Backendprogrammering 1

NODE.JS

Utbildare: Mikael Olsson
mikael.olsson@emmio.se
076-174 90 43

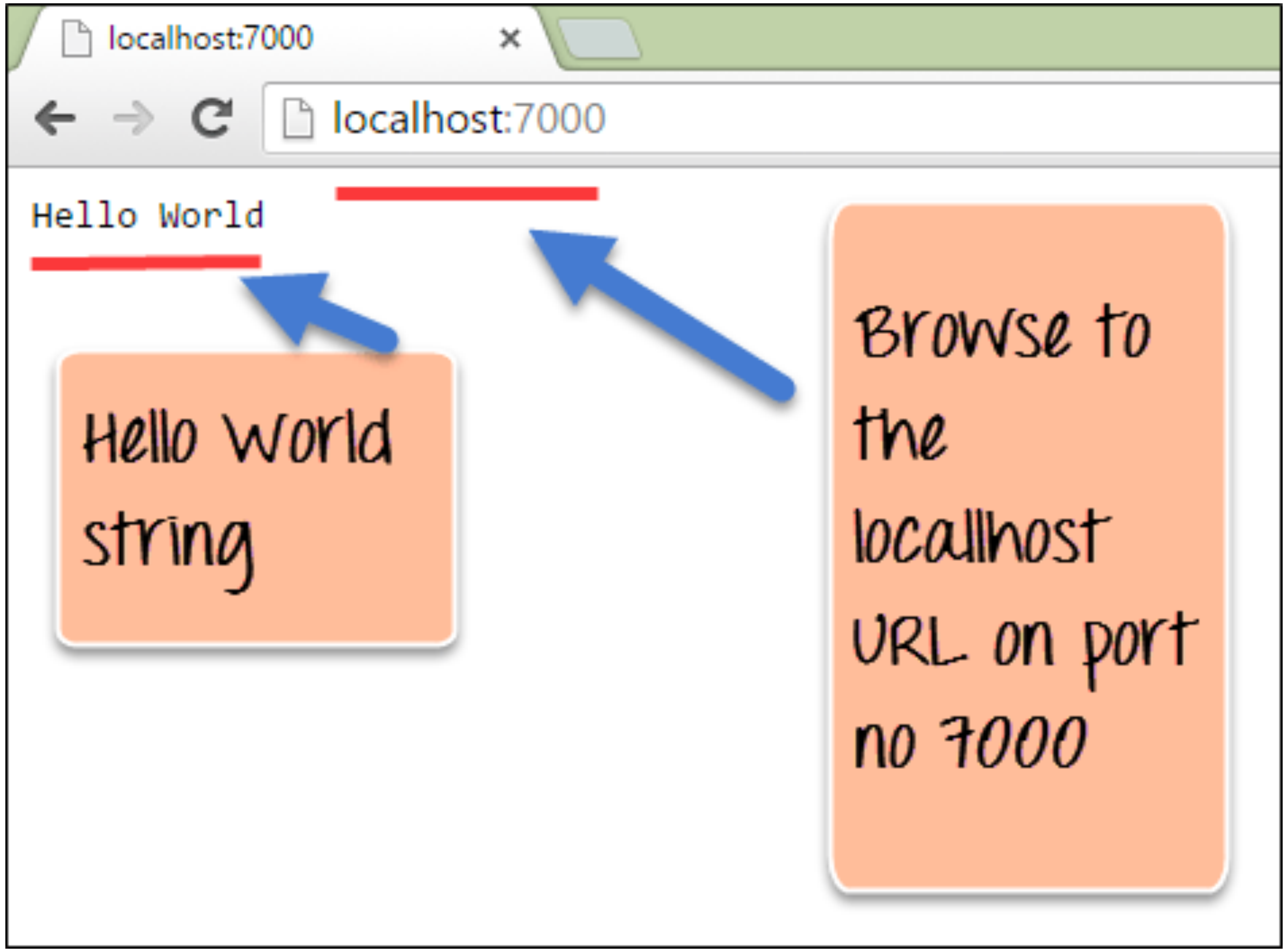
NACKADEMIN

Skapa HTTP Webbserver i Node.js

```
var http=require('http')

var server=http.createServer((function(request,response)

{
  response.writeHead(200,
      {"Content-Type": "text/plain"});
  response.end("Hello World\n");
}));

server.listen(7000);
```

1
calls the http library
2
Create the server using the http library
3
Set the content header
4
Send the string to the response
5
Make the server listen on port 7000

Webbserver



```
Hello World
```

Göra GET-requests i Node.js

Sending the response from Google to console.log
Using the request module
1
2
3
Making a GET request to Google.com

```
var request = require("request");

request("http://www.google.com", function(error, response, body)
{
  console.log(body);
});
```

Stackar

- In computing, a solution stack or software stack is a set of software subsystems or components needed to create a complete platform such that no additional software is needed to support applications.
  - Wikipedia
- En vanlig variant är LAMP
  - Linux (operating system)
  - Apache (web server)
  - MySQL or MariaDB (database management systems)
  - Perl, PHP, or Python (scripting languages)

MEAN

- I Node-världen är MEAN eller MERN en vanlig stack.
    - *MongoDB* - The standard NoSQL database
    - *Express.js* - The default web applications framework
    - *Angular.js* / *React.js* - The JavaScript framework used for web applications
    - *Node.js* - Framework used for scalable server-side and networking applications.

Express

- Vi ska kolla lite närmare på Express.
- The Express.js framework makes it very easy to develop an application which can be used to handle multiple types of requests like the GET, PUT, and POST and DELETE requests.
- Installera med `npm install express --save`

Express

```
var express=require('express');

var app=express();

app.get('/', function (req, res) {

   res.send('Hello World!');

});

var server = app.listen(3000, function () {

});
```

1
Use the express module
2
Create an object of the express module
3
Create a callback function
4
Send 'Hello World' response
5
Make the server listen on port 3000

one to send content back to the web page.

Routes

- Routing determine the way in which an application responds to a client request to a particular endpoint.

- For example, a client can make a GET, POST, PUT or DELETE http request for various URL such as the ones shown below;

```
http://localhost:3000/Books
http://localhost:3000/Students
```

- So based on the URL which is accessed, a different functionality on the webserver will be invoked, and accordingly, the response will be sent to the client. This is the concept of routing.

- Each route can have one or more handler functions, which are executed when the route is matched.

Routes

- The general syntax for a route:

```
app.METHOD(PATH, HANDLER)
```

- Wherein,

1. *app* is an instance of the express module

2. *METHOD* is an HTTP request method (`GET`, `POST`, `PUT` or `DELETE`)

3. *PATH* is a path on the server.

4. *HANDLER* is the function executed when the route is matched.

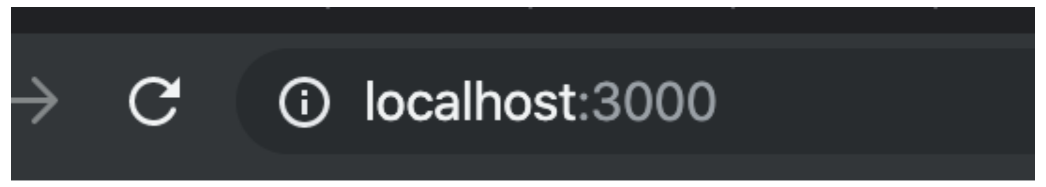
Routes

```
app.route('/Node').get(function(req,res)

    {
        res.send("Tutorial On Node");

    });

app.route('/Angular').get(function(req,res)

    {
        res.send("Tutorial On Angular");

    });

app.get('/', function (req, res) {
    res.send('Welcome to Guru99 Tutorials');
});
```

1
Create a Node route
2
Send a different response for the Node route
3
Create a Angular route
4
Send a different response for the Angular route
5
Our default route

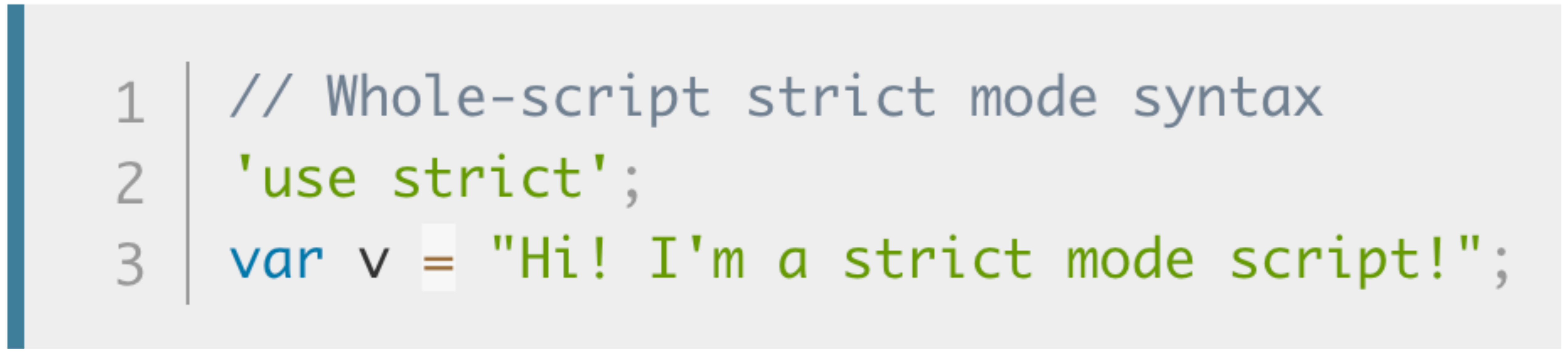
Localhost

- DNS:en översätter domännamn till IP-adresser.
- Vissa namn är reserverade, t ex `localhost` som betyder samma som `127.0.0.1`.



use strict

- The "use strict" directive was new in ECMAScript version 5.
- It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
- The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
- With strict mode, you can not, for example, use undeclared variables.
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode



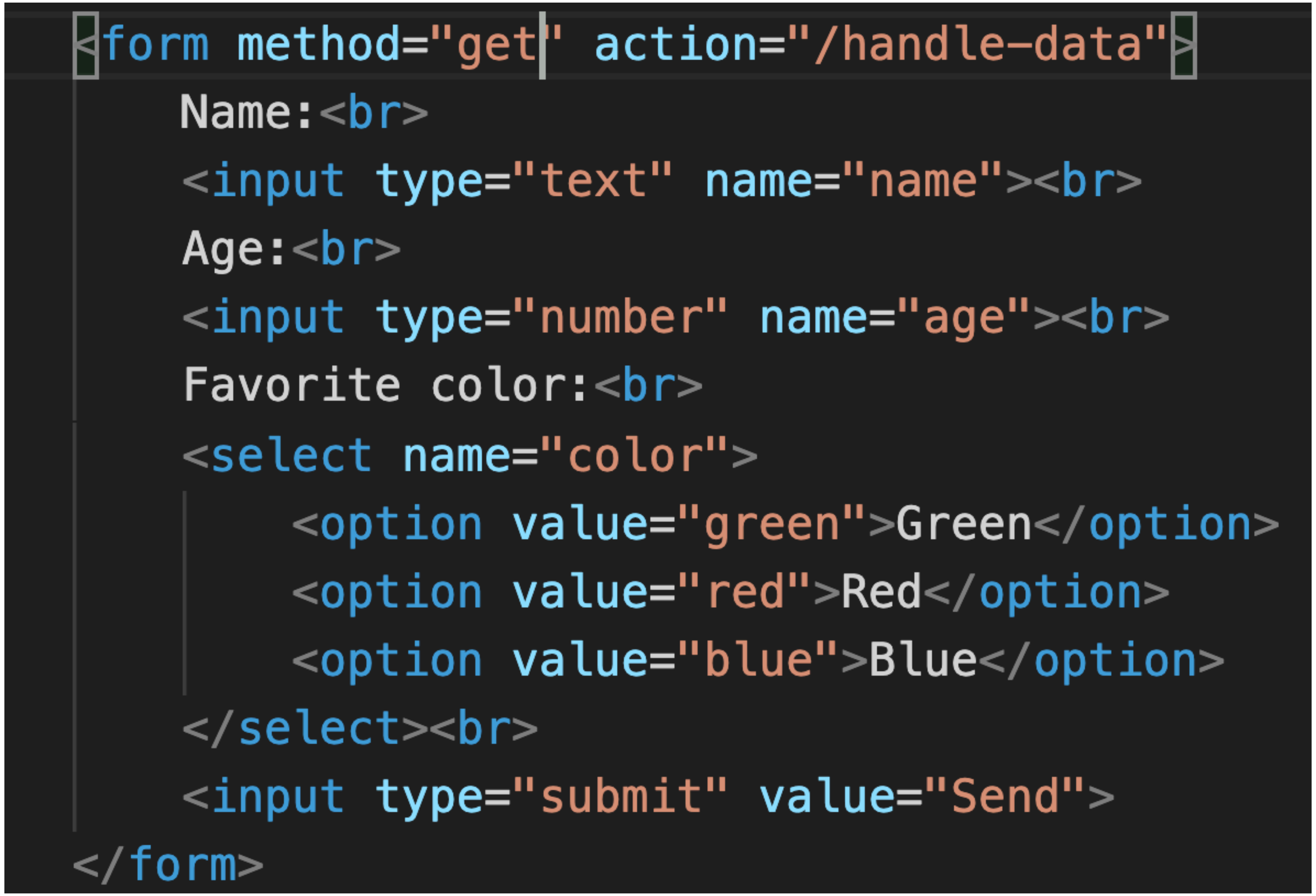
```
// Whole-script strict mode syntax
'use strict';
var v = "Hi! I'm a strict mode script!";
```

URL parameters

- Vi kan hämta delar av URL:en som parametrar med kolon.
- *Uppgift:* Skriv en route som lyssnar på adressen `/age/` och skriver ut olika saker beroende på vilken ålder användaren angett i URL:en, t ex att användaren får ta körkort om hen är äldre än 18.
- Parametrar är alltid strängar!

```
// Request comes in to /instructor/ANYTHING
app.get('/instructor/:name', function (request, response) {
    response.send('Läraren heter ' + request.params.name);
});
```

Formulär



```html
<form method="get" action="/handle-data">
    Name:<br>
    <input type="text" name="name"><br>
    Age:<br>
    <input type="number" name="age"><br>
    Favorite color:<br>
    <select name="color">
        <option value="green">Green</option>
        <option value="red">Red</option>
        <option value="blue">Blue</option>
    </select><br>
    <input type="submit" value="Send">
</form>
```

Ta emot formulärdata

- GET: `request.query.`*`komponent-namn`*
- POST: lite knepigare:
  ```
  npm install body-parser —save
  const bodyParser = require('body-parser');
  app.use(bodyParser.urlencoded({extended: true}));
  ```
- Nu kan vi använda: `request.body.`*`komponent-namn`*

Uppgift

- Gör ett formulär som innehåller en datumväljare och en färgväljare.

- Ta emot requestet i din applikation och skriv ut datumet med färgen som användaren valde, typ:

```
<p style="color: #65f988”>2021-02-17</p>
```

Litet projekt

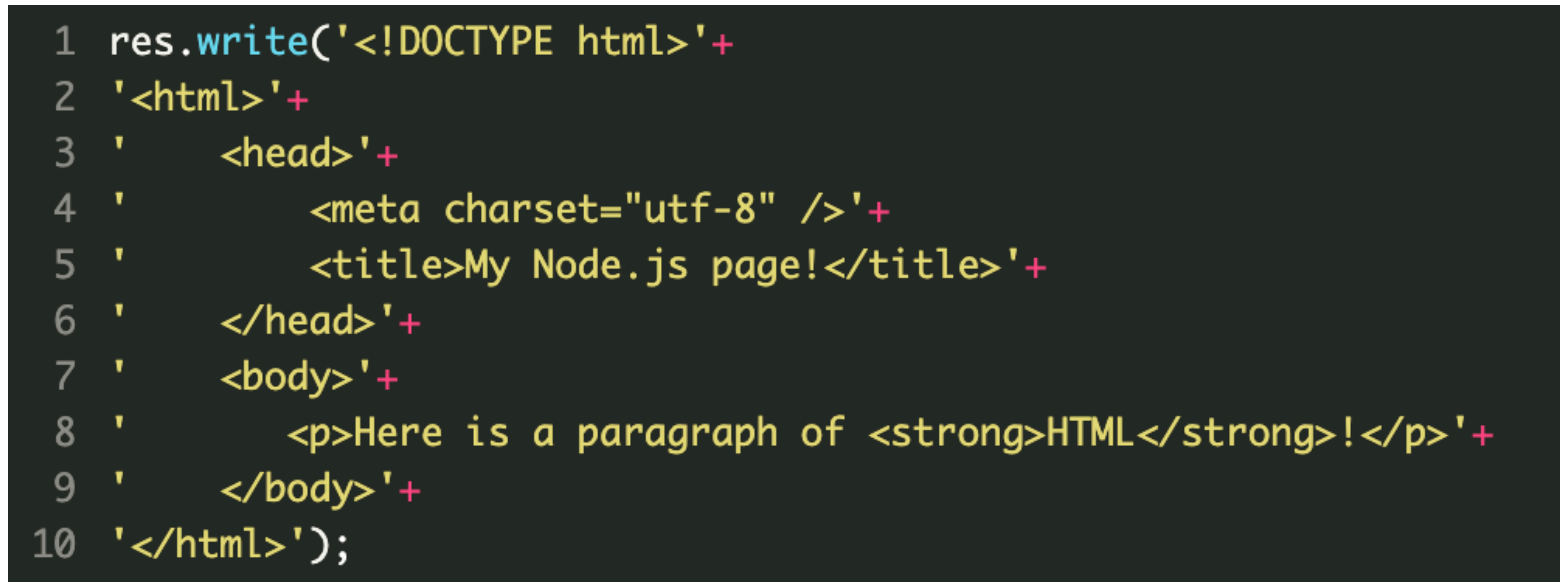
- Bygg ett formulär där användaren kan fylla i ett sökord.
- Ta emot formulärdatat och hämta skämt som matchar sökningen på https://icanhazdadjoke.com/api .
- Presentera resultatet på sidan.

Templates

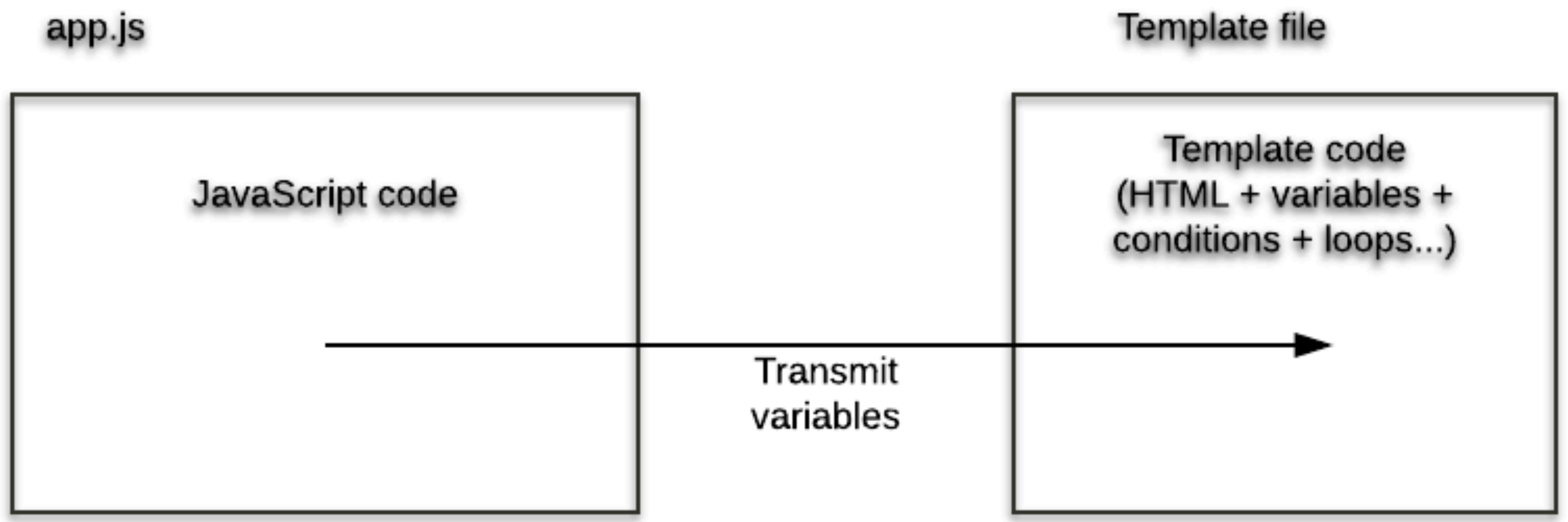
- Vi har skickat tillbaka enstaka strängar eller hela html-filer hittills.

- Express låter oss använda olika *template engines* (eller *template languages*) för att skapa sidor enklare.

- Vi ska titta lite på ett av dem: EJS. (Embedded JavaScript)



```
res.write('<!DOCTYPE html>'+
'<html>'+
'    <head>'+
'        <meta charset="utf-8" />'+
'        <title>My Node.js page!</title>'+
'    </head>'+
'    <body>'+
'       <p>Here is a paragraph of <strong>HTML</strong>!</p>'+
'    </body>'+
'</html>');
```



EJS

- `npm install ejs --save`
- Nu kan vi skicka med variabler till en template:
- `res.render('bedroom.ejs', {floor: req.params.floornum});`
- This code calls for a `bedroom.ejs` file that must be found in a sub-folder called ”views".

Multiple parameters and loops

- JS:

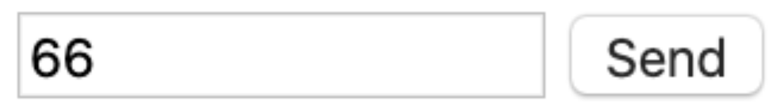
```
app.get('/', function(req, res) {
    let names = ['Robert', 'Jack', 'David'];
    res.render('page.ejs', {names: names, subject: 'Lorem ipsum'});
});
```

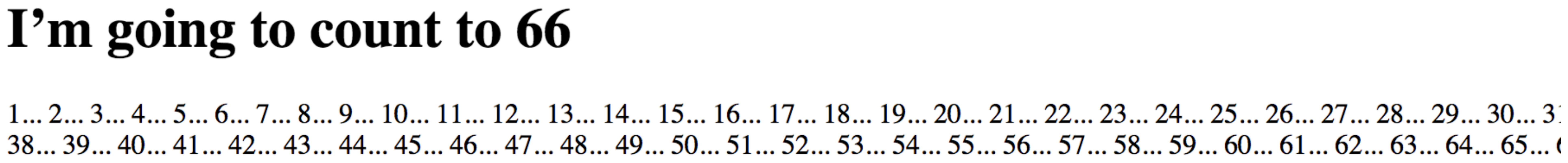
- Template:

```
<h1><%= subject %></h1>

<ul>
  <% for(let i = 0; i < names.length ; i++) { %>
    <li><%= names[i] %></li>
<% } %>
</ul>
```

Uppgift

- Gör en sida som renderar ett formulär som innehåller en siffra. Formuläret ska postas (POST) till en annan sida. En router ska ta hand om det anropet och räkna från 1 till det nummer användaren postade med en template.





I'm going to count to 66

1... 2... 3... 4... 5... 6... 7... 8... 9... 10... 11... 12... 13... 14... 15... 16... 17... 18... 19... 20... 21... 22... 23... 24... 25... 26... 27... 28... 29... 30... 3
38... 39... 40... 41... 42... 43... 44... 45... 46... 47... 48... 49... 50... 51... 52... 53... 54... 55... 56... 57... 58... 59... 60... 61... 62... 63... 64... 65...

Middlewares

- Express is a framework based on middleware, which are application modules each providing a specific feature. You can decide which middleware you want to load.

- Express comes with over 15 basic pieces of middleware, and of course developers can add others via NPM. Each piece of middleware provides a micro-feature. Here are a few examples:

- *compression*: enables for gzip compression of pages for faster sending to the server.
- *cookie-parser*: allows you to work with cookies.
- *cookie-session*: allows you to generate session information (during a visitor’s stay on the website).
- *serve-static*: allows the return of static files contained in a folder (images, files to download, etc.).
- *serve-favicon*: manages your website's favicon.
- etc.

Middlewares

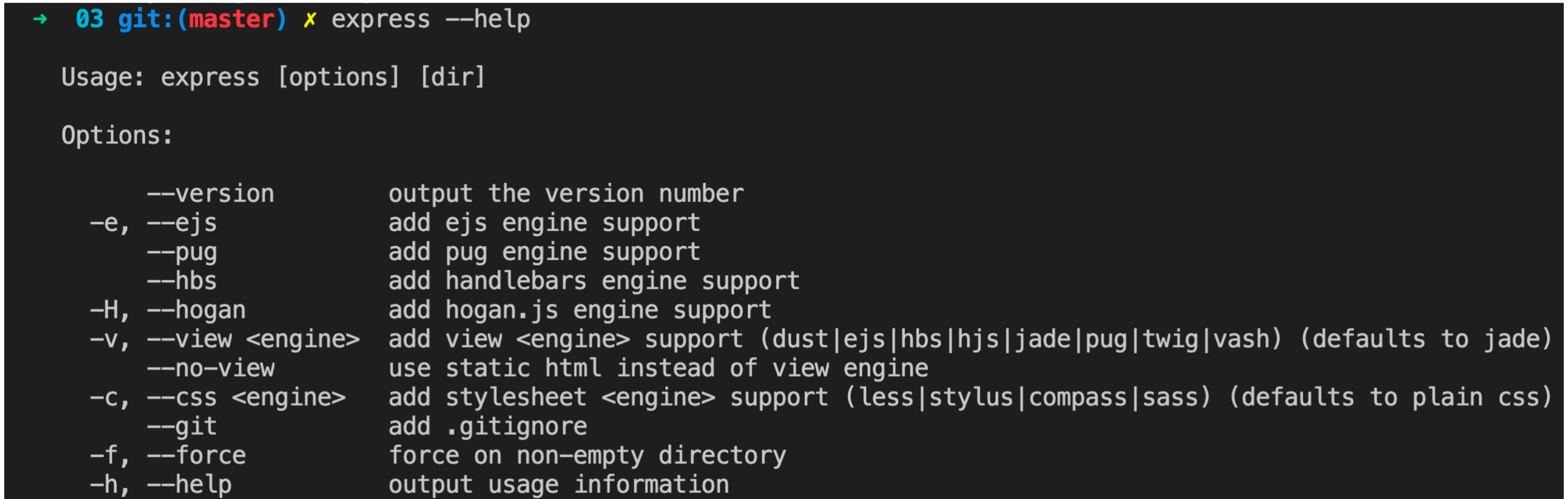
```javascript
var express = require('express');
var morgan = require('morgan'); // loads the piece of middleware for logging
var favicon = require('serve-favicon'); // loads the piece of middleware for the favicon

var app = express();

app.use(morgan('combined')) // loads the piece of middleware for logging
.use(express.static(__dirname + '/public')) // Specifies that the /public folder includes static files (basic piece of middleware loaded)
.use(favicon(__dirname + '/public/favicon.ico')) // Activates the favicon specified
.use(function(req, res){ // finally answers
    res.send('Hello');
});

app.listen(8080);
```

You'll need to install the middleware you need with `npm install` before running this code.

The calling order for the middleware is extremely important. For example, we start by activating the logger. If we did it last, we wouldn't be able to log anything! When you call the middleware, think of the order, because it can have a big impact on the functioning of your app.

Express application generator

- Hjälper oss att generera ett ”skelett” för ett projekt.
- `npm install -g express-generator`

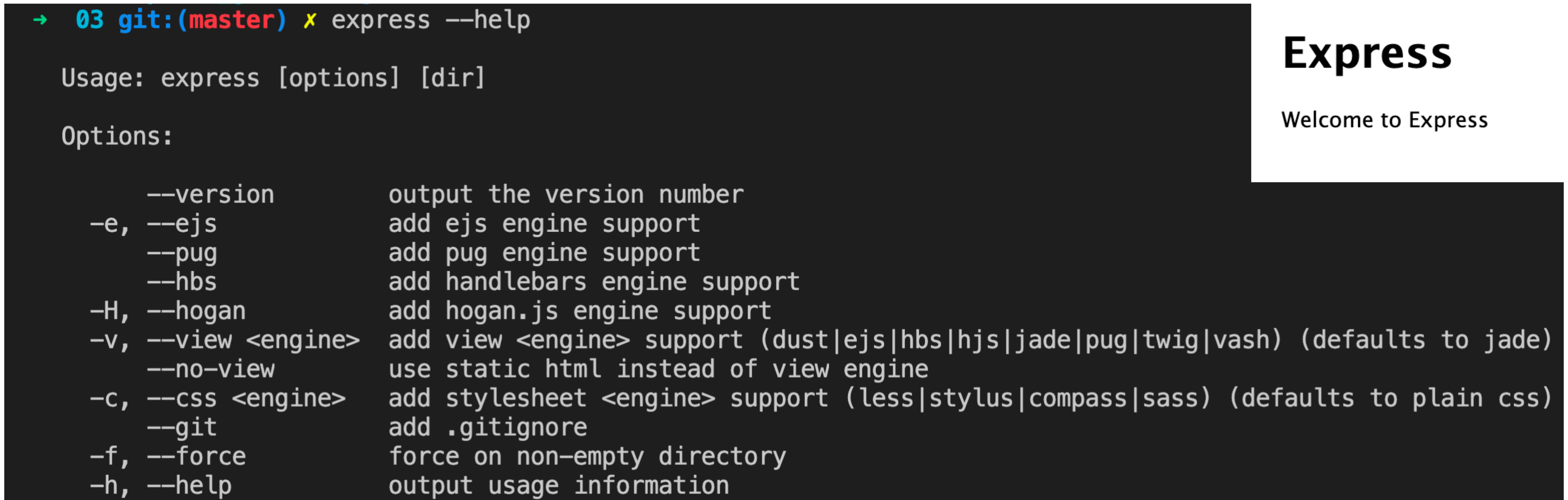


```
➜  03 git:(master) ✗ express --help

  Usage: express [options] [dir]

  Options:

        --version        output the version number
    -e, --ejs            add ejs engine support
        --pug            add pug engine support
        --hbs            add handlebars engine support
    -H, --hogan          add hogan.js engine support
    -v, --view <engine>  add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
        --no-view        use static html instead of view engine
    -c, --css <engine>   add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
        --git            add .gitignore
    -f, --force          force on non-empty directory
    -h, --help           output usage information
```

Express application generator

- Skapa projekt med de options du vill, t ex:

- `express --view=ejs --css=sass --git my_proj`
- `cd my_proj`
- `npm install`
- `DEBUG=myapp:* npm start`



```
➜ 03 git:(master) ✗ express --help

  Usage: express [options] [dir]

  Options:

        --version        output the version number
    -e, --ejs            add ejs engine support
        --pug            add pug engine support
        --hbs            add handlebars engine support
    -H, --hogan          add hogan.js engine support
    -v, --view <engine>  add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
        --no-view        use static html instead of view engine
    -c, --css <engine>   add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
        --git            add .gitignore
    -f, --force          force on non-empty directory
    -h, --help           output usage information
```

Express

Welcome to Express

Session

- En session är lite som en cookie. Kan användas för att spara temporära data på servern, t ex om användaren är inloggad eller inte.
- https://www.npmjs.com/package/cookie-session
- Låt oss kolla in ett exempel.

cookieSession

- We can generate a session using the following command:

-
  ```
  app.use(session({
    secret: 'veryimportantsecret',
  }))
  ```

- The secret is used to sign the cookie using the cookie-signature library. Cookies are signed using Hmac-sha256 and converted to a base64 string. We can have multiple secrets as an array. The first secret will be used to sign the cookie. The rest will be used in verification.

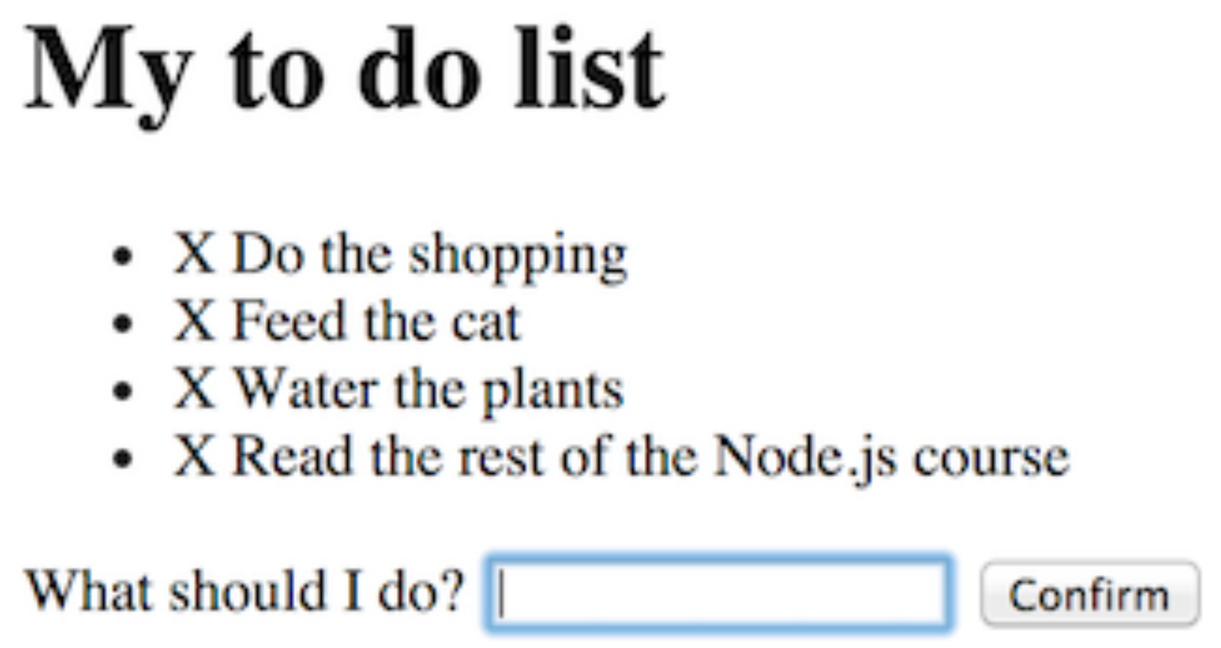
```
app.use(session({
  secret: ['veryimportantsecret', 'notsoimportantsecret',
'highlyprobablysecret' ],
}))
```

https://blog.jscrambler.com/best-practices-for-secure-session-management-in-node/

Todo-list

- *Uppgift*: Skapa en todo-lista. Skapa applikationen mha express-generate.
- We can add elements to the to do list via the form.
- We can delete items by clicking on the crosses in the list.
- The list is stored in the visitor's session. If someone else connects to the site, they will have their own list.
- In theory, we can associate a route to each of these features:
  (Vilken metod ska vi lyssna efter på varje?)

- `/todo`: list of tasks.
- `/todo/add`: add a task.
- `/todo/delete/:id`: delete task n°id.



My to do list

- X Do the shopping
- X Feed the cat
- X Water the plants
- X Read the rest of the Node.js course

What should I do?