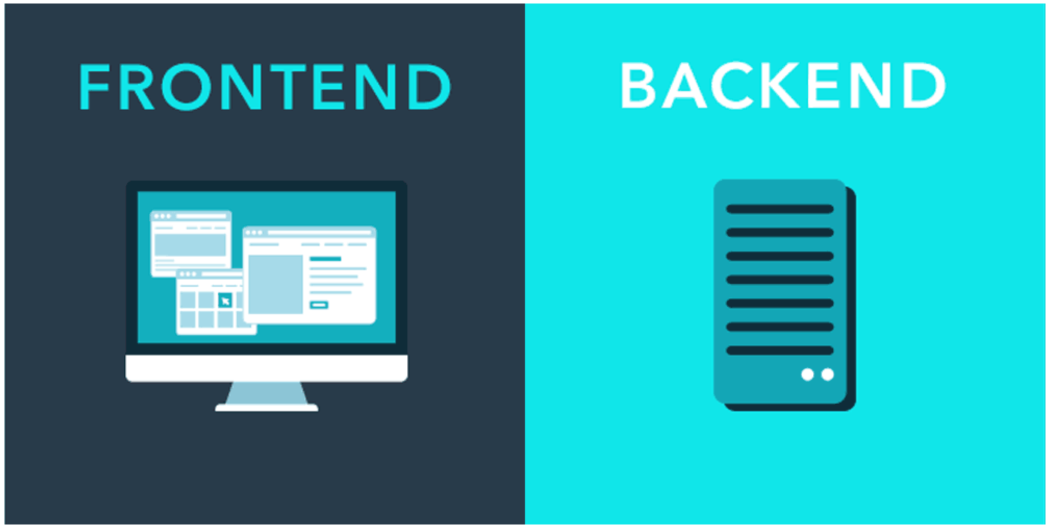
BACKEND DEVELOPMENT WITH RESTFUL APIS & EXPRESS.JS

Workshop 2

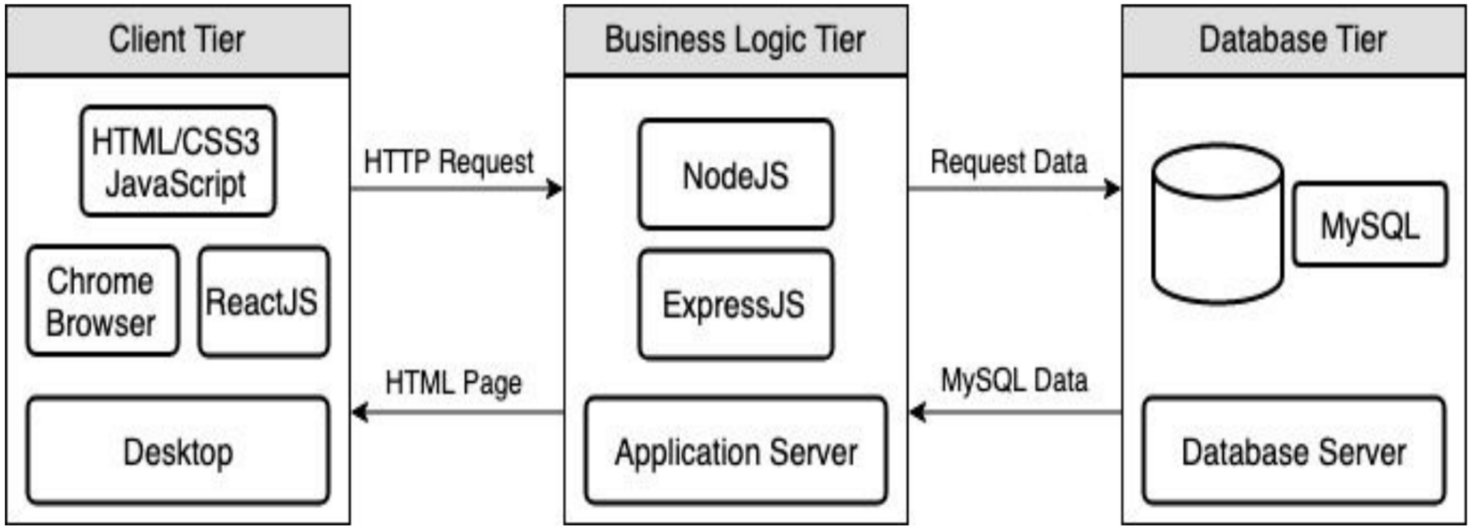
Rest apis and requests

What is backend Programming

- Backend Development is also known as server-side development
- It is everything that the users don't see and contains behind-the-scenes activities that occur when performing any action on a website
- Focuses primarily on databases, backend logic, APIs, and Servers



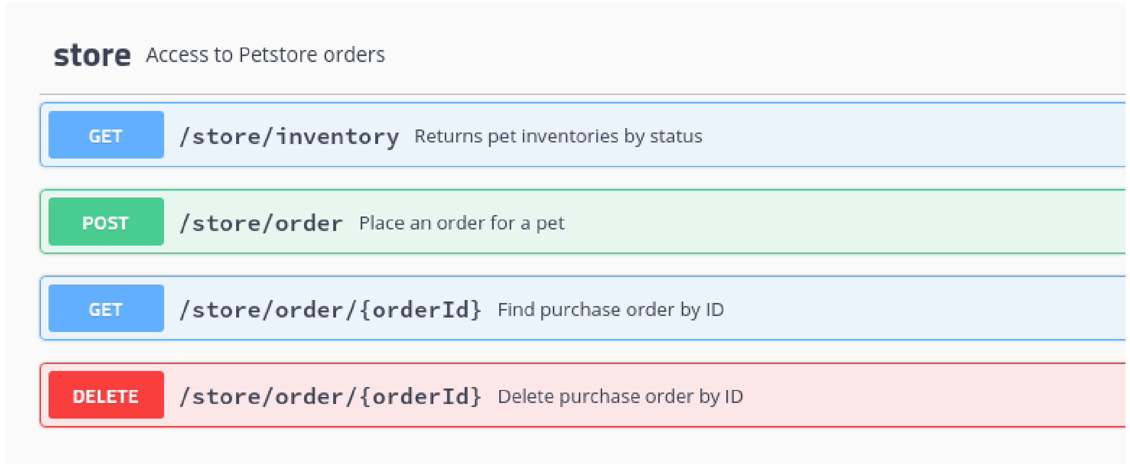
ARCHITECTURE



HTTP REQUESTS

- Are requests by which you communicate with the server (some computer listening for requests) on some port

- The requests *hit* the endpoints that perform functions of:
  - Get
  - Post
  - Patch
  - Put
  - Delete



store Access to Petstore orders

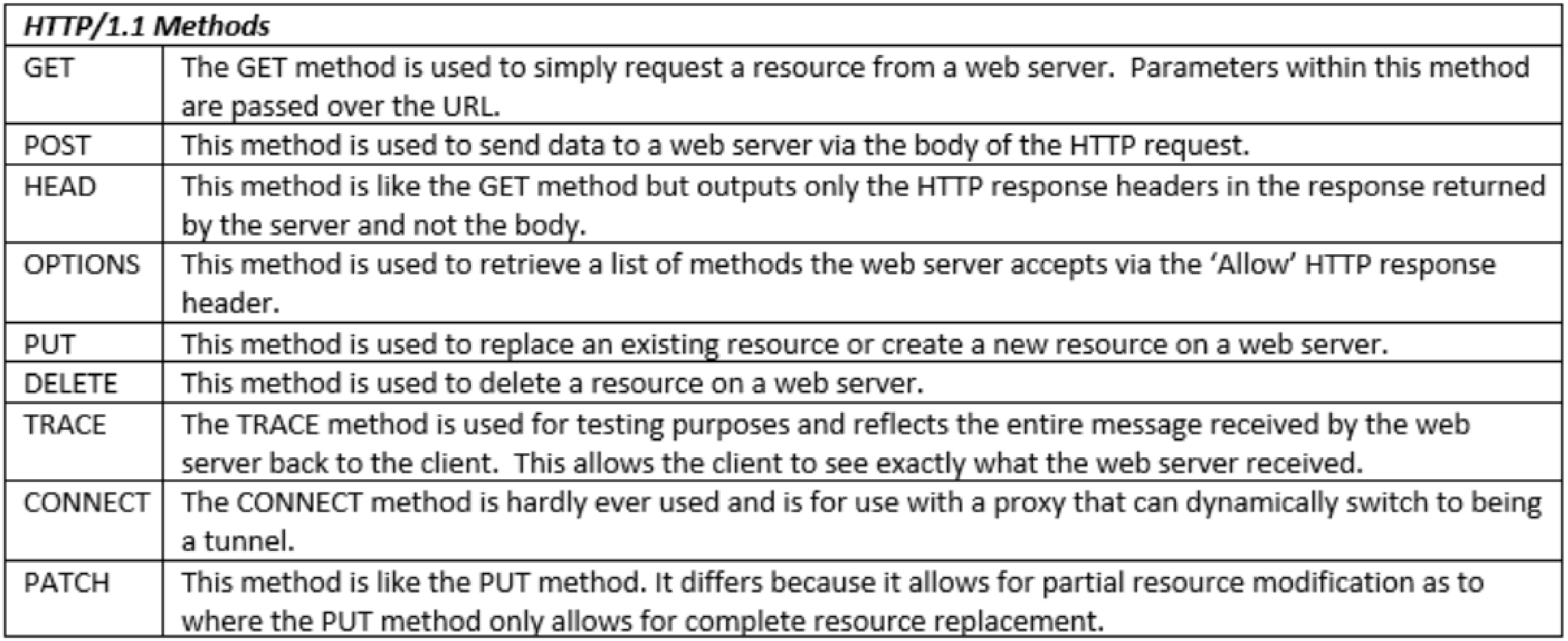
GET `/store/inventory` Returns pet inventories by status

POST `/store/order` Place an order for a pet

GET `/store/order/{orderId}` Find purchase order by ID

DELETE `/store/order/{orderId}` Delete purchase order by ID

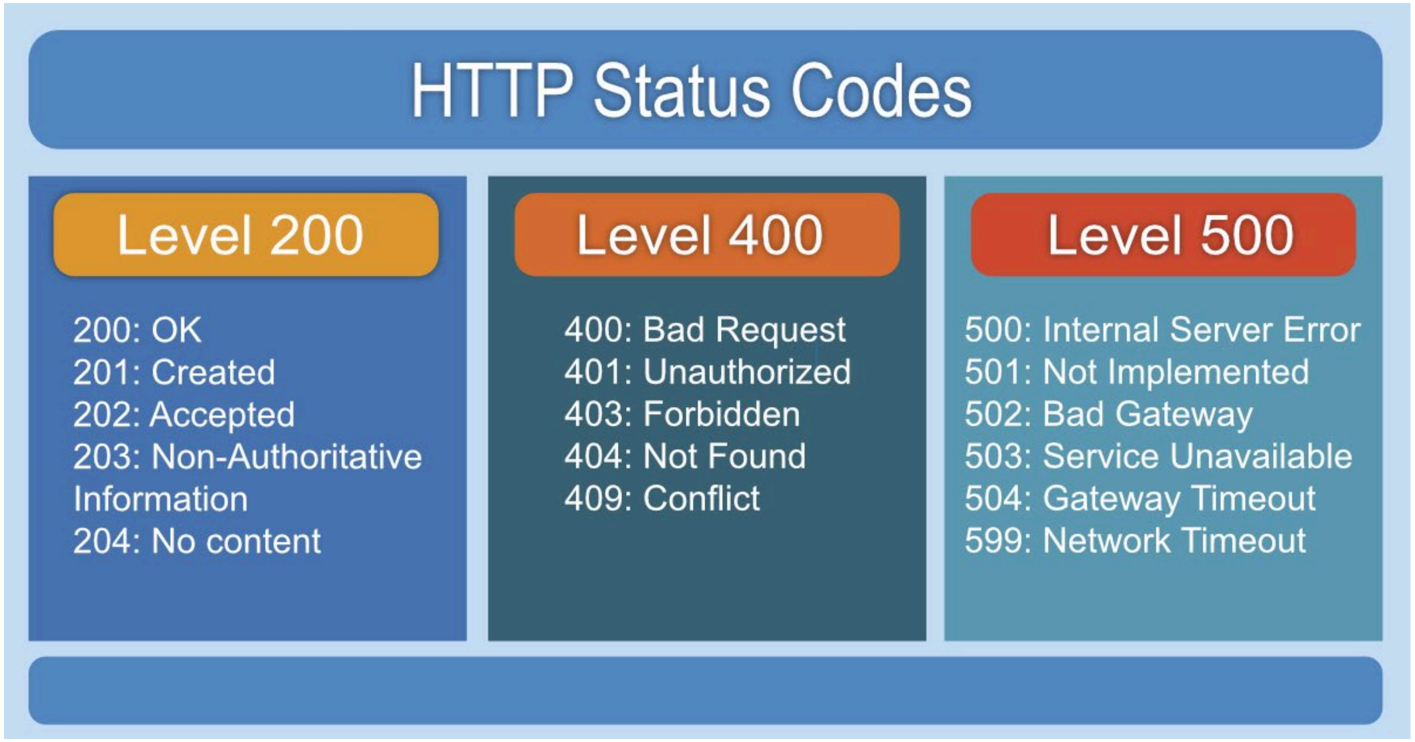
HTTP Methods



HTTP/1.1 Methods	
GET	The GET method is used to simply request a resource from a web server. Parameters within this method are passed over the URL.
POST	This method is used to send data to a web server via the body of the HTTP request.
HEAD	This method is like the GET method but outputs only the HTTP response headers in the response returned by the server and not the body.
OPTIONS	This method is used to retrieve a list of methods the web server accepts via the 'Allow' HTTP response header.
PUT	This method is used to replace an existing resource or create a new resource on a web server.
DELETE	This method is used to delete a resource on a web server.
TRACE	The TRACE method is used for testing purposes and reflects the entire message received by the web server back to the client. This allows the client to see exactly what the web server received.
CONNECT	The CONNECT method is hardly ever used and is for use with a proxy that can dynamically switch to being a tunnel.
PATCH	This method is like the PUT method. It differs because it allows for partial resource modification as to where the PUT method only allows for complete resource replacement.

HTTP RESPONSES

- Let us know whether the request was successful

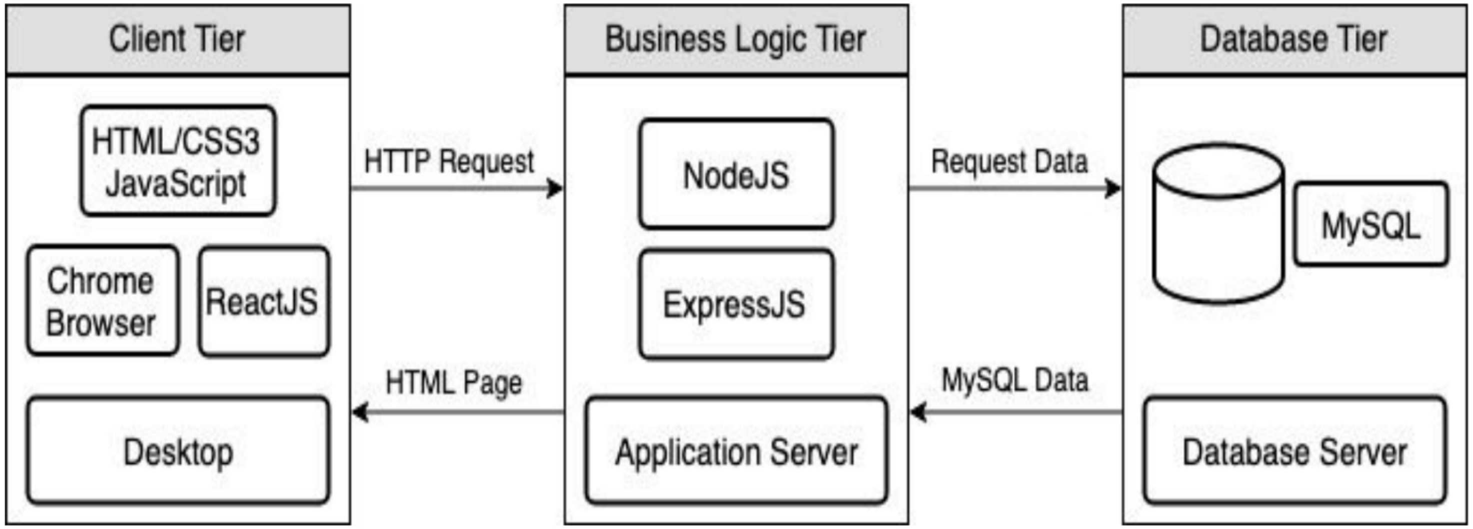


HTTP Status Codes

Level 200	Level 400	Level 500
200: OK	400: Bad Request	500: Internal Server Error
201: Created	401: Unauthorized	501: Not Implemented
202: Accepted	403: Forbidden	502: Bad Gateway
203: Non-Authoritative Information	404: Not Found	503: Service Unavailable
204: No content	409: Conflict	504: Gateway Timeout
		599: Network Timeout

RESTFUL SERVICES

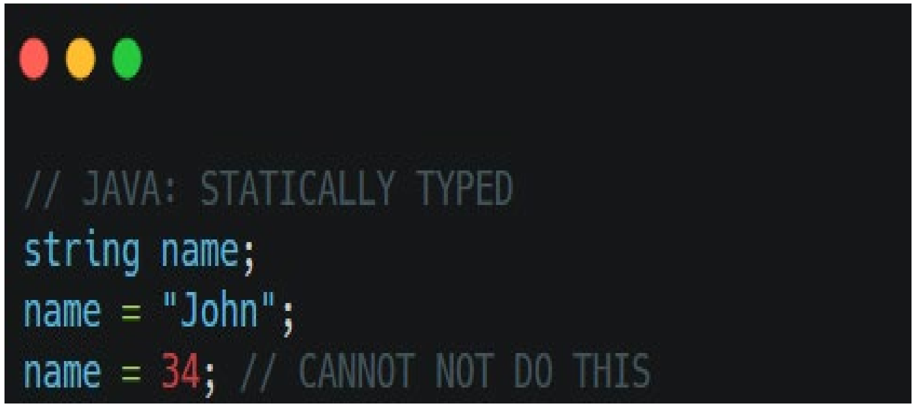
- **Stateless**: the client and server do not need to know about each other
- **Separation of client and server**: as long as the format of the messages is known, they can remain modular and separate
- **resource identification through URI**: Resources are accessed via paths



Javascript

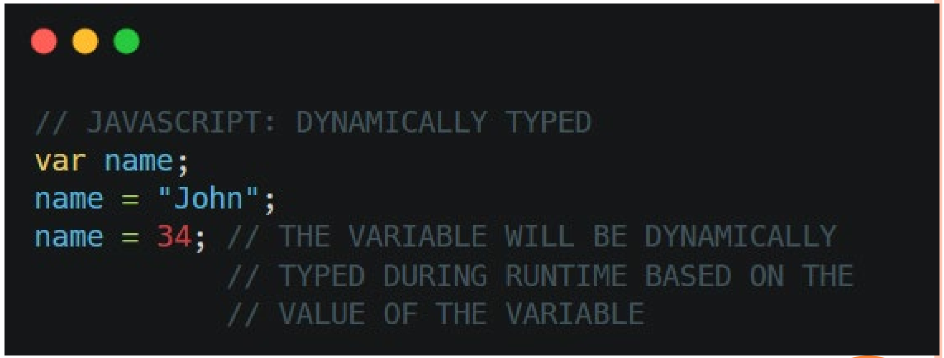
What is JavaScript

- Interpreted at runtime: the code is executed as it is ran, so no compiled files like .jar or .exe



```
// JAVA: STATICALLY TYPED
string name;
name = "John";
name = 34; // CANNOT NOT DO THIS
```

- Dynamically typed: types of the objects are interpreted during run time



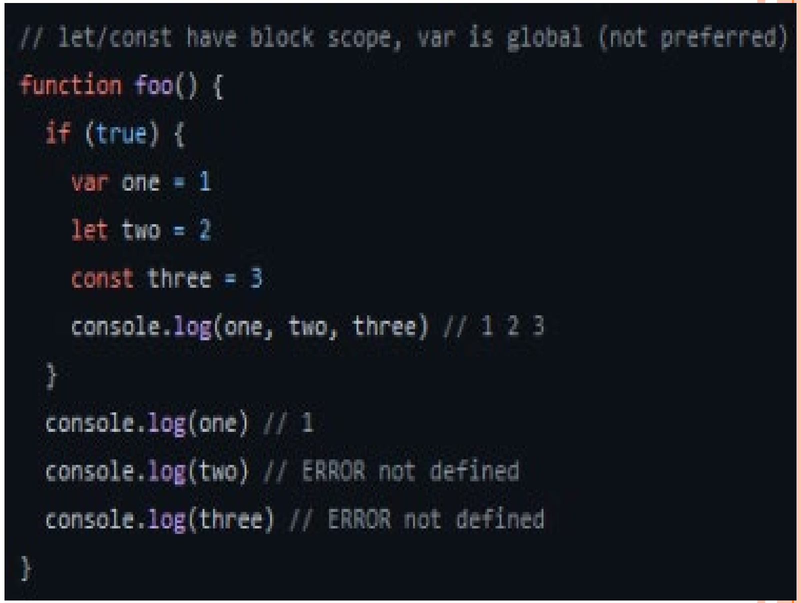
```
// JAVASCRIPT: DYNAMICALLY TYPED
var name;
name = "John";
name = 34; // THE VARIABLE WILL BE DYNAMICALLY
           // TYPED DURING RUNTIME BASED ON THE
           // VALUE OF THE VARIABLE
```

Syntax: Declaration and Scope

Variable Declaration: const vs var vs let

- **Scopes**:
    - **Global Scope-** variable accessible everywhere
    - **Function Scope-** variable accessible in function only
    - **Block Scope-** variable accessible in blocks of: while loops, for loops, switch statements, if statements

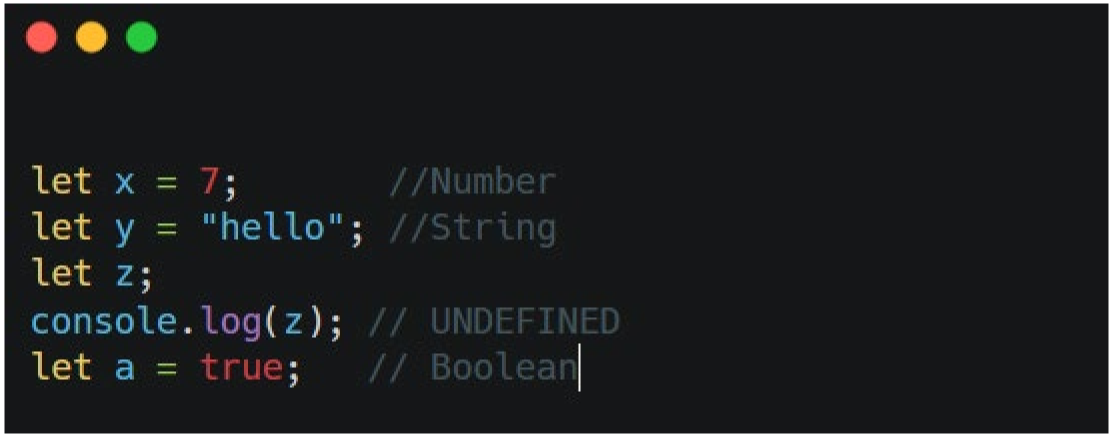
- **Hoisting:** All declarations are moved to the top of their scope



```
// let/const have block scope, var is global (not preferred)
function foo() {
  if (true) {
    var one = 1
    let two = 2
    const three = 3
    console.log(one, two, three) // 1 2 3
  }
  console.log(one) // 1
  console.log(two) // ERROR not defined
  console.log(three) // ERROR not defined
}
```

Syntax: Types

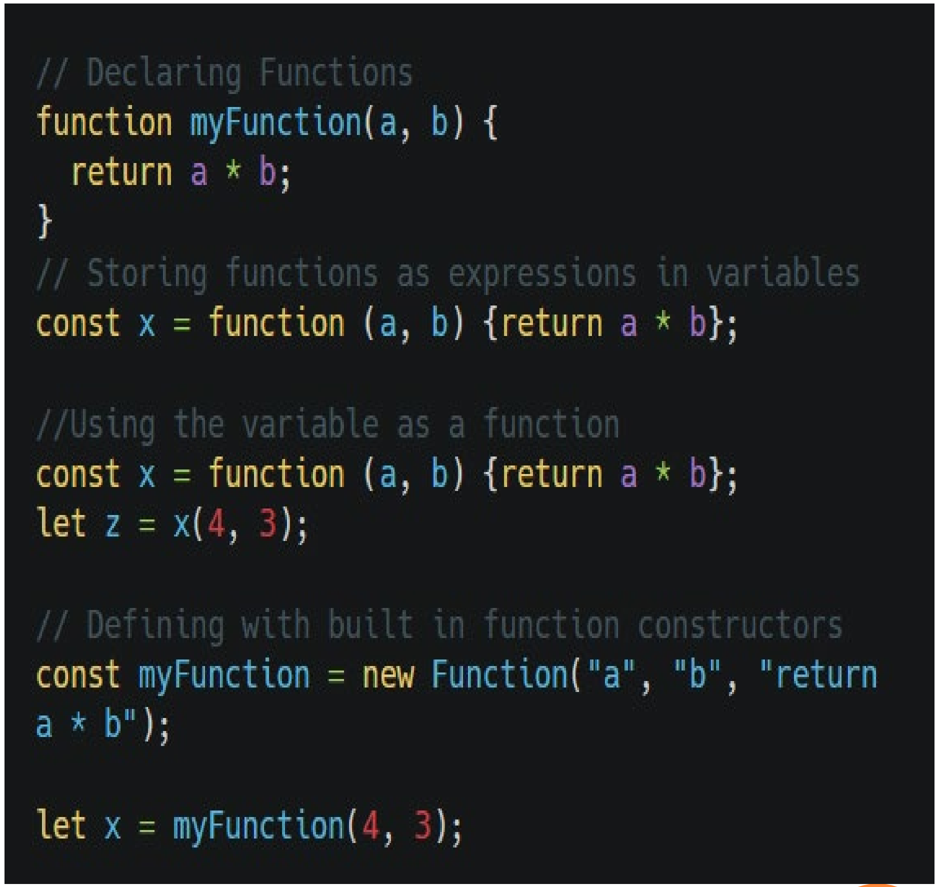
- Number:
- String
- Boolean
- Undefined



```
let x = 7;          //Number
let y = "hello"; //String
let z;
console.log(z); // UNDEFINED
let a = true;    // Boolean
```

Functions

- Multiple ways to create functions
- Function declarations are also moved to the top of their scope because of hoisting, except for those that are defined as variable expressions

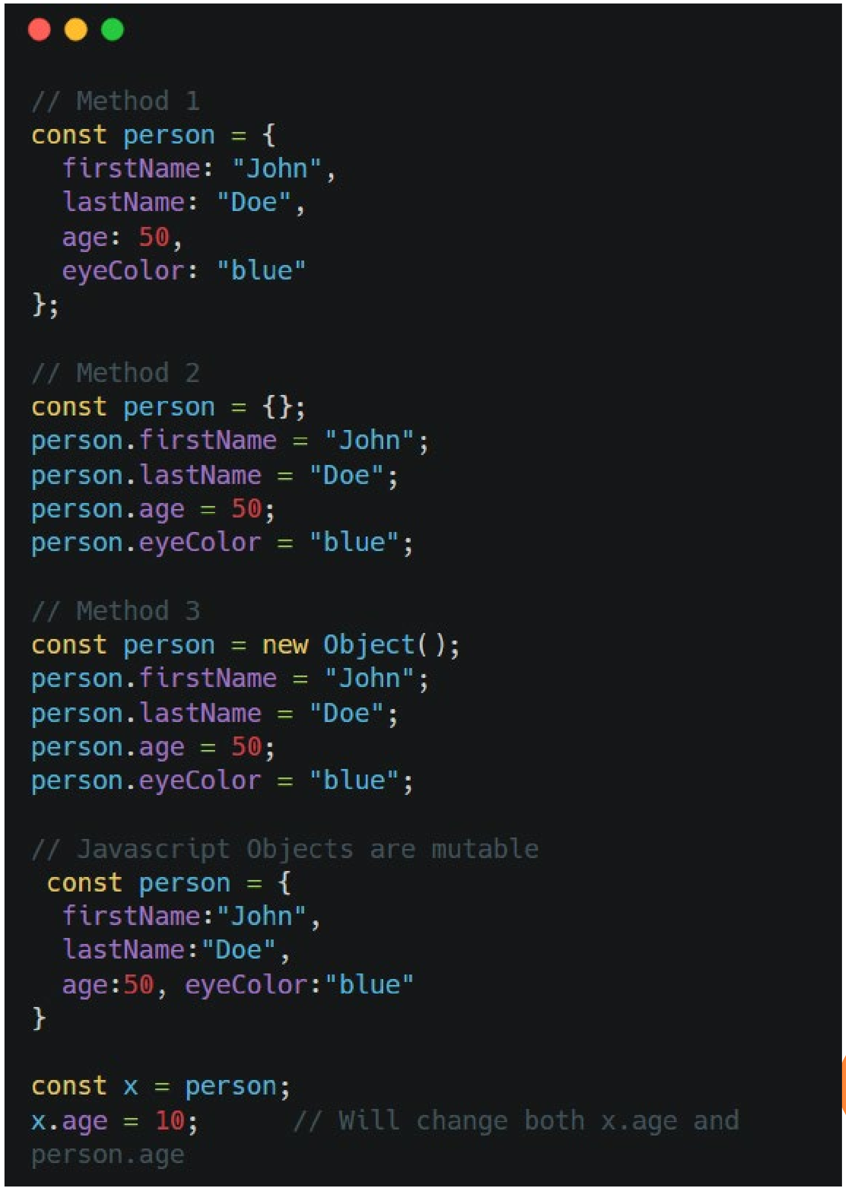


```
// Declaring Functions
function myFunction(a, b) {
  return a * b;
}
// Storing functions as expressions in variables
const x = function (a, b) {return a * b};

//Using the variable as a function
const x = function (a, b) {return a * b};
let z = x(4, 3);

// Defining with built in function constructors
const myFunction = new Function("a", "b", "return
a * b");

let x = myFunction(4, 3);
```

OBJECTS

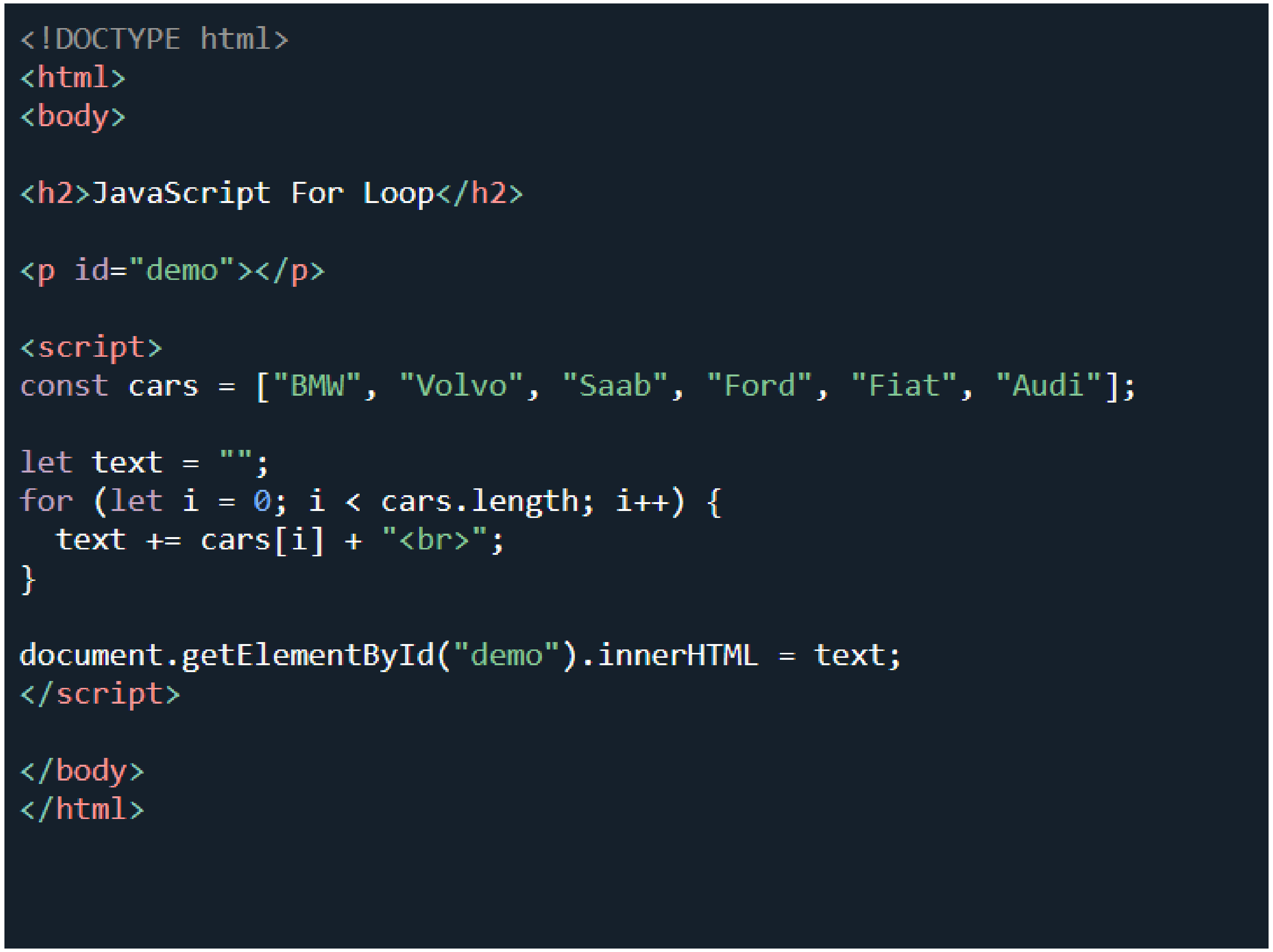
- Objects can be created in a variety of different ways

- Objects are mutable



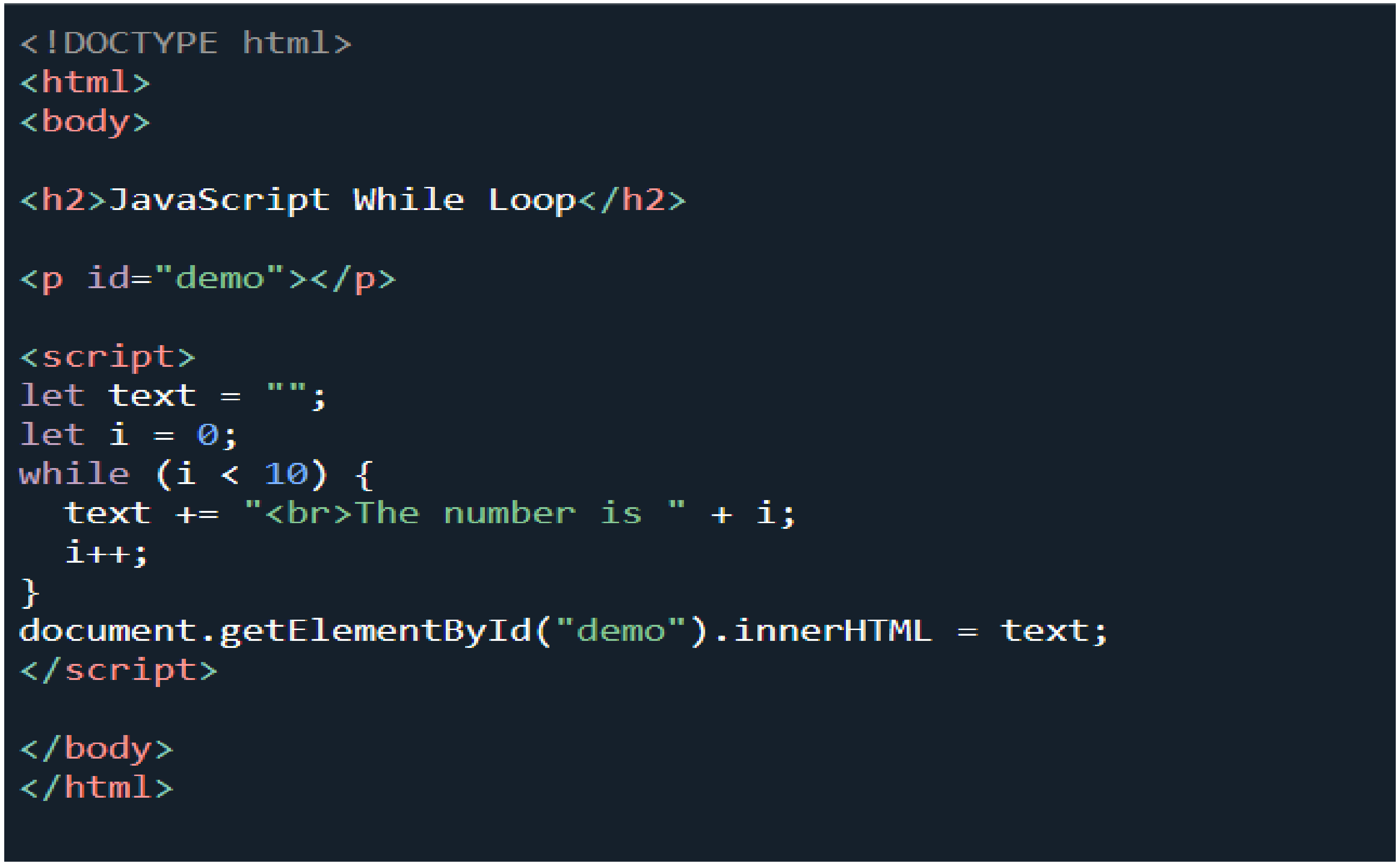
```
// Method 1
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

// Method 2
const person = {};
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";

// Method 3
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";

// Javascript Objects are mutable
 const person = {
  firstName:"John",
  lastName:"Doe",
  age:50, eyeColor:"blue"
}

const x = person;
x.age = 10;      // Will change both x.age and
person.age
```

For loop



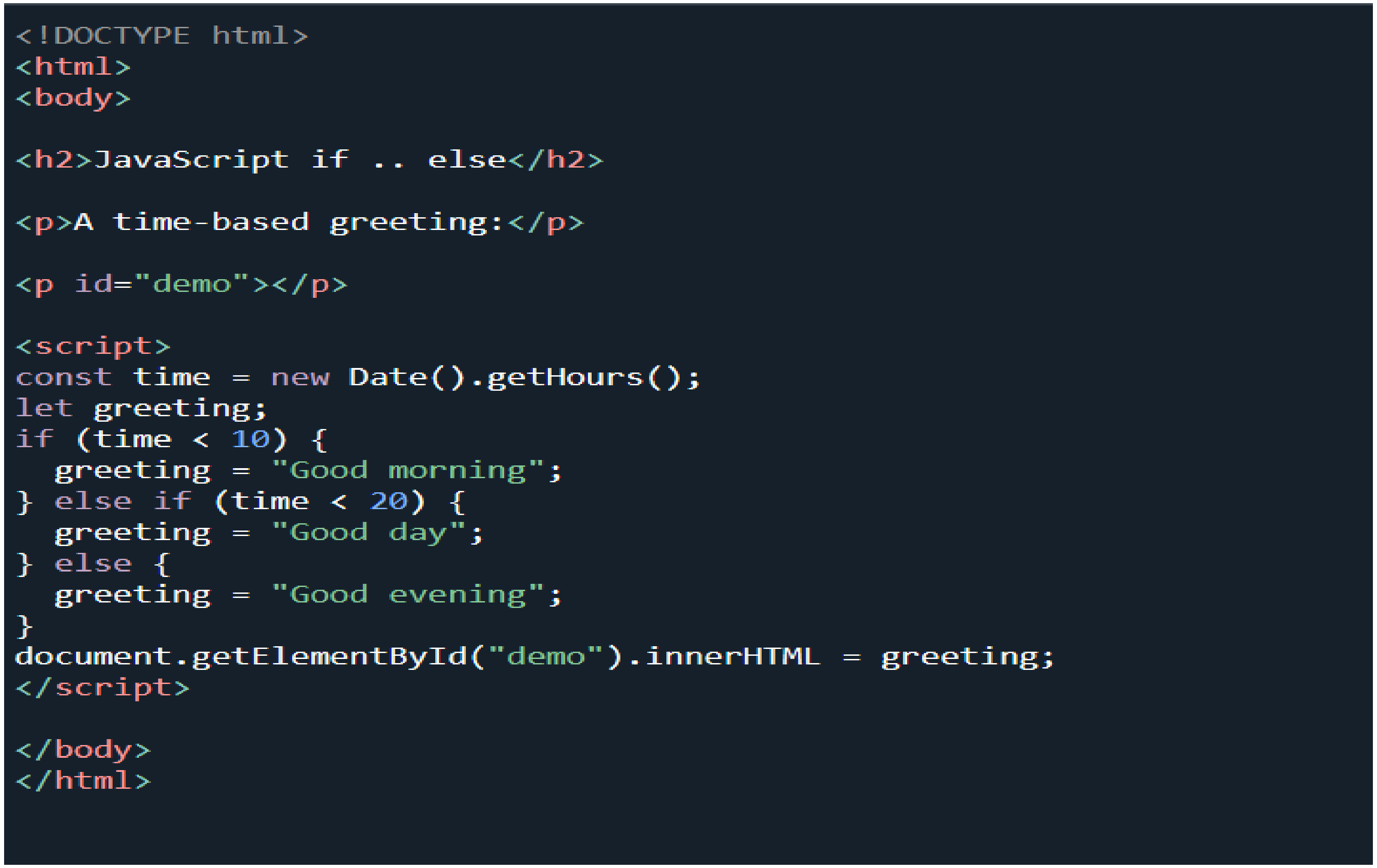
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For Loop</h2>

<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

let text = "";
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

While loop



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript While Loop</h2>

<p id="demo"></p>

<script>
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

If else if else



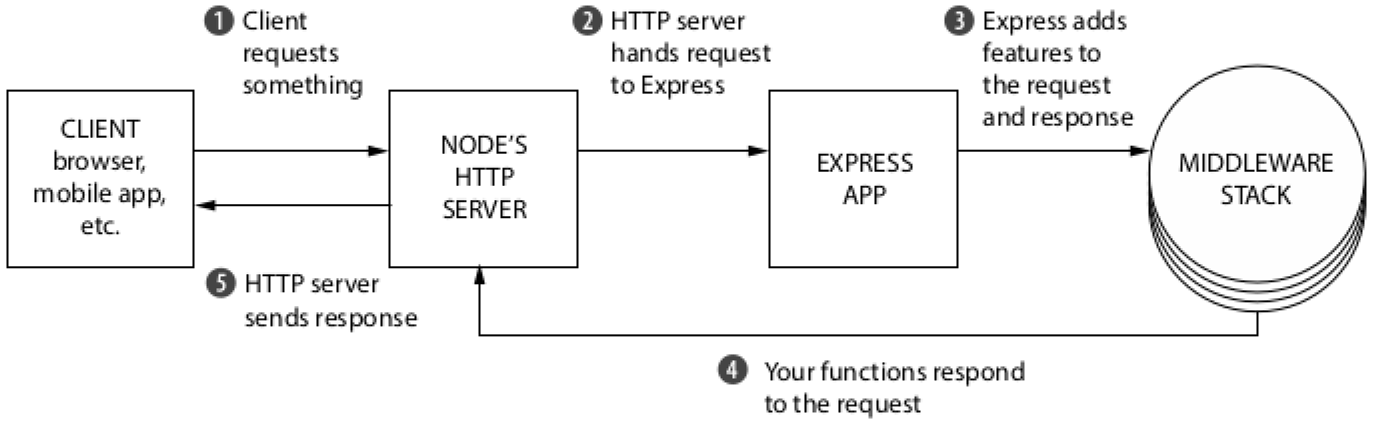
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

<p id="demo"></p>

<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
</script>

</body>
</html>
```

Node and express.js

What is Node and Express

- **Node**: for server-side programming, and primarily deployed for non-blocking, event-driven servers, such as traditional web sites and back-end API services

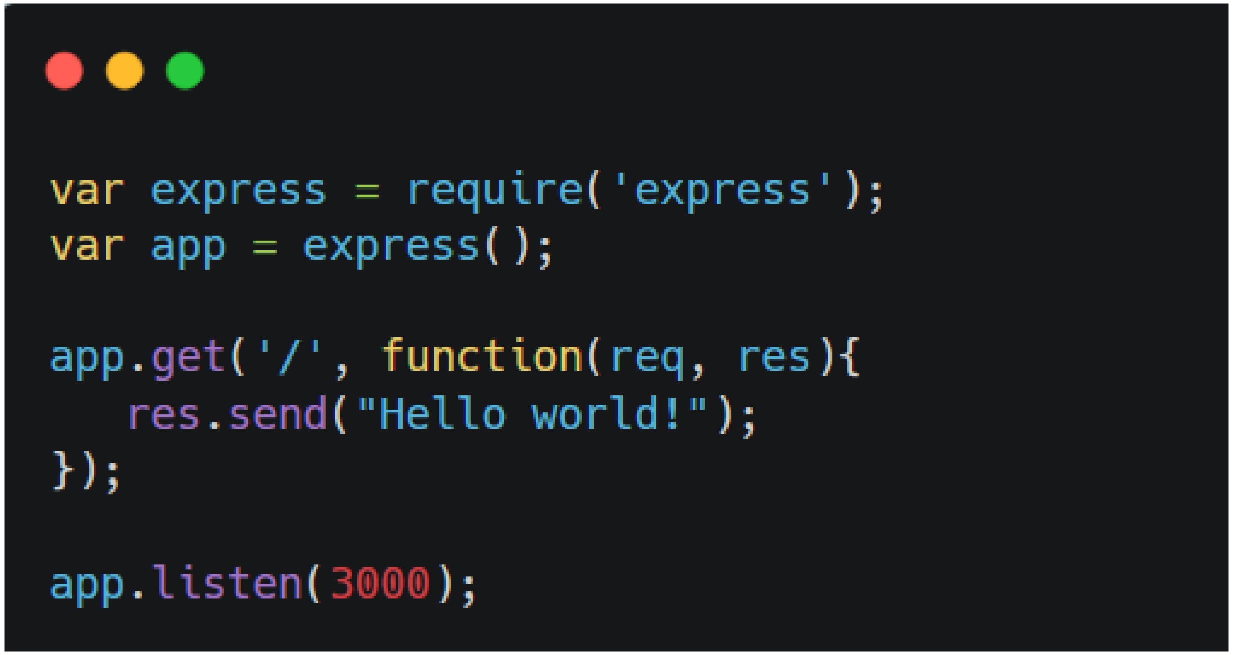
- **Express**: Express.js, or simply Express, is a back end web application framework for building RESTful APIs with Node.js



1 Client requests something
2 HTTP server hands request to Express
3 Express adds features to the request and response
CLIENT browser, mobile app, etc.
NODE'S HTTP SERVER
EXPRESS APP
MIDDLEWARE STACK
5 HTTP server sends response
4 Your functions respond to the request

Setting up a server in Index.js

1. Import express
2. Assign it to app
3. Use app for HTTP methods with given route and the callback function to execute
4. Tell the app to listen on a specific port

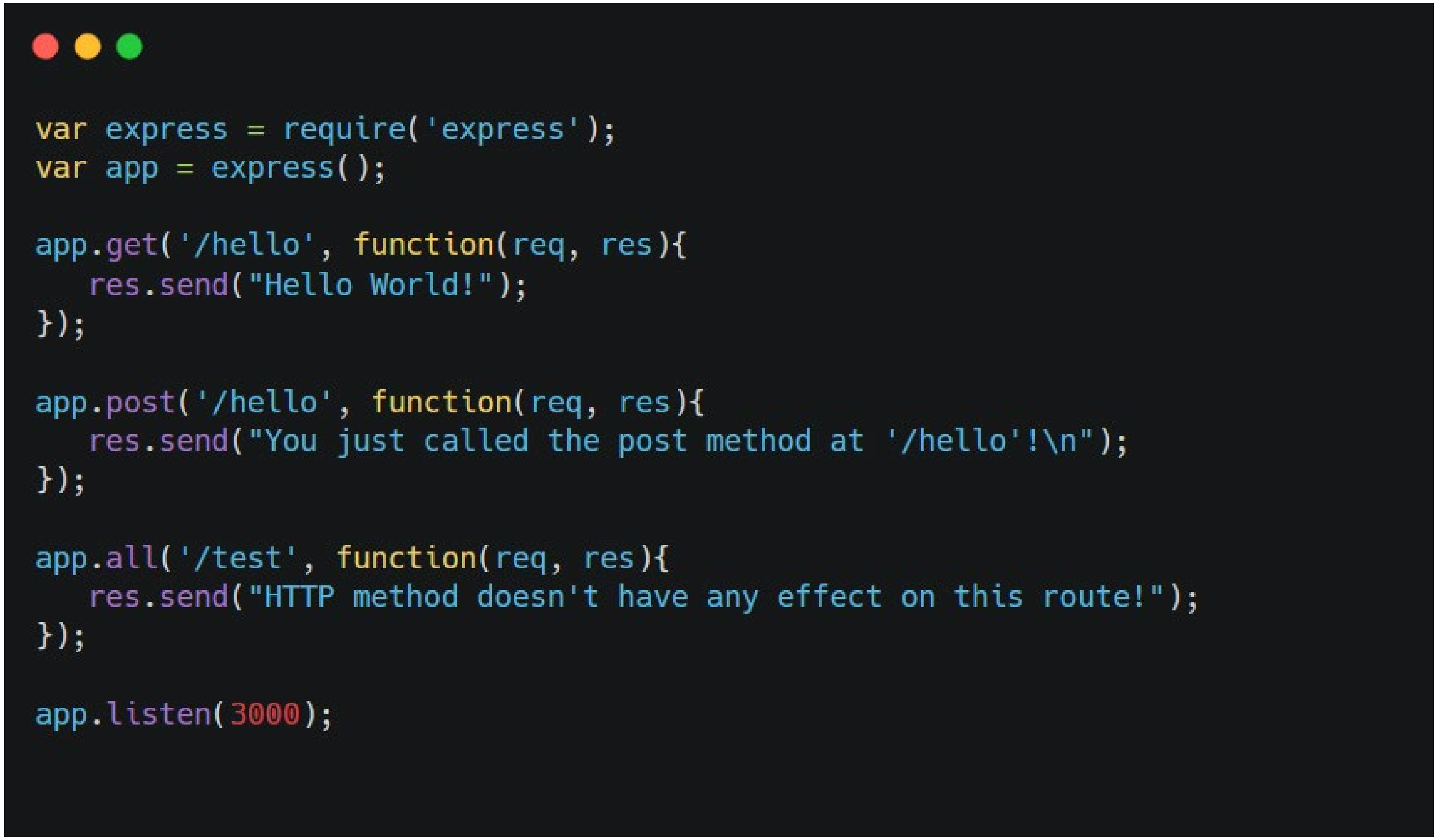


```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   res.send("Hello world!");
});

app.listen(3000);
```

Routing

- **app.method(path, handler)**
  - Path is the route at which the request will run
  - Handler is a callback function

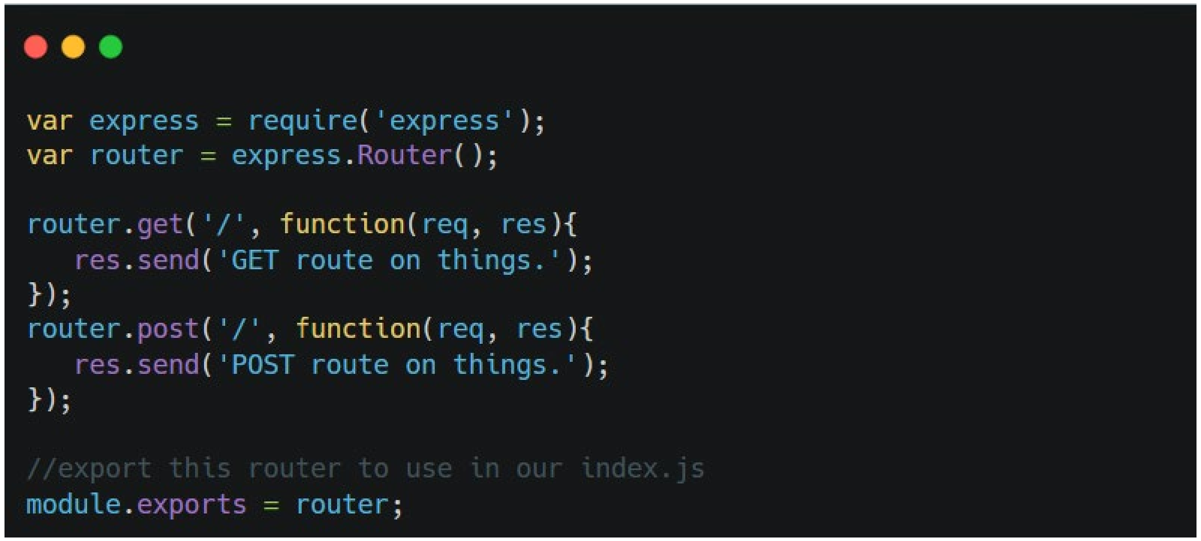
- **app.all** routes all methods to the same request



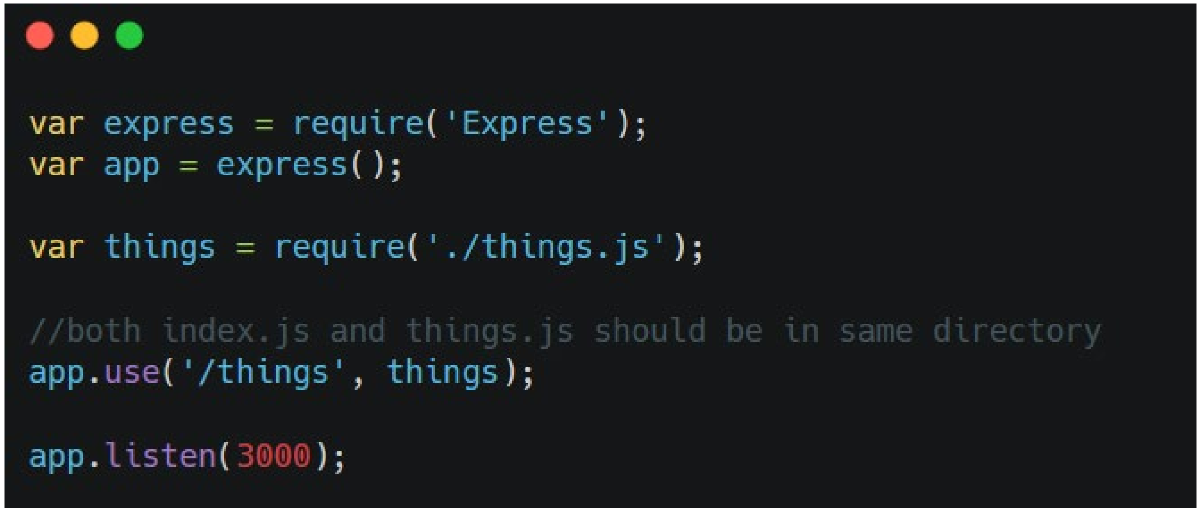
```
var express = require('express');
var app = express();

app.get('/hello', function(req, res){
   res.send("Hello World!");
});

app.post('/hello', function(req, res){
   res.send("You just called the post method at '/hello'!\n");
});

app.all('/test', function(req, res){
   res.send("HTTP method doesn't have any effect on this route!");
});

app.listen(3000);
```

Separating Routing Files

- ***app.use*** function call on route **'/things'**
- attaches the **things** router with this route
- The '/' route in things.js is actually a subroute of '/things'



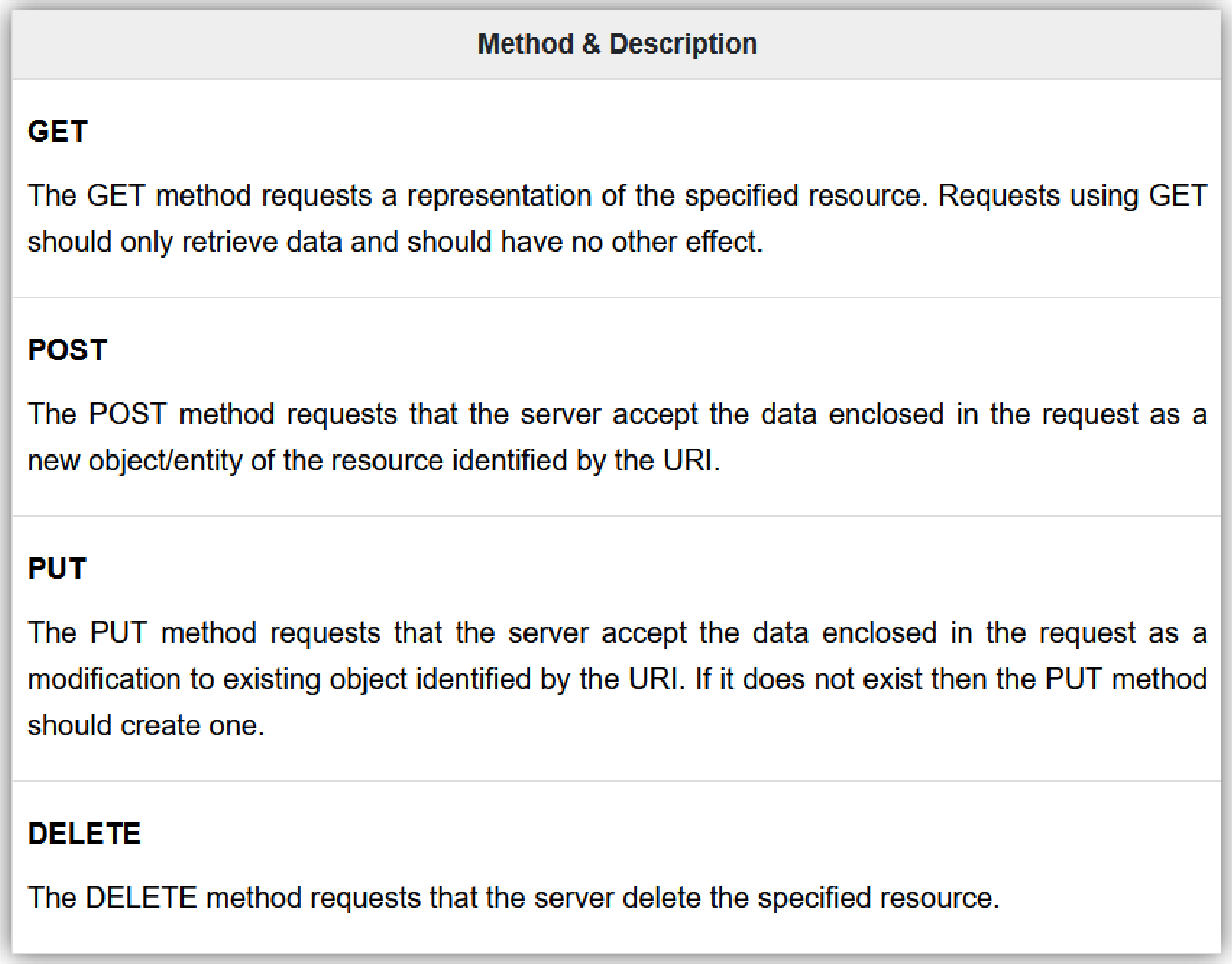
```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res){
   res.send('GET route on things.');
});
router.post('/', function(req, res){
   res.send('POST route on things.');
});

//export this router to use in our index.js
module.exports = router;
```



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

var things = require('./things.js');

//both index.js and things.js should be in same directory
app.use('/things', things);

app.listen(3000);
```

HTTP METHODS

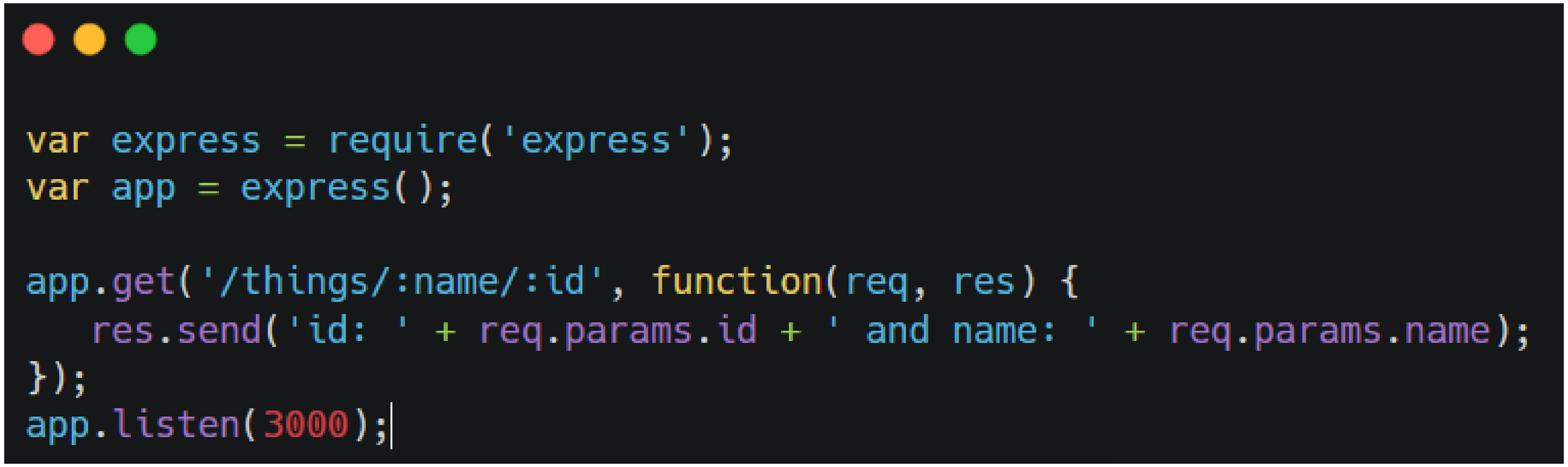
- The HTTP method is supplied in the request and specifies the operation that the client has requested.



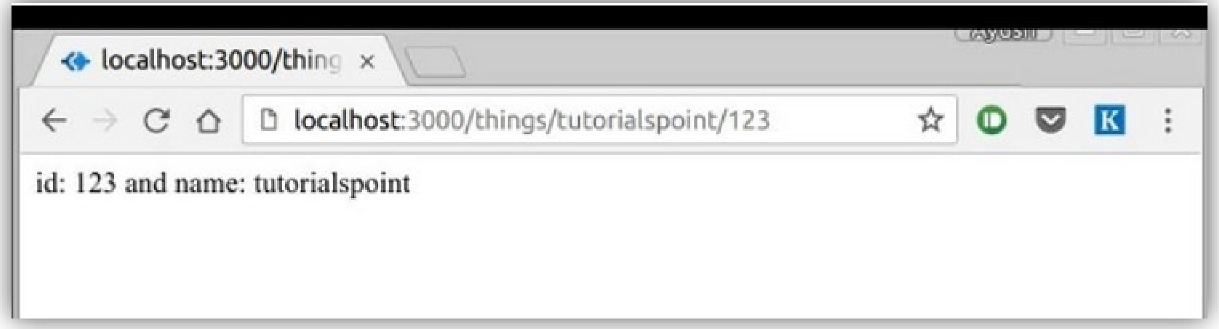
Method & Description
GET The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
POST The POST method requests that the server accept the data enclosed in the request as a new object/entity of the resource identified by the URI.
PUT The PUT method requests that the server accept the data enclosed in the request as a modification to existing object identified by the URI. If it does not exist then the PUT method should create one.
DELETE The DELETE method requests that the server delete the specified resource.

Dynamic Routing

- Using dynamic routes allows us to pass parameters and process based on them



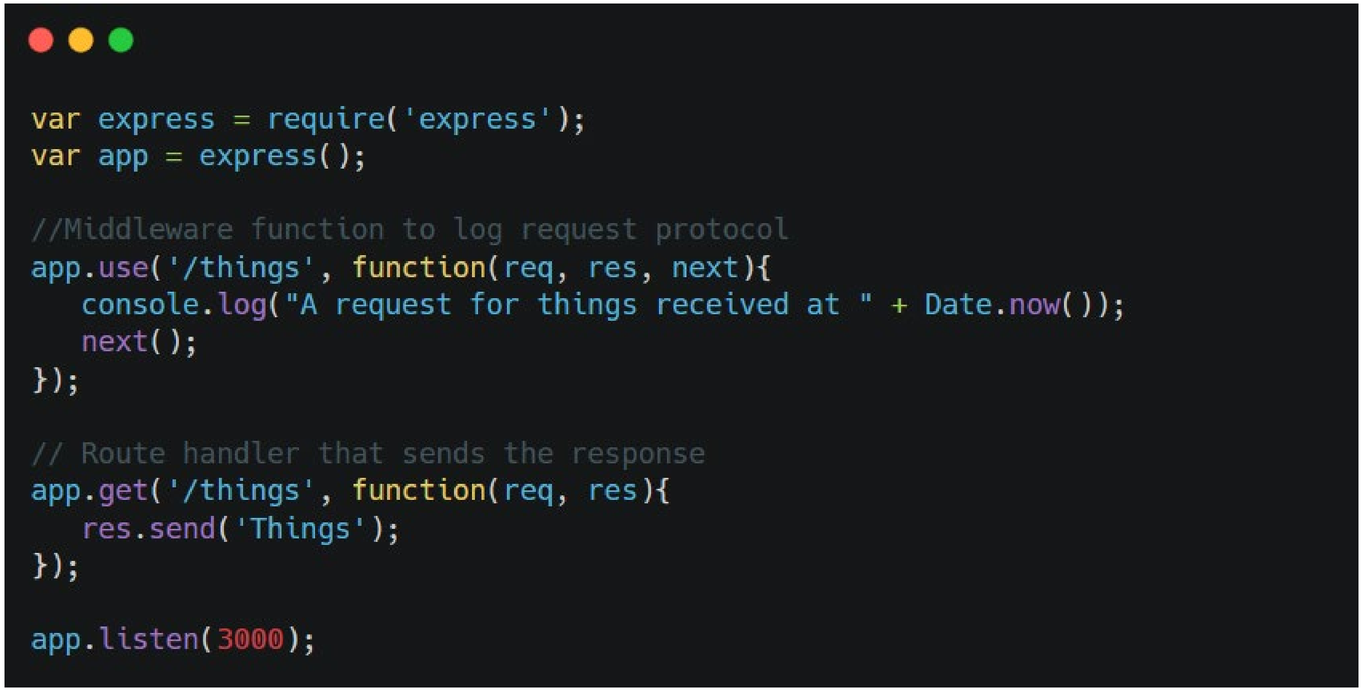
```
var express = require('express');
var app = express();

app.get('/things/:name/:id', function(req, res) {
   res.send('id: ' + req.params.id + ' and name: ' + req.params.name);
});
app.listen(3000);
```



```
id: 123 and name: tutorialspoint
```

Middleware

- Middleware functions have access to the **request object (req)**, the **response object (res)**, and the next middleware function in the application's request-response cycle
- used to modify **req** and **res** objects for tasks like parsing request bodies, adding response headers, etc.



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

//Middleware function to log request protocol
app.use('/things', function(req, res, next){
   console.log("A request for things received at " + Date.now());
   next();
});

// Route handler that sends the response
app.get('/things', function(req, res){
   res.send('Things');
});

app.listen(3000);
```

DEMO TIME!