;
```

Drugi tipovi se mogu automatski konvertirati u Boolean:

```
let hello = "hello";
if( hello ) console.log( "String hello je true" );
```

- Number

Svi brojevi (i cijeli i realni) se reprezentiraju po IEEE-754, tj. kao C-ov `double`.

```
let a = 5, b = 3.14, c = 1.5e2, d = 012, e = 0x1f;
```

Konverzije iz drugih tipova su jednostavne (vidi i `parseInt`):

```
let x = true, y = "3.124";
let x_num = Number( x ), y_num = Number( y );
```

- String
  - String-konstante mogu biti unutar `'...'` ili `"..."`.
  - Specijalni znakovi: `\n`, `\t`, `\"`, `\unnnn`.
  - Pristup znakovima unutar stringa pomoću `[...]`.
  - Znakove unutar stringa ne možemo mijenjati (*immutable*).
  - Usporedba po abecedi sa `<`.
  - Konkatenacija sa `+`.
  - Jedino svojstvo klase `String`: `length`.
  - Metode klase `String`: `indexOf`, `match`, `substr`, ...
  - Drugi tipovi se u string konvertiraju metodom `toString`.

```
let str = "hello ", x = 3.14;
str = str + "world";
console.log( str ); // "hello world"
console.log( str.length ); // 11
console.log( str.substr( 1, 4 ) ); // "ello"
str = str + x.toString(); // ili str=str+String(x);
console.log( str ); // "hello world3.14"
```

- String
  - String-konstante mogu biti i unutar `...` ("izvrnuti" navodnici).
  - Takve konstante mogu zauzimati nekoliko redaka.
  - Unutar njih je moguće "ispisati" varijablu ili rezultat izraza.

```
1 let x = 123, y = 456;
2 let str = `Zbroj brojeva
3     ${x} i ${y} je
4     ${x+y}`;
5 console.log( str );
6 // ispis je u 3 reda, s razmacima na početku 2. i 3. reda
```

- Object
  Najjednostavniji tip koji se čuva na heap-u (*reference type*).
  Predstavlja kolekciju podataka i funkcija (slično klasi).

```
let osoba = new Object(); // osoba je "pointer"!
osoba.ime = "Pero"; osoba.starost = 20;

let x = osoba;
x.ime = "Ana"; console.log( osoba.ime ); // "Ana"
```

Često se koristi ovakav ekvivalentan zapis:

```
let osoba = {
    ime: "Pero",
    starost: 20
};
```

Pazi:

```
let x = 5, y = new Number( 5 );
console.log(typeof(x) + " " + typeof(y)); // "number object"
```

- Funkcije
  - Sintaksa relativno slična C/PHP-u:

```
function square(number) { return number * number; }
let x = square( 5 ); // x = 25
let f = square;
console.log( f(3) ); // 9
```

  - Nema overloadanja.
  - Vrlo se često koriste anonimne funkcije (lambda izrazi):

```
let f = function(x) { return x*x; };
let kvadrat = ( x => { return x*x; } );
let zbroj = ( (x, y) => x+y ); // "arrow function"
```

  - Argumenti se prenose po vrijednosti (pazi, `Object` je "pointer").

- Funkcije
  - Funkcije mogu primati po volji mnogo parametara → polje `arguments`

```
function f() { console.log( arguments[1] ); }
f( 3, "nesto", 123 ); // "nesto"
```

  - No tada se obično koristi fleksibilnija sintaksa za ostale parametre pomoću tri točke (*"rest parameters"*):

```
function f(prvi, drugi, ...ostali) {
    console.log( ostali[1] );
}
f( "a", "b", "hm", "nesto", "xyz" ); // "nesto"
```

  - Parametre koji su u polju možemo "raspakirati" pomoću tri točke prilikom poziva funkcije (*"spread syntax"*):

```
function f(a, b, c) { console.log( b ); }
let parametri = [1, 2, 3];
f( ...parametri ); // "2"
```

Tipovi podataka

- Array
  - Niz podataka ne nužno istog tipa!
  - Pristup elementima sa `[...]`.
  - Indexi su nenegativni cijeli brojevi.
  - Automatsko produljivanje/skraćivanje po potrebi.
  - Svojstvo `length`; nije read-only(!)
  - Metode: `push`, `pop`, `reverse`, `sort`, `indexOf`, `map`, `forEach` ...
  - "Raspakiravanje" pomoću tri točke radi i kod inicijalizacije polja.

```
let x = new Array();
x.push( 7 ); x.push( 3 ); x[2] = 5; // x = [7, 3, 5]
x.sort( function(a, b) {
    return (a<b) ? -1 : ((a>b) ? 1 : 0);
} );
console.log( x.toString() ); // "3,5,7"
x.length = 5; // x = [3, 5, 7, undefined, undefined]

let y = ["red", "blue"], z = [1, 2, 3];
let w = [...y, 5, ...z]; // ["red", "blue", 5, 1, 2, 3];
```

Operatori, kontrola toka, petlje

- Operatori kao u C-u; iznimka `==` i `===`.
- `a==b` radi konverziju ako `a` i `b` nisu istog tipa.
- `a===b` vraća `false` ako `a` i `b` nisu istog tipa.
- Slično `!=` i `!==`.
- `if`, `switch`, `while`, `do...while`, `for`, `break`, `continue`, `throw/try/catch` identični kao u C-u.
- for ... in
  Iteracija po svim svojstvima objekta.

```
let x = {
    ime: "Pero",
    starost: 20
};
for( let i in x )
    console.log( i + "->" + x[i] );
// "ime->Pero", "starost->20"
```

Zadatak 1

Napišite program koji na web-stranicu ispisuje prvih 100 prirodnih brojeva. Ispišite te brojeve i u konzolu.

Što se dogodi ako probamo ispisati prvih 100000 brojeva?

Upute:

- Poziv `document.write(str);` će na to mjestu u HTML-u ispisati string `str`.
- Poziv `console.log(str);` ispisuje string `str` u konzolu. Ovo je korisno za debuggiranje.
- Ako `x` nije string, onda `document.write(x);` automatski poziva `x.toString()` prije ispisa.

Napomena: `document.write()` ćemo koristiti samo u ovoj početnoj fazi. U idućem predavanju ćemo vidjeti bolje alternative.

Zadatak 2

Napišite program koji na web-stranicu ispisuje tablicu množenja za prvih N prirodnih brojeva. Broj N učitajte pomoću “skočnog okvira” (engl. *popup*).

Uputa:

- Postoje 3 vrste “skočnog okvira”:
  - `alert("Poruka");`
    Okvir u kojem piše "Poruka". Jedino možemo kliknuti na OK.
  - `let ret = prompt("Što želiš?");`
    Okvir u kojem piše "Što želiš?". String koji utipkamo će biti spremljen u varijablu `ret`.
  - `let ret = confirm("Da ili ne?");`
    Okvir u kojem piše "Da ili ne?". Možemo kliknuti na "OK" ili "Cancel"; boolean će biti spremljen u varijablu `ret`.

Zadatak 3

Pomoću ugrađenog objekta Date, dohvatite trenutno vrijeme (na klijetnovom računalu!). Ispišite trenutni datum (s mjesecima na hrvatskom!) i trenutno vrijeme, te:

- Ako je trenutno manje od 12h, ispišite poruku "Dobro jutro!".
- Ako je između 12h i 18h, ispišite "Dobar dan!".
- Ako je više od 18h, ispišite "Dobra večer!".

Uputa:

- `let t = new Date();`
  U varijablu `t` sprema trenutno vrijeme. Nakon toga je dostupan cijeli niz funkcija poput `t.getDate()`, `t.getMinutes()`.

Zadatak 4

Koristeći “skočni okvir”, učitajte neki string, te ispišite sve njegove podstringove.

Na primjer, za učitani string ”abc”, treba ispisati:

- ””, ”a”, ”b”, ”c”, ”ab”, ”ac”, ”bc”, ”abc”

(ne nužno tim redom).

Regularni izrazi

- U JavaScriptu, regularne izraze možemo deklarirati na dva ekvivalentna načina:
  - `let re = /ab+c/;`
  - `let re = new RegExp("ab+c");`
- U klasi RegExp postoje metode:
  - `exec`, `test`, `match`, `search`, `replace`, `split`.

```
let str = "110010001";
let re = /^(0|(1[01]*))$/;

if( re.test( str ) )
    document.write( str + " je binarni broj " );
else
    document.write( str + " nije binarni broj " );
```

Zadatak 5

Koristeći "skočni okvir", učitavajte datum sve dok se ispravno ne unese u formatu `dd/mm/yyyy`. Nakon toga, ispišite koji je to dan u tjednu.

Uputa:

- Dan u tjednu možete dohvatiti pomoću metode `getDay()` klase `Date`. Datum koji odgovara učitanom možete napraviti ovako (oprez, za siječanj je month=0!):
  ```
  let d = new Date( year, month, day );
  ```
- Dan, mjesec i godinu iz učitanog stringa možete dohvatiti i pomoću metode exec klase RegExp.

Objekti mogu imati i svojstva i metode kao članove:

```
let osoba = {
    ime: "Pero",
    starost: 20,

    kaziBok: function() { console.log("Bok!"); }
};
osoba.prezime = "Perić"; osoba.kaziBok();
```

Kako napraviti puno objekata istog tipa?
JavaScript:

- Tek u ECMAScript 2015 su uvedeni klasični elementi OOP poput klasa, konstruktora, destuktora, nasljeđivanja i slično.
- JavaScript koristi tzv. OOP baziran na prototipima.
- OOP elementi se ostvaruju procesom “dekoriranja”, odnosno proširivanja postojećih objekata koji služe kao prototipi.
- Ovo je tzv. *bezklasno* ili *prototype-oriented* programiranje.

Objektno orijentirano programiranje: Constructor pattern

Stvaranje puno objekata istog tipa → Constructor pattern:

```
function Osoba( ime, starost ) {
    // this se odnosi na objekt koji se konstruira sa new.
    this.ime = ime;
    this.starost = starost;
    this.kaziBok = kaziBok;
}

function kaziBok() { console.log( "Bok!" ); }

let pero = new Osoba( "Pero", 20 );
let ana = new Osoba( "Ana", 19 );

console.log( pero instanceof Osoba ); // "true"
console.log( ana instanceof Object ); // "true"
console.log( pero.kaziBok === ana.kaziBok ); // "true"
```

Problem ovog pristupa je “zagađivanje” koda globalnim funkcijama.

Objektno orijentirano programiranje: prototype

Svaka funkcija u JavaScriptu ima svojstvo `prototype`.
Ono propisuje kako se kreira objekt kad se pozove `new` na toj funkciji.

```
function Osoba() {}

Osoba.prototype.ime = "Pero";
Osoba.prototype.starost = 20;
Osoba.prototype.kaziBok = function() { console.log("Bok!"); };

let pero = new Osoba(), ana = new Osoba();
ana.ime = "Ana";

console.log( [pero.ime, ana.ime] ); // "Pero, Ana"
console.log( pero instanceof Osoba ); // "true"
console.log( ana instanceof Object ); // "true"
console.log( ana.kaziBok === pero.kaziBok ); // true
```

Svojstvo `prototype` sadrži podsvojstvo `constructor`.
Po defaultu, `prototype.constructor` sadrži "pointer" na samu funkciju (`Osoba`).

Objektno orijentirano programiranje: Constructor + prototype

Rješenje problema “zagađivanja” globalnog dosega daje kombinacija Constructor + prototype.

```
function Osoba(ime, starost) {
    this.ime = ime;
    this.starost = starost;
}

Osoba.prototype = {
    constructor: Osoba, // treba jer {} kreira NOVI objekt
    kaziBok: function() { console.log( "Bok!" ); }
};

let pero = new Osoba( "Pero", 20 ),
    ana  = new Osoba( "Ana", 19 );

console.log( [pero.ime, ana.ime] ); // "Pero, Ana"
console.log( pero instanceof Osoba ); // "true"
console.log( ana instanceof Object ); // "true"
console.log( ana.kaziBok === pero.kaziBok ); // true
```

Alternativno, konstruktor možemo napisati ovako:

```
function Osoba(ime, starost) {
    this.ime = ime;
    this.starost = starost;

    if( typeof(this.kaziBok) != "function" )
        Osoba.prototype.kaziBok = function() {
            console.log( "Bok!" );
        };
}

let pero = new Osoba( "Pero", 20 ),
    ana  = new Osoba( "Ana", 19 );

console.log( [pero.ime, ana.ime] ); // "Pero, Ana"
console.log( pero instanceof Osoba ); // "true"
console.log( ana instanceof Object ); // "true"
console.log( ana.kaziBok === pero.kaziBok ); // true

pero.kaziBok();
```

Objektno orijentirano programiranje: nasljeđivanje

Donji kod implementira tzv. *parazitsko kombinirano* nasljeđivanje. Za detalje o OOP u JavaScriptu vidi i Mozilla Developer Network.

```
// .. definiramo tip Osoba kao prije
function Student( ime, starost, JMBAG ) {
    Osoba.call( this, ime, starost ); // konstr. bazne klase
    this.JMBAG = JMBAG;               // dodaj novo svojstvo
}

Student.prototype = Object.create(Osoba.prototype);
Student.prototype.constructor = Student;

Student.prototype.dajJMBAG = function() {
    return this.JMBAG; // nova metoda u klasi Student
}

let pero = new Student( "Pero", 20, "12345" );
pero.kaziBok();
console.log( pero.dajJMBAG() );
```

Klasičnija notacija OOP u JavaScriptu

ECMAScript 2015 (FF 45+, CH 42+) uvodi standardniju OOP notaciju.
Ovo je samo syntactic sugar za OOP baziran na prototipovima!

```
1  class Osoba {
2      constructor(ime) { this.ime = ime; }
3      kaziBok() { console.log( "Bok!" ); }
4  }
5
6  class Student extends Osoba {
7      constructor(ime, JMBAG) { super(ime); this.JMBAG = JMBAG; }
8      dajJMBAG() { return this.JMBAG; }
9  }
10
11 let pero = new Student("Pero", "12345"), ana = new Osoba("Ana");
12 pero.kaziBok(); console.log( pero.dajJMBAG() );
13
14 console.log( pero instanceof Object );  // "true"
15 console.log( pero instanceof Osoba );   // "true"
16 console.log( pero instanceof Student ); // "true"
17 console.log( ana instanceof Student );  // "false"
```

Zadatak 6

- Napravite hijerarhiju klasa `Cetverokut`, `Pravokutnik`, `Kvadrat`.
- Za svaki lik može se izračunati opseg.
- Za pravokutnike i kvadrate može se izračunati površina.
- Napravite i konverziju svakog od likova u string tako da predefinirate metodu `toString`:

```
Kvadrat.prototype.toString = function()
{
    return "Kvadrat sa stranicom " + this.a;
}

// Sada (uz definiciju konstruktora) radi:
let K = new Kvadrat( 5 );
console.log( "K je " + K );
```

Closure

- Closure je funkcija koja koristi "nezavisnu" varijablu (onu koja nije definirana unutar te funkcije).
- U donjem primjeru, funkcije `povecaj`, `smanji` i `getX` su *closure*.
- Ovakvi trikovi se često koriste u JS; treba biti oprezan jer mogu imati neobične nuspojave.

```
let brojac = function() {
    let x = 0;

    return {
        povecaj: function() { ++x; },
        smanji: function() { --x; },
        getX: function() { return x; }
    };
}

let A = brojac(), B = brojac();
console.log( A.getX() + " " + B.getX() ); // 0 0
A.povecaj(); A.povecaj(); B.smanji();
console.log( A.getX() + " " + B.getX() ); // 2 -1
```