

Common JavaScript by N.W. Knook, The Netherlands, 12.March.2023.

- Explain the difference between "let" and "var".

Var declared variables are 'function-scoped':

- Accessible within the entire function in which its declared
- Can be accessed before they are declared: hoisted to the top of its enclosed function or global scope
- Can be re-assigned multiple times in its scope

Let declared variables are 'block-scoped':

- Only in block its declared in
- Not accessible before its declaration (not hoisted)
- Only re-assignable in its block-scope

Example: 'var-let-example.js'

```
var x = 1;
let y = 2;

function example() {
  var x = 3;
  let y = 4;

  if (true) {
    var x = 5;
    let y = 6;
    console.log(x); // output: 5
    console.log(y); // output: 6
  }

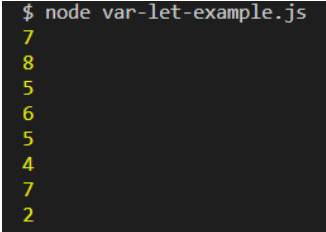
  console.log(x); // output: 3
  console.log(y); // output: 4
}

if (true) {
  var x = 7;
  let y = 8;
  console.log(x); // output: 7
  console.log(y); // output: 8
}

example();

console.log(x); // output: 7
console.log(y); // output: 2
```

Code run in VSC (v 1.76.0) Git-bash terminal:



```
$ node var-let-example.js
7
8
5
6
5
4
7
2
```

- Using modern vanilla JavaScript only, create an Object with some textual key-value properties, create a copy of that object, modify some of its properties and then iterate through this object while outputting the key-value names. Bonus: use more than one iteration approach and explain benefits of one over the other.

Example: 'object-example.js':

```
// Object with some textual values
const decenomyObject = {
  name: "Nick",
  project: "Decenomy 2.0",
  location: "Malta"
}
console.log("decenomyObject: ", decenomyObject);

// Copy the object
const copyOfDecenomyObject = { ...decenomyObject };
console.log("copied: ", copyOfDecenomyObject);

// Modify some properties
copyOfDecenomyObject.name = "Cas";
copyOfDecenomyObject.location = "The Netherlands";

// 1: Iterate through it
for(const [key, val] of Object.entries(copyOfDecenomyObject)) {
  console.log(`1 Entry: ${key} = ${val}`);
}

// 2: Alternative iteration
Object.keys(copyOfDecenomyObject).forEach(key => {
  console.log(`2 Entry: ${key} = ${copyOfDecenomyObject[key]}`);
})

// 3: Another alternative
for(const key in copyOfDecenomyObject) {
  if (copyOfDecenomyObject.hasOwnProperty(key)) {
    console.log(`3 Entry: ${key} = ${copyOfDecenomyObject[key]}`);
  }
}
```

Run in VSC Git-bash terminal:

```
$ node object-example.js
decenomyObject: { name: 'Nick', project: 'Decenomy 2.0', location: 'Malta' }
copied: { name: 'Nick', project: 'Decenomy 2.0', location: 'Malta' }
1 Entry: name = Cas
1 Entry: project = Decenomy 2.0
1 Entry: location = The Netherlands
2 Entry: name = Cas
2 Entry: project = Decenomy 2.0
2 Entry: location = The Netherlands
3 Entry: name = Cas
3 Entry: project = Decenomy 2.0
3 Entry: location = The Netherlands
```

Backend JavaScript

- Explain the difference between "cjs" and "esm".

Basically their differences are the syntax, loading mechanisms and methods to remove unused (dead) code from final code (tree-shaking).

'cjs':

- Means: "CommonJS" used in Node.js (backend side)
- Use `require()` to load modules synchronously
- Use `module exports` to export modules
- Modules run synchronously
- No tree-shaking

'esm':

- Means: "Javascript Modules" used in web browsers (client side)
- Use `import` to load modules asynchronously
- Use `export` to export a module
- Modules run asynchronously
- Support tree-shaking -> more efficient

- Using modern vanilla JavaScript only (node), write a class and an asynchronous getter that awaits 2 seconds before returning "OK" value.
- Include and call it from another file.

Solution (with alternative): 'wait-2-seconds-getter.js':

```
class Wait2SecondsClass {
  async okValue() {
    await new Promise(resolve => setTimeout(resolve, 2000));
    return "OK";
  }
}

function wait2SecondsFunction() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve("OK");
    }, 2000);
  });
}

module.exports = { Wait2SecondsClass, wait2SecondsFunction };
```

and 'wait-2-seconds.js':

```
const {Wait2SecondsClass, wait2SecondsFunction} = require('./wait-2-seconds-getter');

(async () => {
  console.log("1: Start time: " + new Date());
  const wait2seconds = new Wait2SecondsClass();
  const value = await wait2seconds.okValue();
  console.log("1: Result: " + value + ": " + new Date()); // "OK"
})();

(async () => {
  console.log("2: Start time: " + new Date());
  const value = await wait2SecondsFunction();
  console.log("2: Result: " + value + ": " + new Date()); // "OK"
})();
```

Run in VSC Git-bash terminal:

```
$ node wait-2-seconds.js
1: Start time: Sun Mar 12 2023 13:39:24 GMT+0100 (Midden-Europese standaardtijd)
2: Start time: Sun Mar 12 2023 13:39:24 GMT+0100 (Midden-Europese standaardtijd)
1: Result: OK: Sun Mar 12 2023 13:39:26 GMT+0100 (Midden-Europese standaardtijd)
2: Result: OK: Sun Mar 12 2023 13:39:26 GMT+0100 (Midden-Europese standaardtijd)
```

- Using modern JavaScript with Axios library (node), write a simple file downloader which will download a file to current working directory only if its remote size differs from the local one. If it does not - the downloader shall inform about it. Bonus: build this in "esm" standard, fetch the absolute path of the main .js file and pre-set the absolute path of the downloaded file relatively to the main .js file.

=> At (my) domain "dAppIT.nl" I uploaded a file 'loremipsum-234b.txt' file with a 234 bytes content generated at <https://www.lipsum.com/> => <https://dappit.nl/loremipsum-234b.txt>

```
-rw-r--r--  1 dappit.nl psacln    234 mrt 12 14:04 loremipsum-234b.txt
```

Code:

```
const axios = require('axios');
const fs = require('fs');
const path = require('path');
const zlib = require('zlib');

const externalFile = 'https://dappit.nl/loremipsum-234b.txt';

async function downloadFile(url) {
  const downloadedFileName = path.basename(url);
  const response = await axios.head(url); // Send a HEAD request to get the file size
  const remoteGzipped = response.headers['etag']
    ? response.headers['etag'].substr(-5,4) === "gzip"
    : false;
  const remoteSize = parseInt(response.headers['content-length'], 10);
  console.log(`remote ${downloadedFileName} filesize=${remoteSize} bytes`
    + `gzipped=${remoteGzipped}`);
  // Get the absolute path of the downloaded file
  const localFilePath = path.join(path.dirname(require.main.filename),
    downloadedFileName);

  if (fs.existsSync(localFilePath)) {
    const localSize = remoteGzipped
      ? (await new Promise((resolve, reject) => {
        // Gzip the local file and get its size
        const gz = zlib.createGzip();
        const inp = fs.createReadStream(localFilePath);
        const out = fs.createWriteStream(`${localFilePath}.gz`);
        inp.pipe(gz).pipe(out);
        out.on('close', () => {
          const gzippedSize = fs.statSync(`${localFilePath}.gz`).size;
          fs.unlinkSync(`${localFilePath}.gz`);
          resolve(gzippedSize);
        });
      }))
      : fs.statSync(localFilePath).size;

    console.log(`File ${downloadedFileName} exists locally, `
      + `localSize=${localSize} bytes`);
    if (remoteSize === localSize) {
      console.log(`File ${downloadedFileName} already downloaded `
        + `with the same size.`);
      return;
    } else {
      console.log(`File ${downloadedFileName} exists, `
        + `but with different size: download again.`);
    }
  } else {
    console.log(`File ${downloadedFileName} does not exist, download it.`);
  }
}
```

```

const writer = fs.createWriteStream(localFilePath);
const { data } = await axios.get(url, {
  responseType: 'stream',
}); // Send a GET request to download the file

data.pipe(writer); // write (efficient) locally

return new Promise((resolve, reject) => {
  writer.on('finish', resolve);
  writer.on('error', reject);
});
}

downloadFile(externalFile)
  .then(() => { console.log('All done!'); })
  .catch((error) => { console.error('An error occurred:', error); });

```

Install needed libraries in VSC Git-bash terminal:

```

$ npm i axios fs path zlib

added 15 packages, and audited 16 packages in 2s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities

```

Then run:

```

$ node downloader.js
remote loremipsum-234b.txt filesize=170 bytes gzipped=true
File loremipsum-234b.txt does not exist, download it.
All done!

```

Again:

```

$ node downloader.js
remote loremipsum-234b.txt filesize=170 bytes gzipped=true
File loremipsum-234b.txt exists locally, localSize=170 bytes
File loremipsum-234b.txt already downloaded with the same size.
All done!

```

Removed some chars from localfile, run again:

```

$ node downloader.js
remote loremipsum-234b.txt filesize=170 bytes gzipped=true
File loremipsum-234b.txt exists locally, localSize=155 bytes
File loremipsum-234b.txt exists, but with different size: download again.
All done!

```

- With modern vanilla JavaScript only (node), write a "batch script" that will include your previously written file downloader, pre-configure it with 4 different URLs and launch the download process on all of them at once simultaneously (4 threads). When all downloads are done, let the script display a message and exit safely.

At dAppIT.nl extra remote files, filled as previous code:

```
-rw-r--r--  1 dappit.nl psacln      234 mrt  12  14:04 loremipsum-234b.txt
-rw-r--r--  1 dappit.nl psacln      327 mrt  12  16:06 loremipsum-327b.txt
-rw-r--r--  1 dappit.nl psacln      567 mrt  12  16:06 loremipsum-567b.txt
-rw-r--r--  1 dappit.nl psacln      836 mrt  12  16:08 loremipsum-836b.txt
```

Changed 'downloader.js' at end, so it does not run the previous exercise(!):

```
/*downloadFile(externalFile)
  .then(() => { console.log('All done!'); })
  .catch((error) => { console.error('An error occurred:', error); });
*/
module.exports = downloadFile;
```

Created 'download4urls.js':

```
const downloadFile = require('./downloader.js');

// Define the URLs to download
const urls = [
  'https://dappit.nl/loremipsum-234b.txt',
  'https://dappit.nl/loremipsum-327b.txt',
  'https://dappit.nl/loremipsum-567b.txt',
  'https://dappit.nl/loremipsum-836b.txt',
];

// Create an array of promises for all downloads
const promises = urls.map(downloadFile);

// Launch the downloads in parallel
Promise.all(promises)
  .then(() => { console.log('All downloads completed'); })
  .catch(error => { console.error(`Error during download: ${error}`); });
```

Run (due file sizes the parallel threads is not good visible):

```
$ node download4urls.js
remote loremipsum-327b.txt filesize=221 bytes gzipped=true
File loremipsum-327b.txt does not exist, download it.
remote loremipsum-567b.txt filesize=333 bytes gzipped=true
File loremipsum-567b.txt does not exist, download it.
remote loremipsum-234b.txt filesize=170 bytes gzipped=true
remote loremipsum-836b.txt filesize=452 bytes gzipped=true
File loremipsum-836b.txt does not exist, download it.
File loremipsum-234b.txt exists locally, localSize=170 bytes
File loremipsum-234b.txt already downloaded with the same size.
All downloads completed
```

Run again (parallel threads better visible):

```
$ node download4urls.js
remote loremipsum-327b.txt filesize=221 bytes gzipped=true
remote loremipsum-234b.txt filesize=170 bytes gzipped=true
remote loremipsum-567b.txt filesize=333 bytes gzipped=true
remote loremipsum-836b.txt filesize=452 bytes gzipped=true
File loremipsum-327b.txt exists locally, localSize=221 bytes
File loremipsum-327b.txt already downloaded with the same size.
File loremipsum-567b.txt exists locally, localSize=333 bytes
File loremipsum-567b.txt already downloaded with the same size.
File loremipsum-234b.txt exists locally, localSize=170 bytes
File loremipsum-234b.txt already downloaded with the same size.
File loremipsum-836b.txt exists locally, localSize=452 bytes
File loremipsum-836b.txt already downloaded with the same size.
All downloads completed
```

Changed 2 files, run again:

```
$ node download4urls.js
remote loremipsum-234b.txt filesize=170 bytes gzipped=true
remote loremipsum-836b.txt filesize=452 bytes gzipped=true
remote loremipsum-327b.txt filesize=221 bytes gzipped=true
remote loremipsum-567b.txt filesize=333 bytes gzipped=true
File loremipsum-234b.txt exists locally, localSize=143 bytes
File loremipsum-234b.txt exists, but with different size: download again.
File loremipsum-836b.txt exists locally, localSize=452 bytes
File loremipsum-836b.txt already downloaded with the same size.
File loremipsum-327b.txt exists locally, localSize=221 bytes
File loremipsum-327b.txt already downloaded with the same size.
File loremipsum-567b.txt exists locally, localSize=308 bytes
File loremipsum-567b.txt exists, but with different size: download again.
All downloads completed
```

Dear Nick,

Nice exercises! Love to create and run 😊.

Regards,