HTML

```
<!DOCTYPE html>
<!-- Prvi HTML5 primer -->
<html>
  <head>
    <meta charset="utf-8" />
    <title>Pozdravljeni</title>
  </head>
  <body>
    <p>Lepo pozdravljeni pri predmetu Spletno programiranje!</p>
    <img
      src="img/Deitel_2011.jpg"
      width="100"
      height="130"
      alt="Internet and World Wide Web: How to program"
    />
    <h1>Naslov 1. ravni</h1>
    <h2>Naslov 2. ravni</h2>
    <h3>Naslov 3. ravni</h3>
    <h4>Naslov 4. ravni</h4>
    <h5>Naslov 5. ravni</h5>
    <h6>Naslov 6. ravni</h6>
    <p><a href="https://eon.tv/">EON TV</a></p>
    <a href="mailto:Dejan.Lavbic@fri.uni-lj.si">elektronsko sporočilo</a>
    <p>
      <del>
        Iz te spletne strani lahko povlečete 3.14 x 10<sup>2</sup>
        podatkov.
      </del>
      Prvi podatek v seriji je x<sub>1</sub>. &frac14; je ena četrtina.
    </p>
    <hr /><!-- vodoravna črta -->
    <ul>
      <!-- neurejeni seznam, napisane samo pikce -->
      <li>spoznaš lahko ljudi iz različnih držav,</li>
      <!-- element seznama -->
      <li>
        dostop do gradiva, ko postane javno dostopno:
        <ul>
          <li>igre,</li>
          <li>
            aplikacije,
            <ol>
              <!-- urejen seznam, namesto pikc so številke -->
              <li>za podjetja,</li>
```

```
            <li>za zasebne uporabnike,</li>
          </ol>
        </li>
      </ul>
    </li>
  </ul>
  <table border="1">
    <caption>
      <strong>Cena sadja</strong>
    </caption>
    <thead>
      <tr>
        <th colspan="5">Sadje</th>
        <!-- th je samo za glavo in nogo, odebeljeno -->
        <!-- element zaseda 5 stolpcev -->
        <th rowspan="2">Cena</th>
        <!-- element zaseda dve vrstici -->
      </tr>
    </thead>
    <tfoot>
      <tr>
        <th>Skupaj</th>
        <th>3,00 €</th>
      </tr>
    </tfoot>
    <tbody>
      <tr>
        <td>banana</td>
        <td>1,00 €</td>
      </tr>
    </tbody>
  </table>
 </body>
</html>
```

- znak IN: &
- opuščaj: '
- večje: >
- manjše: <
- dvojni narekovaj: "
- presledek:
- avtorska pravica: ©
- pomišljaj: —
- vezaj: –

JavaScript

```
console.log("Gremo na lepše"); //nizi lahko v enojnih ali dvojnih oklepajih
console.log(3.14); //decimalke ločene s piko
console.log(null || "uporabnik"); //če je vrednost na desni strani vrne true, sicer false
// var - znotraj funkcije
// let - znotraj bloka
// const - konstantna vrednost
console.log("Danes je temperatura", temperatura);
console.log(ocenaStudenta >= 49.5 ? "Predmet ste opravili!" : "Predmeta niste opravili!");
// == - enaka vrednost
// === - enaka vrednost in enak tip
```

```
switch(stPik) {
  case 1:
    console.log(stPik + " pika");
    break;
  case 2:
    console.log(stPik + " piki");
    break;
  case 3:
    //izpise se default
  case 4:
    console.log(stPik + " pike");
    break;
  default:
    console.log(stPik + " pik");
    break;
}
```

```
const potenca = function (osnova, eksponent) {
  var rezultat = 1;
  for (var stevec = 0; stevec < eksponent; stevec++) rezultat *= osnova;
  return rezultat;
};
potenca();

function bla() {
  return "bla";
}

// ne rabimo nujno podati vseh parametrov, vendar je potem prevzeta vrednost spremenljivke u
```

```
var seznamStevil = [3, 4, 5, 7, 11, 18];
var dan1 = {
  veverica: false,
  dogodki: ["delo", "drevo", "pica", "tek", "TV"]
```

```
};
console.log(typeof dan1);

var seznam = {};
seznam.prvi = 1;
seznam.drugi = 2;
console.log(seznam);
// output: {prvi: 1, drugi: 2}

var vrednosti = [2, 4, 6, 8, 10, 12, 14];
vrednosti.push(16);
// Doda element na konec in vrne novo dolžino
vrednosti.pop();
// Izbriše zadnji element in ga vrne
vrednosti.shift();
// Izbriše prvi element in ga vrne
vrednosti.unshift(2);
// Doda element na začetek in vrne novo dolžino
console.log(vrednosti.indexOf(10));
// Vrne indeks prve pojavitve elementa v polju
console.log(vrednosti.slice(2,5));
// Vrne vmesne elemente v obliki polja (brez petega indeksa)

console.log("posel".slice(1,5));
// Vrne podniz od začetnega do končnega indeksa
console.log("posel".indexOf("e"));
// Vrne indeks izbranega podniza
console.log("  posel \n ".trim());
// Odstrani nepotrebne presledke
console.log("posel".length);
// Vrne dolžino niza
console.log("posel".charAt(0));
// Vrne črko na indeksu
console.log("posel"[0]);
// Vrne črko na indeksu

function max(...stevila) {
  var rezultat = -Infinity;
  for (var stevilo of stevila) {
    if (stevilo > rezultat)
    rezultat = stevilo;
  }
  return rezultat;
}
console.log(max(13, 8, -22, 38, 7));

var niz = JSON.stringify({ime: "Dejan", rojen: 1980});
console.log(niz);
```

```
//izpiše vse kot string, tudi ime in rojen
console.log(JSON.parse(niz).rojen);

function obdelajElement(polje, akcija) {
  for (var i = 0; i < polje.length; i++) akcija(polje[i]);
}
obdelajElement(stevila, console.log);

var vsota = 0;
obdelajElement(stevila, function(stevilo) {
  vsota += stevilo;
});
console.log(vsota);

var ribica = {};
ribica.povej = function(izjava) {
  console.log("Ribica pravi '" + izjava + "'");
};
ribica.povej("Še sem živa.");

function Ribica(tip) {
  this.tip = tip;
}
var crnaRibica = new Ribica("Črna");
```

Regularni izrazi

```
var ri1 = new RegExp("abc");
var ri2 = /abc/;
//dva načina za kreiranje niza
var ri3 = /C\+\+/;
//kej je + rezerviran, moramo dodati \
console.log(/abc/.test("abxde"));
//false, ni ujemanja
console.log(/[0-9]/.test("leta 1980"));
```

- `\d`: številka
- `\w`: alfanumerični znak (majhne, velike črke, številke itd.)
- `\s`: prazen znak (presledek, tab, znak za novo vrstico)
- `\D`: znak, ki ni številka
- `\W`: znak, ki ni alfanumerični
- `\S`: znak, ki ni prazen znak
- `.`: poljuben znak razen prehoda v novo vrstico

```
var riNiBinarno = /[^01]/;
//če obstaja znak, ki je različen od 0 ali 1
console.log(riNiBinarno.test("11001000101101310101"));
var riDatumCasKrajse = /\d{4}-\d{1,2}-\d{1,2} \d{1,2}:\d{2}/;
```

```
console.log("iti".replace(/i/g, "a"));
//vse i-je zamenjamo z a-jem; če želimo samo samo en i napišemo "i"
```

HTML in JavaScript

```
<script>alert("Lepo pozdravljeni pri spletnem programiranju!");</script>
<script src="js/pozdrav.js"></script>
<!-- izpisalo bi alert(...) -->

<button onclick='alert("Lepo pozdravljeni pri spletnem programiranju!");'>
    Ne me pritisniti
</button>

<script>
  var povezava = document.body.getElementsByTagName("a")[0];
  console.log(povezava.href);
</script>

var predmet = document.getElementById("predmet");

var odstavki = document.body.getElementsByTagName("p");
document.body.insertBefore(odstavki[1], odstavki[0]);

<p><button onclick="zamenjajSlike()">Zamenjaj</button></p>
<script>
  function zamenjajSlike() {
    var slike = document.body.getElementsByTagName("img");
    for (var i = slike.length - 1; i >= 0; i--) {
      var slika = slike[i];
      if (slika.alt) {
        var besedilo = document.createTextNode(slika.alt + " :: ");
        slika.parentNode.replaceChild(besedilo, slika);
      }
    }
    var gumb = document.body.getElementsByTagName("button")[0];
    gumb.disabled = "disabled";
  }
</script>

<!-- .appendChild(parameter) : doda element-->
<bla></bla>
<!-- uporabljamo lahko lastne atribute -->
```

Postavitev na strani:

- border
- text-align

- offsetWidth/Height
- clientWidth/Height

```
<p id="odstavek" style="color: purple">Obarvano besedilo</p>
<script>
  var odstavek = document.getElementById("odstavek");
  console.log(odstavek.style.color);
  odstavek.style.color = "red";
  console.log(odstavek.style.color);
</script>

<script>
  addEventListener("click", function () {
    console.log("Kliknil si na dokument.");
  });
</script>

<script>
  var prviGumb = document.querySelector("#prviGumb");
  prviGumb.addEventListener("click", function () {
    console.log("Klik na gumb.");
  });
  var drugiGumb = document.querySelector("#drugiGumb");
  function enkratniKlik() {
    console.log("Enkratni klik na gumb.");
    drugiGumb.removeEventListener("click", enkratniKlik);
  }
  drugiGumb.addEventListener("click", enkratniKlik);
</script>
<!-- # za id, . za class -->

<style>
  #kazalnik_CSS:hover {
    color: blue;
  }
</style>

<!-- "focus" tudi obstaja, ko je nekaj fokusirano pac pri addEventListener()-->

<!-- setTimeout(čas) -->
```

Spletni obrazci in vnosna polja

Type:

- text
- password

- checkbox (več izbir)
- radio (ena izbira)
- file

Atribut disabled lahko dodamo.

```
<form action="akcija.html">
  Uporabniško ime: <input type="text" name="uporabnisko_ime" /><br />
  Geslo: <input type="password" name="geslo" /><br />
  <button type="submit">Prijava</button>
</form>

<textarea>Tam dol na ravnem polju</textarea>

<select multiple id="binarni_vnos">
  <option value="1">0001</option>
  <option value="2">0010</option>
  <option value="4">0100</option>
  <option value="8">1000</option>
</select>

<input type="file" multiple id="vec" />
```

NODE.JS, MVC

Dokument popaci.js:

```
module.exports = function (niz) {
  return niz
    .split("")
    .map(function (znak) {
      return String.fromCharCode(znak.charCodeAt(0) + 5);
    })
    .join("");
};
```

Dokument uporabi_modul.js:

```
var popaci = require("./popaci");
// v 2. indeksu je dejanski argument, ki ga potrebujemo
var argument = process.argv[2];
console.log(popaci(argument));
```

Funkcije lahko zapišemo tudi tako:

```
var figlet = require("figlet");
figlet.text("Spletno programiranje", (napaka, podatki) => {
  if (napaka) {
```

```
    console.error(napaka);
  } else {
    console.log(podatki);
  }
});

// namesto funckija(napaka, podatki)
```

Modul http:

```
var http = require("http");
var streznik = http.createServer((zahteva, odgovor) => {
  odgovor.writeHead(200, { "Content-Type": "text/html" });
  odgovor.write(
    "<h1>Lep pozdrav</h1><p>Zahteval si <code>" + zahteva.url + "</code>.</p>"
  );
  odgovor.end();
});
streznik.listen(process.env.PORT || 8080, () => {
  console.log(`Strežnik je pognan in posluša na ${process.env.PORT || 8080}.`);
});
```

Ko dostopamo do http virov, uporabljamo:

```
var metode = Object.create(null);

metode.DELETE = (pot, odgovori) => {
  fs.stat(pot, function (napaka, podatki) {
    if (napaka && napaka.code == "ENOENT") odgovori(204);
    else if (napaka) odgovori(500, napaka.toString());
    else if (podatki.isDirectory())
      fs.rmdir(pot, odgovoriNapakoAliNic(odgovori));
    else fs.unlink(pot, odgovoriNapakoAliNic(odgovori));
  });
};
```

Uporaba Bootstrapa v aplikaciji:

```
<link rel="icon" type="image/x-icon" href="/favicon.ico" />
<link rel="stylesheet" href="/styles/fa.all.min.css" />
<link rel="stylesheet" href="/styles/bootstrap.sandstone.min.css" />
<link rel="stylesheet" href="/styles/style.css" />
```

PODATKOVNA BAZA IN REST API

MongoDB dokument:

```
{
  "predpona": "gospod",
  "ime": "Janez",
  "priimek": "Kranjski",
  "vzdevek": "stari",
  _id: ObjectId("52279effc62ca8b0c1000007")
}
```

MongoDB shema:

```
{
  predpona: String,
  ime: String,
  priimek: String,
  vzdevek: String
}
```

V .js datoteki uporabljamo zapis:

```
id: { type: Number, required: true, min: 0, max: 5, default: 0}
```

Type je lahko Date, String, Number

```
db.Locations.updateOne({
    name: "Celje - Celjski grad"
  }, {
    $push: {
      comments: {
        _id: ObjectId(),
        avtor: "Vladimir Putin",
        ocena: 2,
        datum: new Date("Jul 30, 2016"),
        besediloKomentarja: "Čisti dolgčas, še kava je zanič. Edina svetla točka je razstava
      }
    }
  })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

HTTP šifre statusa (pri izvedbi metod GET, POST, PUT, DELETE):

- 200: OK (uspešna GET ali PUT zahteva)
- 201: Created (uspešna POST zahteva)
- 204: No content (uspešna DELETE zahteva)

- 400: Bad request (neuspešna GET, POST ali PUT zahteva, zaradi napačne vsebine)
- 401: Unauthorized (zahteva zaščitenega URL naslova z napačnimi prijavnimi podatki)
- 403: Forbidden (zahteva, ki ni dovoljena)
- 404: Not found (neuspešna zahteva, zaradi napačnega parametra v URL naslovu)
- 405: Method not allowed (metoda zahteve ni dovoljena za zahtevan URL naslov)
- 409: Conflict (neuspešna POST zahteva, ko že obstaja objekt z istimi podatki)
- 500: Internal server error (težava s strežnikom ali podatkovno bazo)

Pri opredelitvi API URL naslova se pogosto uporabljajo parametri, s katerimi določimo odvisne dokumente, kot je to v našem primeru komentarjev, ki so odvisni od lokacije. Vključitev parametra v URL naslov je preprosta, in sicer imenu spremenljivke samo dodamo predpono dvopičja :.

MongoDB ima preko dostopa z Mongoose na voljo številne metode za iskanje podatkov, kjer so ključne naslednje:

- **find** se uporablja za splošno iskanje, glede na podan filter iskanja,
- **findById** išče dokument s podanim enoličnim identifikatorjem,
- **findOne** vrne prvi rezultat ujemanja podanega filtra iskanja,
- **geoNear** (preko funkcije aggregate) išče zadetke, ki so geografsko blizu podani zemljepisni širini in dolžini.

TYPESCRIPT IN ANGULAR

Podatkovni tipi v TypeScriptu:

- **string** predstavlja besedilni niz,
- **number** je poljubna številčna vrednost, kjer cela in decimalna števila obravnavamo na enak način,
- **boolean** je logična vrednost (true ali false),
- **object** predstavlja vrednost, ki ni enostaven podatkovni tip,
- **array** predstavlja polje vrednosti danega podatkovnega tipa,
- **enum** nam omogoča dodajanje prijaznih oznak množici numeričnih vrednosti,
- **any** je poljuben podatkovni tip, tako kot je privzeto pri jeziku JavaScript,
- **void** predstavlja pomanjkanje podatkovnega tipa in se običajno uporablja pri funkcijah, ki ne vračajo ničesar.

Angular v html:

```
<body>
  <app-root></app-root>
</body>
```

Prevzeti modul v Angular:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule,
    HttpClientModule,
    RouterModule.forRoot([
      { path: "", component: LocationListComponent },
      { path: "about", component: AboutComponent },
    ])],
  providers: [],
  bootstrap: [AppComponent],

})

@Component({
  selector: "app-root",
  template: `
    <!--The content below is only a placeholder and can be replaced.-->
    <div style="text-align:center" class="content">
      <h1>Welcome to {{ title }}!</h1>
      <span style="display: block">{{ title }} app is running!</span>
      ...
    </div>
    ...
  `,
  templateUrl: "framework.component.html",
  styles: [],
})

export class AppComponent {
  title = "Demo";
}

export class AppModule { }
```

DAPP, SOLIDITY, TRUFFLE, WEB3

Struktura pogodbe:

```
pragma solidity ^0.8.17;
```

```
contract AuthorizedToken {
  enum UserType {
    TokenHolder,
    Admin,
    Owner
  }
  struct AccountInfo {
    address account;
    string firstName;
    string lastName;
    UserType userType;
  }
  mapping(address => uint256) public tokenBalance;
  mapping(address => AccountInfo) public registeredAccount;
  mapping(address => bool) public frozenAccount;
  address public owner;
  uint256 public constant maxTransferLimit = 15000;
  event Transfer(address indexed from, address indexed to, uint256 value);
  event FrozenAccount(address target, bool frozen);
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }
  constructor(uint256 _initialSupply) {
    owner = msg.sender;
    mintToken(owner, _initialSupply);
  }
  function transfer(address _to, uint256 _amount) public {
    require(checkLimit(_amount));
    // ...
    emit Transfer(msg.sender, _to, _amount);
  }
  function registerAccount(
    address account,
    string memory firstName,
    string memory lastName,
    bool isAdmin
  ) public onlyOwner {
    /* ... */
  }
  function checkLimit(uint256 _amount) private pure returns (bool) {
    if (_amount < maxTransferLimit) return true;
    else return false;
  }
  function validateAccount(address _account) internal view returns (bool) {
    if (!frozenAccount[_account] && tokenBalance[_account] > 0) return true;
```

```
    else return false;
  }
  function mintToken(address _recipient, uint256 _mintedAmount)
    public
    onlyOwner
  {
    tokenBalance[_recipient] += _mintedAmount;
    emit Transfer(owner, _recipient, _mintedAmount);
  }
  function freezeAccount(address target, bool freeze) public onlyOwner {
    frozenAccount[target] = freeze;
    emit FrozenAccount(target, freeze);
  }
}
```

Globalni imenski prostor:

- **block** vsebuje podatke o najnovejšem bloku na verigi blokov
- **msg** vsebuje podatke o dohodnem sporočilu
- **tx** zagotavlja podatke o transakcijah
- **this** je referenca na trenutno pogodbo
- **now** je čas kreiranja najnovejšega bloka

Tiste funkcije, za katere dovolimo, da jih pogodbe v prihodnje prilagodijo in dopolnijo delovanje, označimo z **virtual**.

Pametna pogodba je abstraktna:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

abstract contract AbstractContract {
  int256 public stateVar;
  constructor(int256 param1) {
    stateVar = param1;
  }
  function operation1(int256 opParam1, bool opParam2)
    public pure returns (int256)
  {
    if (opParam2) return opParam1;
    else return 0;
  }
  function operationA(int256 opParamA) public virtual;
}
```

Test:

```
const SimpleVoting = artifacts.require("SimpleVoting");
```

```
contract("SimpleVoting", (accounts) => {
  contract("SimpleVoting.endProposalRegistration", () => {
    it("onlyAdministrator modifier", async () => {
      // Arrange
      let simpleVotingInstance = await SimpleVoting.deployed();
      let votingAdministrator = accounts[0];
      let nonVotingAdministrator = accounts[1];
      try {
        // Act
        await simpleVotingInstance.endProposalsRegistration({
          from: nonVotingAdministrator,
        });
        assert.isTrue(false);
      } catch (error) {
        // Assert
        assert.isTrue(votingAdministrator != nonVotingAdministrator);
        assert.isTrue(error.hijackedStack.indexOf("the caller must be the administrator") >
      }
    });

    it("onlyDuringProposalsRegistration modifier", async () => {
      // Arrange
      let simpleVotingInstance = await SimpleVoting.deployed();
      let votingAdministrator = accounts[0];
      try {
        // Act
        await simpleVotingInstance.endProposalsRegistration({
          from: votingAdministrator,
        });
        assert.isTrue(false);
      } catch (error) {
        // Assert
        assert.isTrue(error.hijackedStack.indexOf("only during proposals registration") > 0)
      }
    });

    it("successful", async () => {
      // Arrange
      let simpleVotingInstance = await SimpleVoting.deployed();
      let votingAdministrator = accounts[0];
      await simpleVotingInstance.startProposalsRegistration({
        from: votingAdministrator,
      });
      let status = await simpleVotingInstance.getWorkflowStatus();
      let expectedWorkflowStatus = 1;
      assert.equal(
```

```
        status.valueOf(),
        expectedWorkflowStatus,
        "status does not correspond to proposals registration session started"
      );
      // Act
      await simpleVotingInstance.endProposalsRegistration({
        from: votingAdministrator,
      });
      let newWorkflowStatus = await simpleVotingInstance.getWorkflowStatus();
      let newExpectedWorkflowStatus = 2;
      // Assert
      assert.equal(
        newWorkflowStatus.valueOf(),
        newExpectedWorkflowStatus,
        "status does not correspond to proposals registration session ended"
      );
    });
  });
});
```