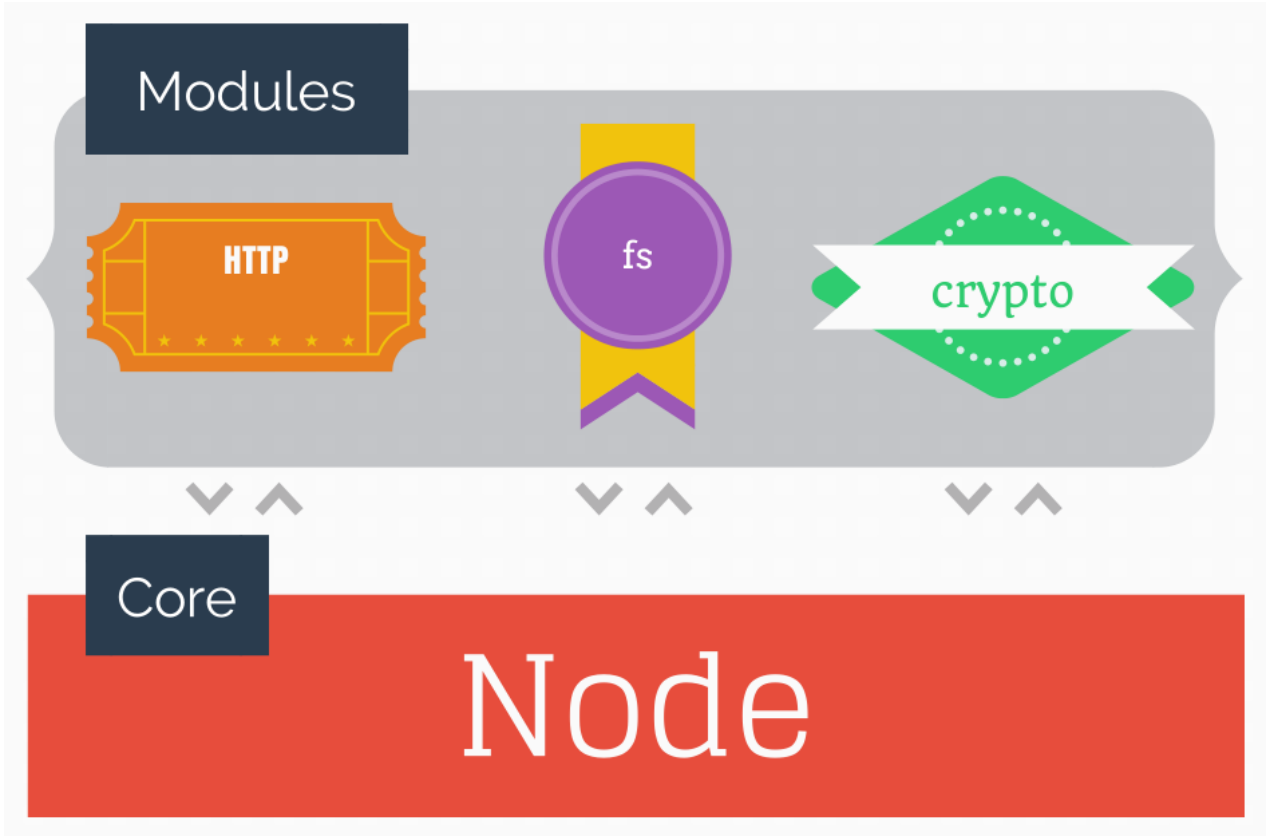
Node.js

Ch 2: Node Module System

hogeschool vives



D. HOSTENS & M. DIMA

— Rules

Attendance in class is not mandatory but **important**!

Remote troubleshooting is very difficult.

- Questions during class: **always**
- Questions after class: **forum**
- **Independent research** (Google, Stack Overflow, AI, etc.)
- Complete the Classroom Labs

— Nodule Module System

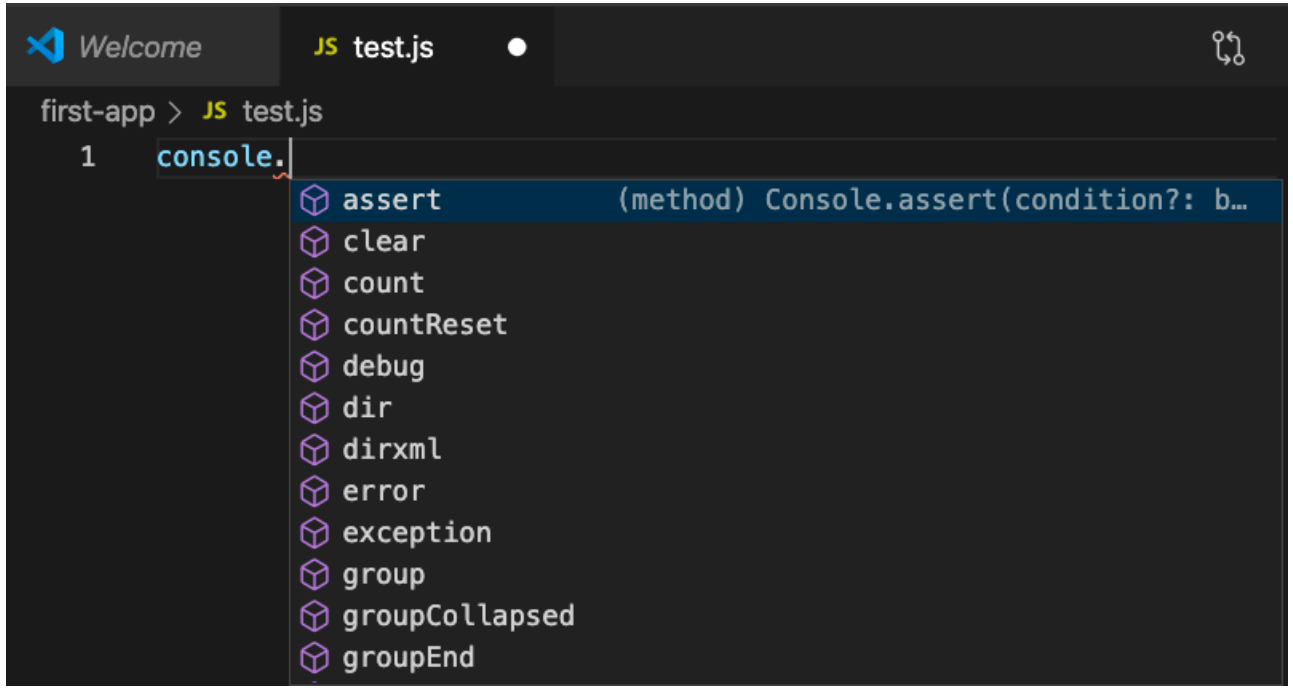
- We've already seen a few: `os, fs, events, http`
- What are they?
- Why do we need them?
- How do we use them?

- We will create our own modules

— Global Object

e.g. console.log => console is a global object and can be called everywhere (in all files)

Intellisense will autofill



— Global Object - cont.

There are other Global Available objects (functions) in Node

Some examples (also work inside the browser):

- `setTimeout()` - call function after delay bv 2 sec
- `clearTimeout()`
- `setInterval()` - repeatedly call the same function with interval
- `clearInterval()` - stop the repeatedly calling of a function
- …

https://nodejs.org/api/globals.html

— Inside the browser

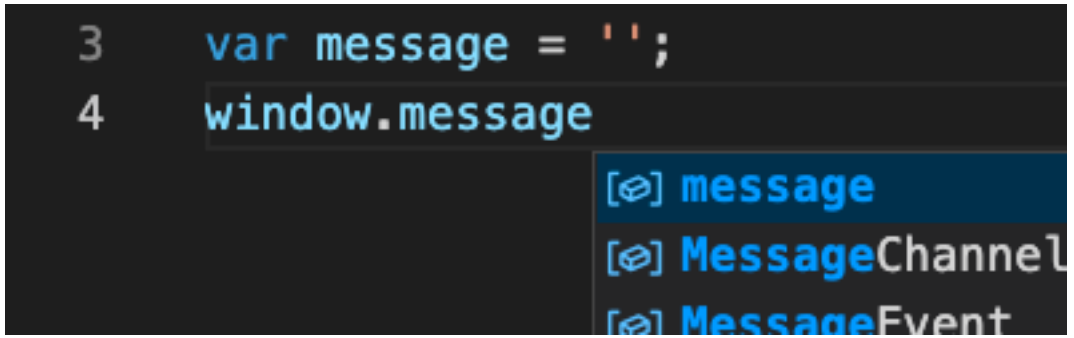
window object (automatically prefixed by JS engine)

e.g. `console.log()` => behind the screens= > `window.console.log()`

`window.setTimeout()` or `setTimeout()`

..

Declared variables are also available inside the window object



— In Node

No `window` object

We do have a `global` object

Will automatically be prefixed to all objects/functions

`global.console.log()` or `global.setTimeout()`

In Node variables are not added to the Global Object!

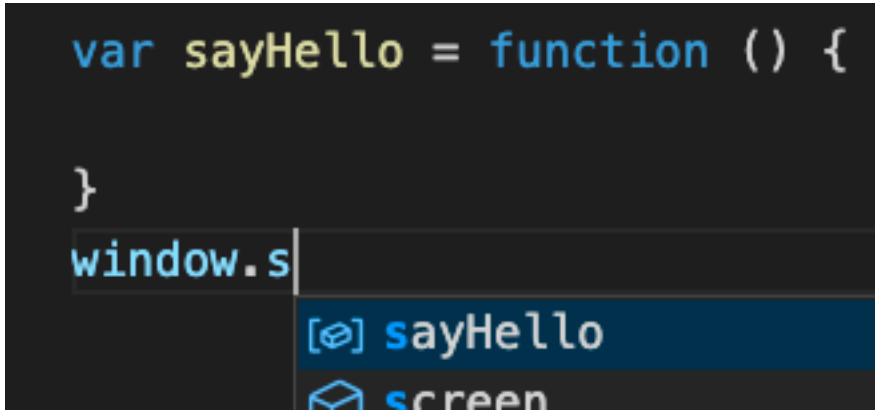
Scope is limited to the files where they are declared e.g. app.js

```
var message = 'test';
console.log(global.message);
```

```
milan@first-app node test.js
undefined
milan@first-app
```

— Global scope (in JS)

In JS functions are added to the global scope and are available trough the `window` object



What if you have different files with identical declarations?

Definitions will be *overridden*

Need of modularity! (Small building blocks = modules)

— Modulariteit

Modularity

Functions and objects = encapsulated in object

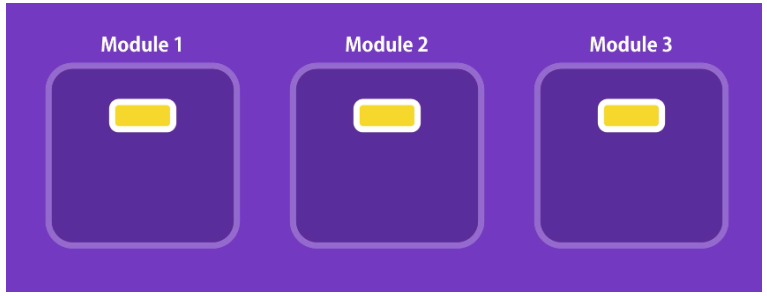
In a Node app = module (each separate file is a module)

Variables declared in a file have the scope of the file

In OOP ≈ private (not accessible outside of the module)

Can be exported

Each Node app has at least 1 file = main module



— Module object (not global!)

```
console.log(module);
```

```
milan@first-app~ node test.js
Module {
  id: '.',
  path: '/Users/milan/Dev/first-app',
  exports: {},
  filename: '/Users/milan/Dev/first-app/test.js',
  loaded: false,
  children: [],
  paths: [
    '/Users/milan/Dev/first-app/node_modules',
    '/Users/milan/Dev/node_modules',
    '/Users/milan/node_modules',
    '/Users/node_modules',
    '/node_modules'
  ]
}
milan@first-app~
```

```
console.log(global.module);
can not!
```

— Create and load Module

We create a new file: logger.js

```
var url = ‘http://mijnlogger.io/log';
function log(message) {
    //send HTTP request
    console.log(message);
}
```

We want to use this logger inside the app

We need to export it first

To do this we add the following line:

```
module.exports.log = log;
```

Exporting Objects/variables (implementation detail)

```
module.exports.logurl = url;
Good practice: don’t export implementation details!
```

— Use the Module in app.js

Load Module with `require` (only in Node!)

In app.js: require('./logger');

```
var logger = require('./logger');
console.log(logger);
```

```
milan@les2ᄊ node app.js
{ log: [Function: log], logurl: 'http://
mijnlogger.io:log' }
milan@les2ᄊ
```

The require functie will return the object

— var or const? Always const where possible!

```
var logger = require('./logger');
logger.log(‘message');
```

```
milan@les2〽 node app.js
message
milan@les2〽
```

```
var logger = require('./logger');
logger = 1;
logger.log('message');
```

```
milan@les2〽 node app.js
/Users/milan/Dev/first-app/les2/app.js:3
logger.log('message');
       ^

TypeError: logger.log is not a function
```

```
Beter=> const logger = require('./logger');
Throws error @ compile instead of @ runtime
```

— Export an Object or function

ℹ️ Tools for error check: jshint (`npm install -g jshint`)

In this case it's better to export the function instead of the full object

Instead of:

```
module.exports.log = log;
```

Use:

```
module.exports = log;
```

in `app.js`

```
const logger = require('./logger');
logger.log('message');
```

Rename to log (`F2` of `fn + F2`)

```
const log = require('./logger');
log('message');
```

— How does Node keep the scope private? => Module Wrapper function

At runtime code will be wrapped in een function wrapper

IIFE (Immediately Invoked Function Expression)

```
(function(exports, require, module, _ __filename,__dirname) { … })
```

require, module, exports not global! Local for each module.

Add Function to exports =>

```
module.exports.log = log;
```
Or `exports.log = log;`

`exports = log;` can not! Ref to module.exports

```
console.log(__filename);
console.log(__dirname);
```

— Builtin Node modules

Ga to: https://nodejs.org/docs/latest/api/ or https://nodejs.org/dist/latest-v18.x/docs/api/

Not everything is a module! You also have objets like Console

Short list! We will discuss a few important ones:

File system

HTTP (create webservers)

OS (work with your Operating System)

Path (work with paths)

Process, QueryStrings (build HTTP services), Stream..

— Path module (built-in module)

https://nodejs.org/dist/latest-v18.x/docs/api/path.html

We use Path.parse(path) https://nodejs.org/dist/latest-v18.x/docs/api/path.html#pathparsepath

```
const path = require('path'); - !! Without ./
```

```
const path = require('path');
var pathObj = path.parse(__filename);
console.log(pathObj);
```

```
milan@les2〽 node pathvb.js
{
  root: '/',
  dir: '/Users/milan/Dev/first-app/les2',
  base: 'pathvb.js',
  ext: '.js',
  name: 'pathvb'
}
milan@les2〽
```

— OS Module

https://nodejs.org/dist/latest-v18.x/docs/api/os.html

Does not work inside the browser! Information about the server!

```
const os = require('os');
var totalMemory = os.totalmem();
var osType = os.type();
console.log('je totale memory is: ' + TotalMemory);
//ECMAScript 6 ES2015 of ES6 Templatestring
console.log(`Je OS type is: ${osType}`);
 Note! Backtick ` not quotes '
milan@les2〽 node osvb.js
je totale memory is: 8589934592
Je OS type is: Darwin
milan@les2〽
```

— File System Module (synchronous)

https://nodejs.org/dist/latest-v18.x/docs/api/fs.html

Work with (be careful with Sync - Async)

Async = non-blocking! Prefered!

```
const fs = require('fs');
const files = fs.readdirSync('./');
console.log(files);
```

```
milan@les2〽 node fsvb.js
[ 'app.js', 'fsvb.js', 'logger.js', 'osvb.js',
'pathvb.js' ]
milan@les2〽
```

File System Module (Async) Always use async!

```
const fs = require('fs');
fs.readdir('./',function(err,files){
    if (err) console.log('Error', err);
    else console.log('Result', files);
})
```

Async method has 2 params: path and a callback function. The callback function is called when the answer is back

Callback function has two params: error and response

Only one of the two contains information!

```
milan@les2〽 node fsvb.js
Result [ 'app.js', 'fsvb.js', 'logger.js',
'osvb.js', 'pathvb.js' ]
milan@les2〽
```

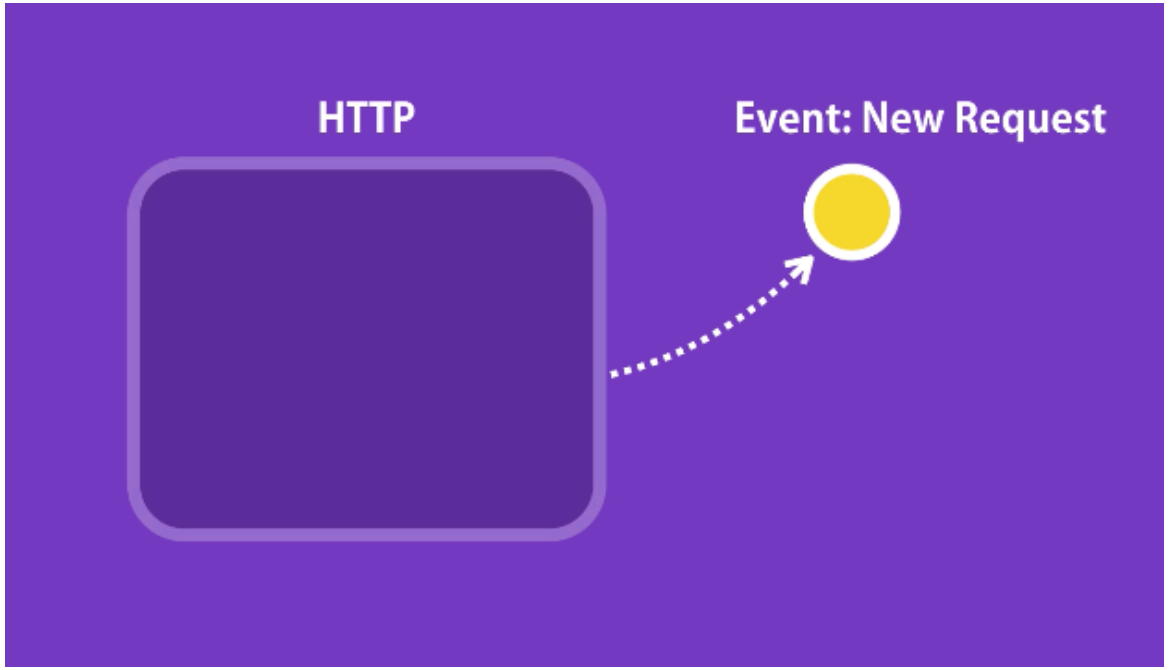
— Events module

Core concept in Node: event

Signals something happend inside the app

E.g. webserver listening to requests

We need to monitor the events and react accordingly



— Events module : cont

https://nodejs.org/dist/latest/docs/api/events.html

EventEmitter convention capital letters -> Class

Class has methods and properties

Order is important! Listener first!

```
const EventEmitter = require('events');
const emitter = new EventEmitter(); //object
//listener
emitter.on('messageLogged', function(){
    console.log('Listener called');
})
//emitter
emitter.emit('messageLogged');
```

— Event Arguments

Sometimes you want to send data with an event, e.g. id and url…

If several values, better to send an object

Callback function receives the arguments

```
const EventEmitter = require('events');
const emitter = new EventEmitter();
emitter.on('messageLogged', function(arg){
    console.log('Listener called', arg);
})
emitter.emit('messageLogged',{id: 1,url:
'http//...'});
```

```
milan@les2〽 node eventargvb.js
Listener called { id: 1, url: 'http//...' }
milan@les2〽
```

— Event Emitter uitbreiden

We rewrite our program: (app + logger)

app.js

```
const EventEmitter = require('events');
const emitter = new EventEmitter();
emitter.on('messageLogged', (arg) => {
    console.log('Listener called', arg);
})
const log = require('./logger');
log('message'); //toont enkel message geen event
```

logger.js

```
const EventEmitter = require('events');
const emitter = new EventEmitter();
var url = 'http://mijnlogger.io:log';
function log(message) {
    //send HTTP request
    console.log(message);
    emitter.emit('messageLogged',{id: 1,url: 'http//...'});
}
module.exports = log;
```

There is No event - only message! Eventlistener not called (app.js) vs obj

— How can we solve this?

No different instances!

Create the same class with all functionality + extra

ES6 class Pascal case convention! Start with capital letter!

Function becomes method

```
class Logger extends EventEmitter{..}
```

By using `extends` the `Logger` class receives all functionality from EventEmitter

`Emitter` becomes `this`

— Solution app.js

```
const EventEmitter = require('events');

const Logger = require('./logger');
const logger = new Logger();

logger.on('messageLogged', (arg) => {
    console.log('Listener called', arg);
})

logger.log('message');
```

— Solution logger.js

```
const EventEmitter = require('events');

var url = 'http://mijnlogger.io:log';
class Logger extends EventEmitter{
    log(message) {
        //send HTTP request
        console.log(message);
        //Raise event
        this.emit('messageLogged',{id: 1,url: 'http//...'});
    }
}
module.exports = Logger;
```

```
milan@les2〽 node app.js
message
Listener called { id: 1, url: 'http//...' }
milan@les2〽
```

— HTTP module

https://nodejs.org/dist/latest-v18.x/docs/api/http.html

`createServer()` is an EventEmitter

`http.Server` class inherits from `net.Server`

```
const http = require('http');
const server = http.createServer(); //is een EventEmitter
server.listen(3000);
console.log('Listening on port 3000 ...');
```

Add Listener

```
server.on('connection',(socket) => {
    console.log('New connection...');
});
```

— Full server

```
const http = require('http');
const server = http.createServer(); //is een EventEmitter
server.on('connection',(socket) => {
    console.log('New connection...');
});
server.listen(3000);
console.log('Listening on port 3000 ...');
```

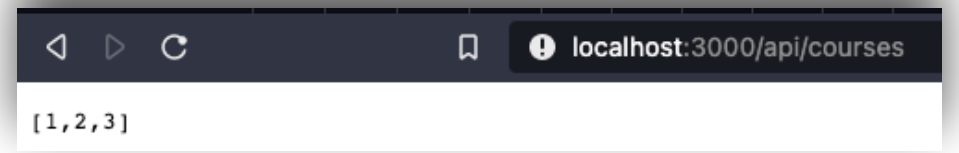
In browser -> go to http://localhost:3000/

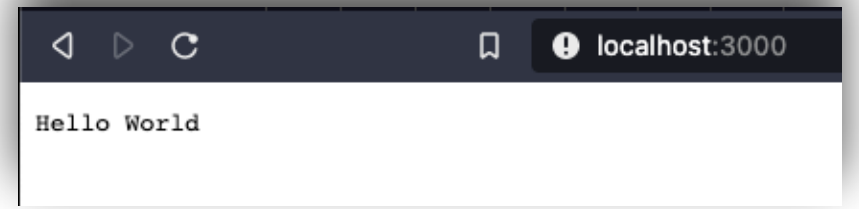
```
milan@les2〽 node httpvb.js
Listening on port 3000 ...
New connection...
```

— Beter with callback in createServer

```
const http = require('http');
const server = http.createServer((req, res) => {
    if (req.url === '/') {
        res.write('Hello World');
        res.end();
    }
    if (req.url === '/api/courses') {
        res.write(JSON.stringify([1, 2, 3]));
        res.end();
    }
});
server.listen(3000);
console.log('Listening on port 3000 …');
```

```
milan@les2〽 node httpvb2.js
Listening on port 3000 ...
```





Assignment Github Classroom

See Toledo for Link