# Node.js
# Callbacks & promises

# Asynchronous code

- Είδαμε μέχρι τώρα ότι ο ασύγχρονος κώδικας μας επιτρέπει να μεταφέρουμε εργασίες που χρειάζονται αρκετό χρόνο στο background

- Έτσι δεν μπλοκάρετε η εκτέλεση του υπόλοιπου κώδικα

- Ασύγχρονος κωδικός - > Non - blocking κώδικας

# From Synchronous -> Asynchronous

Στο προηγούμενο παράδειγμα είδαμε πως διαβάζουμε ένα αρχείο ασύγχρονα χρησιμοποιώντας **callback_functions**

Γενικότερα μια callback function καθορίζει τι θα συμβεί σε περίπτωση αποτυχίας ή επιτυχίας μετά από μια ενέργεια

Πχ readFile callback function : καλείται μετά την ανάγνωση ενός αρχείου.

Λαμβάνει δύο παραμέτρους:

- err: Εάν παρουσιαστεί κάποιο σφάλμα
- data: Τα περιεχόμενα του αρχείου

# Callback hell

Το callback hell αποτελεί ένα μεγάλο πρόβλημα από πολλά εμφωλευμένα callback functions

Σε τέτοιες περιπτώσεις, το κάθε callback δέχεται ως όρισμα το αποτέλεσμα του παραπάνω callback κ.ο.κ.

Δημιουργείται με αυτόν τον τρόπο, η δομή του κώδικα που μοιάζει με *πυραμίδα*, καθιστώντας έτσι δύσκολη την ανάγνωση και τη συντήρησή του.

# Callback hell

Φανταστείτε να υπάρχουν 10 επίπεδα εμφώλευσης…

```
se > JS callbackhell.js > ...
const filesystem=require('fs');
// readFileSync takes as arguments the filepath and the character encoding

filesystem.readFile('./txt/not_embedded.txt', 'utf8', (err, data)=>{
    if(err) return console.log("oups!!!")
    //not_embedded.txt contains the word embedded, that is the name of the second file
    //${data} = + data +
    filesystem.readFile(`./txt/${data}.txt`, 'utf8', (err, data1)=>{

        // Display the file content
        console.log(data1);
        console.log(err);

        filesystem.writeFile(`./txt/newfile.txt`,data1, (err) => {
            if (err) return console.log ('Could not write file 😢');
          });
    });

});

console.log('Reading file....');
```

# Callback hell

```
ise > JS callbackhell.js > filesystem.readFile('./txt/not_embedded.txt', 'utf8') callback
const filesystem=require('fs');
// readFileSync takes as arguments the filepath and the character encoding

filesystem.readFile('./txt/not_embedded.txt', 'utf8', (mpa, kalaphge)=>{
    filesystem.readFile(`./txt/${kalaphge}.txt`, 'utf8', (err, data1)=>{
        filesystem.writeFile(`./txt/newfile.txt`,data1, (err) => {
            filesystem.readFile(`./txt/newfile.txt`, 'utf8', (err, data1)=>{
                filesystem.readFile(`./txt/newfile.txt`, 'utf8', (err, data1)=>{
                    filesystem.readFile(`./txt/newfile.txt`, 'utf8', (err, data1)=>{
                        console.log(data1);
                    });
                });
            });
        });
    });

});

console.log('Reading file....');
```

# Promises

- Για να επιλύσουμε το παραπάνω πρόβλημα, πρέπει να απαλλαγούμε από τα callbacks.

- Εδώ είναι που τα λεγόμενα Promises μπαίνουν στην εικόνα.

- Τα promises αποτελούν αντικείμενα για τον χειρισμό μιας ασύγχρονης λειτουργίας
- Ένα promise αντιπροσωπεύει το **γεγονός** ότι μια ασύγχρονη λειτουργία
  - είτε θα ολοκληρωθεί επιτυχώς (fullfilled)
  - είτε θα αποτύχει (rejected)

  Pending fulfilled rejected

# Promises

- Σκεφτείτε το ως μια «υπόσχεση» ότι κάτι θα συμβεί αργότερα (μπορεί να γίνει μπορεί και όχι!)

- .then() function καλείται όταν μια υπόσχεση ολοκληρώνεται επιτυχώς
- .catch() στην αντίθετη περίπτωση

# Promises

- The fsPromises.readFile() method is used to Asynchronously read the entire contents of a file.

- Syntax:
- fsPromises.readFile( path, options )
- path: It holds the name/path of file to read
- options: holds encoding of file ( default ‘utf8’)

- It returns a Promise : Promise is resolved with the contents of the file

# Lets see an example

```
// Include fs module
var fs = require('fs');

// initial promise
fs.promises.readFile(__dirname+'/txt/not_embedded.txt')
.then( data => {
  console.log(""+data);
}).catch(error => {
   console.log("what is your problem?"+error);
});
console.log("hello everyone");
```

# Promises

- Σε αντίθεση με τα callback functions, τα promises μπορούν να ενωθούν το ένα με το άλλο σαν αλυσίδα

- Nested promises: ξεκινούν με μια .then() και σε καθεμία από τις .then() έχουμε **return**
- Μετά το return το επόμενο .then() ακολουθεί με τον ίδιο τρόπο.
- Το επόμενο βήμα στην αλυσίδα δεν εκτελείται μέχρι να “επιλυθεί” το προηγούμενο βήμα.

# Promises

*then* takes as argument a function that runs when the promise is **resolved**, and receives the result.

```
// Include fs module
var fs = require('fs');

// to read the file
fs.promises.readFile(__dirname+'/txt/not_embedded.txt')
.then( data => {
  console.log(""+data);
  return fs.promises.readFile(__dirname+`/txt/${data}.txt`)
  //data1 is the value returned from above
}).then(data1 => {
    console.log(""+data1);
    return fs.promises.writeFile(`./txt/newfile.txt`,data1);
}).catch(error => {
   console.log("what is your problem?"+error);
});
console.log("hello everyone");
```

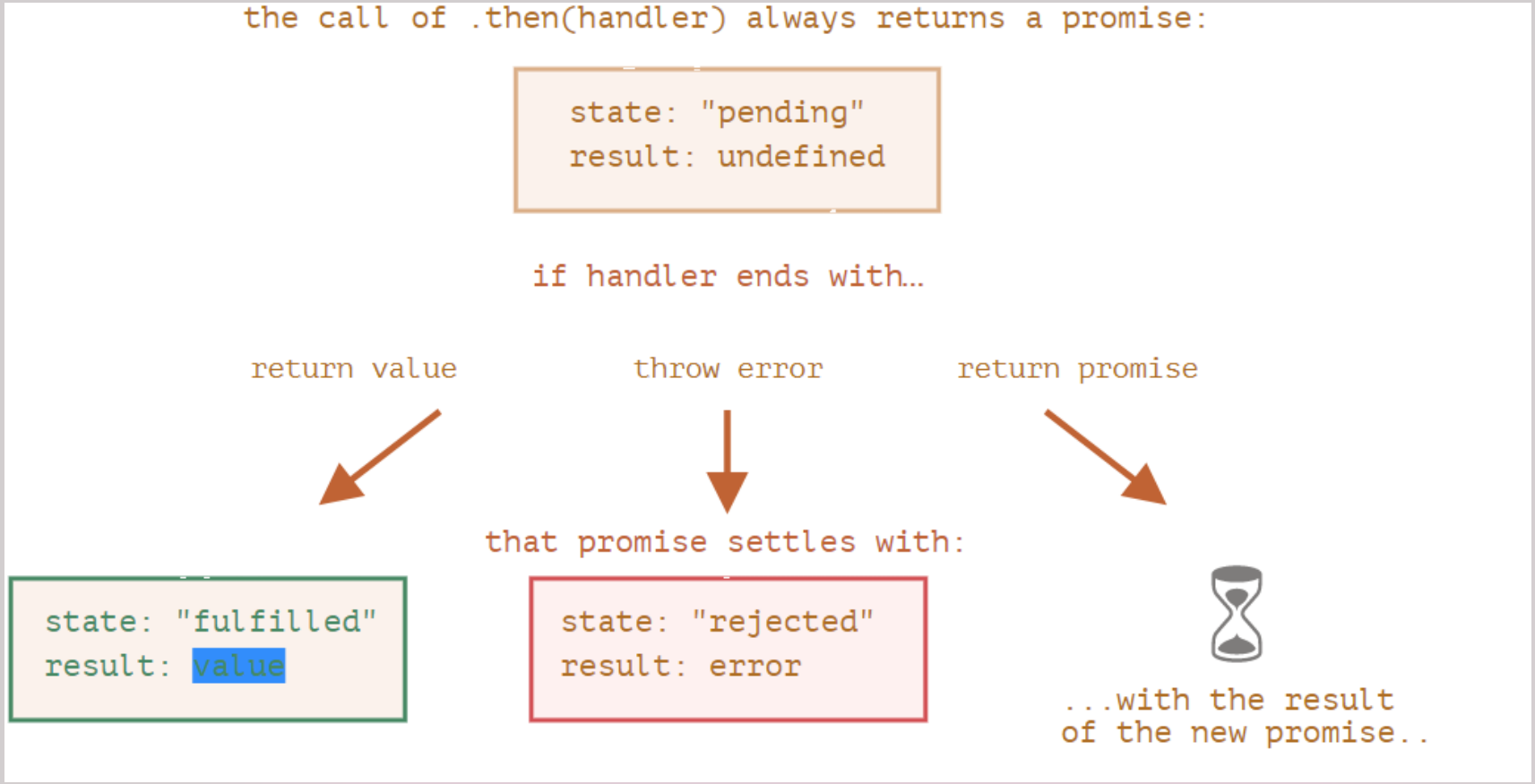


Image source: https://javascript.info/promise-chaining

# Promises

- Μπορούμε ακόμη να φτιάξουμε custom promises χρησιμοποιώντας τον Promise constructor

- var promise = new Promise(function(resolve, reject){});

- Promise constructor -> παίρνει ως ορίσματα ένα callback function
  - Callback function -> παίρνει ως ορίσματα: resolve ,reject

# async/await

“The async and await keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.”

Source: https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Statements/async_function

# async/await

Οπότε **δηλώνουμε μια συνάρτηση ως async** και με τη λέξη-κλειδί **await αναβάλλουμε** την εκτέλεση του κώδικα.

**Προσέξτε** ότι μέσα σε μια async συνάρτηση μπορούμε να έχουμε ένα ή περισσότερα await

Αντί για then έχουμε async… κάνουμε έτσι τον κώδικα πιο όμορφο

omise > JS async.js > ...

```
const fs = require('fs');

(async () => {
  try {
    const data = await fs.promises.readFile(__dirname+'/txt/not_embedded.txt')
    console.log(""+data);

    const data1 = await fs.promises.readFile(__dirname+`/txt/${data}.txt`)

    await fs.promises.writeFile(`./txt/newfile.txt`,data1);

    console.log("Yeap everything went well");
  } catch (err) {
    console.log(err);
    throw err;
  }
})();

```

Async functions:

- are accessible natively in Node: **async keyword** for declaration
- they always return a promise

await keyword is currently restricted to async functions (cannot be used in the global scope)

# Promise.all

- Η μέθοδος Promise.all() λαμβάνει ως input έναν αριθμό promises και επιστρέφει ένα μόνο Promise.

- Αυτή η επιστρεφόμενη *υπόσχεση* εκπληρώνεται όταν εκπληρωθούν όλες οι υποσχέσεις της εισόδου με array από τις τιμές εκπλήρωσης.

To be continued…