STW [Problemes]

node.js

Antonio Hurtado (NIU: 1358933)
Paula Sarqui (NIU: 1368449)

Aquesta entrega no ha sigut una fita d’una sola persona sinó que hi hem col·laborat dues persones per tal de portar-la a terme. El codi l’hem anat penjant a un repositori de lliure accès (GitHub) al que a continuació us facilitem l’enllaç:

Problemes STW

Accedint al repositori no només us podreu descarregar el codi en zip i llest per executar sinó que, a més podreu anar veient el progrès que hem fet per arribar a les diferents solucions.

Índex

Exercici 1

```
let f1 = function (param) {
    console.log(param);
};

f1(3);
```

Exercici 2

```
let f2 = function (a) {
    if (a >= 0){
        a=2*a
    }else{
        a=-1
    }
    return a
};

console.log(f2(3));
console.log(f2(-1));
```

Exercici 3

```
let f3 = function (llista) {
    let llista2 = [];

    for (let key in llista) {
        llista2[key] = llista[key]+23;
    }

    return llista2;
};

let l = f3([1,2,3]);
console.log(l[0]+' '+l[1]+' '+l[2]);
```

Exercici 4

```
console.printaki = () => {
    console.log('aqui');
};

console.printaki();
```

Exercici 5

```
let f4 = function (a,b) {
    return a+b;
};

let llistaA = [1,2,3,4];

let llistaB = llistaA.map(function (val) {
    return f4(val, 23);
});

console.log(llistaB[0]+' '+llistaB[1]+' '+llistaB[2]+' '+llistaB[3]);
```

Exercici 6

```
let f2 = function (a) {
    if (a >= 0){
        a=2*a
    }else{
        a=-1
    }
    return a
};

function f5(a,b,callback) {
    //a és un objecte, b és una funció, i c és una funció
    callback(b(a));
}

f5(1, f2, function(r) {
    console.log(r);
});
```

Exercici 7

```
let f7 = function () {
    let count = 1;
    console.printaki2 = () =>{
        console.log('aqui '+ count);
        count++;
    }
};

f7();

console.printaki2();
console.printaki2();
console.printaki2();
```

Exercici 8

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

let f6 = function (llista, callback) {
    let result = [];

    let readFiles = new Promise((resolve, reject) => {
        llista.forEach(function (element, index, array) {
            fs.readFile('./' + element, 'utf8', function (err, data) {
                if (err) {
                    throw err;
                }
                console.log('data: ' + data);
                result.push(data);

                if (index === array.length - 1) {
                    resolve();
                }
            });
        });
    });

    readFiles.then(() => {
        callback(result);
    });
};

f6(['a1.txt','a2.txt'],function (result) {
    console.log(result)
});
```

Exercici 9

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

f6 = function (llista, callback) {
   let result = [];
   let basePath = './';

   let readFiles = new Promise((resolve, reject) => {
      llista.forEach(function (element, index, array) {
         fs.readFile(basePath + element, 'utf8', function (err, data) {
            if (err) {
               throw err;
            }
            console.log('data: ' + data);
            result[index] = data;

            if (index === array.length - 1) {
               resolve();
            }
         });
      });
   });

   readFiles.then(() => {
      callback(result);
   });
};

f6(['a1.txt','a2.txt'],function (result) {
   console.log(result)
});
```

Exercici 10

```
/**
 * Tanto la función 'array.forEach()' como la función 'fs.readFile()' son funciones asíncronas
 * por lo que, de implementar la variable 'index' de esa forma podría desembocar en un problema
 * de sincronicidad a la hora de hacer los accesos o las asignaciones a los arrays, causando así
 * que no se produzca un resultado consistente y/o correcto.
 */
```

Exercici 11

```
const fs = require('fs');
const {StringDecoder} = require('string_decoder');
const decoder = new StringDecoder('utf8');

function asyncMap (list, f, callback2) {
    let basePath = './';
    let resultList = [];
    let err = null;
    //...
    let map = new Promise((resolve, reject) => {

        list.map((fileName, index, array) => {f(basePath + fileName, function (err, result) {
            if (err != null) {
                callback2(err, resultList);
                throw err;
            }

            resultList[index] = decoder.write(result);

            if (index === array.length - 1) {
                decoder.end();
                resolve({'err' : err, 'resultList' : resultList});
            }
        })});
    });

    map.then((vector) => {
        callback2(vector['err'], vector['resultList']);
    });
}

asyncMap(['a1.txt'], fs.readFile, function (a, b) {
    console.log(b);
});
```

Exercici 12

```
let O = function () {
    this.count = 0;
    this.notify = null;

    this.inc = function () {
        this.count++;

        if ((this.notify != null) && (this.notify instanceof Function)) {
            this.notify(this.count);
        }
    };
};

o1 = new O();

o1.count = 1;
o1.notify = function (a) {
    console.log(a);
};

o1.inc();
```

Exercici 13

```
let o2 = (function () {
    let count = 1;
    let notify = null;
    return {
        inc: function () {
            count++;

            if (notify != null) {
                notify(count);
            }
        },
        count: function () {
            return count;
        },
        setNotify(value) {
            if (value instanceof Function) {
                notify = value;
            }
        }
    };
})();

o2.setNotify(function (a) {
    console.log(a);
});
o2.inc();
```

Exercici 14

```
function Counter() {
    this.a = 1;
    this.notify = 1;

    this.inc = (function () {
        this.a++;
        if ((this.notify != null) && (this.notify instanceof Function)) {
            this.notify(this.a);
        }
    });

    this.count = (function () {
        return this.a;
    });

    this.setNotify = (function (value) {
        if (value instanceof Function) {
            this.notify = value;
        }
    });

}

let o3 = new Counter();
o3.setNotify(function (a) {
    console.log(a);
});
o3.inc();
```

Exercici 15

```
function Counter() {
    this.a = 1;
    this.notify = 1;

    this.inc = (function () {
        this.a++;
        if ((this.notify != null) && (this.notify instanceof Function)) {
            this.notify(this.a);
        }
    });

    this.count = (function () {
        return this.a;
    });

    this.setNotify = (function (value) {
        if (value instanceof Function) {
            this.notify = value;
        }
    });

}

let o3 = new Counter();
o3.setNotify(function (a) {
    console.log(a);
});
o3.inc();
```

Exercici 16

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

let Future = (function (result) {
    this.result = result || null;
    this.isDone = this.result != null;

    return {
        isDone: this.isDone,
        result: this.result,
    };
});

let future = new Future();

function readIntoFuture(fileName) {
    let basePath = './';

    fs.readFile(basePath + fileName, 'utf8', function (err, data) {
        if (err) {
            throw err;
        }

        future = new Future(data);

    });

    return new Future();
}

future = readIntoFuture('a1.txt');

console.log(future);

setTimeout(function () {
    console.log(future);
}, 1000);
```

Exercici 17

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

let Future = (function (result) {
    this.result = result || null;
    this.isDone = this.result != null;

    return {
        isDone: this.isDone,
        result: this.result,
    };
});

let future = new Future();

function asyncToFuture(f) {
    return function (fileName) {
        f(fileName, function (err, data) {
            if (err) {
                throw err;
            }

            future = new Future(data);
        });
        return new Future();
    }
}

let readIntoFuture2 = asyncToFuture(fs.stat);
future = readIntoFuture2('a1.txt');
setTimeout(function () {
    console.log(future);
}, 1000);
```

Exercici 18

```
const fs = require('fs');
const {StringDecoder} = require('string_decoder');
const decoder = new StringDecoder('utf8');

function asyncToEnhancedFuture(action) {
    let registeredCallback = null;
    let enhancedFuture = {
        isDone: false,
        result: null,
        registerCallback: (callback) => {
            if (enhancedFuture.isDone) {
                callback(enhancedFuture);
            } else {
                registeredCallback = callback;
            }
        }
    };
    return (args) => {
        action(args, (error, result) => {
            enhancedFuture.isDone = true;
            enhancedFuture.result = decoder.write(result);

            if (registeredCallback !== null) {
                registeredCallback(enhancedFuture);
            }

            decoder.end();
        });
        return enhancedFuture;
    };
}

// Test the thing...
let readIntoEnhancedFuture = asyncToEnhancedFuture(fs.readFile);
let enhancedFuture = readIntoEnhancedFuture('a1.txt');
enhancedFuture.registerCallback(function(ef) {
    console.log(ef);
});
```

Exercici 19

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

when = function (f1){
    if (!(this instanceof when)) return new when(f1);
    this.do = function (f2) {
        f1(f2);
    }
};

let f1 = function (callback) {
    fs.readFile('a1.txt', callback);
};
let f2 = function (error, result) {
    console.log(result.toString());
};

when(f1).do(f2);
```

Exercici 20

```
const fs = require('fs');
const {StringDecoder} = require('string_decoder');
const decoder = new StringDecoder('utf8');

when = function (f){
   this.promises = [];
   this.results = {};

   this.promises.push(new Promise((resolve, reject) => {
      f (function (err, data) {
         if (err) {
            throw err;
         }

         this.results['error1'] = err;
         this.results['result1'] = decoder.write(data);

         resolve();
      });
   }));

   this.and = (function (g) {
      this.promises.push(new Promise((resolve, reject) => {
         g(function (err, data) {
            if (err) {
               throw err;
            }

            this.results['error2'] = err;
            this.results['result2'] = decoder.write(data);

            decoder.end();
            resolve();
         });
      }));

      return this;
   });

   this.do = (function (h) {
      Promise.all(this.promises).then(() => {
         h(this.results['error1'], this.results['error2'], this.results['result1'], this.results['result2']);
      });
   });

   return this;
};

f1 = function(callback) {
   fs.readFile('a1.txt', callback)
};

f2 = function(callback) {
   fs.readFile('a2.txt', callback);
};

f3 = function(err1, err2, res1, res2) {
   console.log(res1, res2);
};

when(f1).and(f2).do(f3);
```

Exercici 21

```
let composer = function (f, g) {
    return function (x) {
        return f(g(x));
    }
};

let f1 = function (a) {
    return a + 1;
};

let f3 = composer(f1, f1);

console.log(f3(3));

let f4 = function (a) {
    return a * 3;
};

let f5 = composer(f3, f4);

console.log(f5(3));
```

Exercici 22

```
let asyncComposer = function (f1, f2) {
    return function (x, f) {
        f2(x, function (error, res) {
            if (error != null) {
                throw error;
            }

            f1 (res, function (err, data) {
                if (err) {
                    throw err;
                }

                f(err, data);
            });
        });
    };
};

let f1 = function (a, callback) {
    callback(null, a + 1);
};

let f3 = asyncComposer(f1, f1);

f3 (3, function (error, result) {
    console.log(result);
});

f1 = function(a, callback) {
    callback(null, a + 1)
};

f2 = function(a, callback) {
    callback("error", "")
};

f3 = asyncComposer(f1, f2);

f3(3, function(error, result) { console.log(error, result) } );
```

Exercici 23

```
let p;

/**
 * Apartat A
 * @type {Promise<number>}
 */
p = Promise.resolve(0).then(x => x + 1).then(x => x + 2).then(x => x + 4);
p.then(x => console.log('Apartat A: ' + x));

/**
 * La Promesa es resol retornant un 0, al primer 'then' tenim que:
 * x = 0
 * y, a més a això li sumem 1, amb el que ans queda:
 * x = x + 1 = 1
 * Al segon 'then' fem:
 * x = x + 2 = 1 + 2 = 3
 * Al tercer i últim 'then' fem:
 * x = x + 4 = 3 + 4 = 7
 * Finalment tenim que:
 * x = 7
 */

/**
 * Apartat B
 * @type {Promise<never>}
 */
p = Promise.reject(0).then(x => x + 1).catch(x => x + 2).then(x => x + 4);
p.then(x => console.log('Apartat B: ' + x));

/**
 * La promesa es rebutja retornant un 0, com la promesa ha sigut rebutjada ens saltem el 'then', amb el que tenim que:
 * x = 0
 * Al catch, com la promesa va ser rebutjada, no l'ignorem i fem:
 * x = x + 2 = 0 + 2 = 2
 * Al 'then' després del catch no l'ignorem i fem:
 * x = x + 4 = 2 + 4 = 6
 * Amb el que tenim que:
 * x = 6
 */

/**
 *
 */

/**
 * Apartat C
 * @type {Promise<number>}
 */
p = Promise.resolve(0).then(x => x + 1).then(x => x + 2).catch(x => x + 4).then(x => x + 8);
p.then(x => console.log('Apartat C: ' + x));

/**
 * La Promesa es resol retornant un 0, al primer 'then' tenim que:
 * x = 0
 * i, a més a això li sumem 2, amb el que ans queda:
 * x = x + 1 = 1
 * Al segon 'then' fem:
 * x = x + 2 = 1 + 2 = 3
 * Al catch, com la promesa va ser resolta, l'ignorem i no el fem.
 * Al tercer i últim 'then' fem:
 * x = x + 8 = 3 + 8 = 11
 * Finalment tenim que:
 * x = 11
 */


/**
 * Apartat D
 * @type {Promise<never>}
 */
p = Promise.reject(0).then(x => x + 1).then(x => x + 2).catch(x => x + 4).then(x => x + 8);
p.then(x => console.log('Apartat D: ' + x));

/**
 * La Promesa es rebutja retornant un 0.
 * Com la promesa ha sigut rebutjada ens saltem els dos primers 'then', amb el que tenim que:
 * x = 0
 * Al catch, com la promesa va ser rebutjada, no l'ignorem i fem:
 * x = x + 4 = 0 + 4 = 4
 * Al 'then' després del catch no l'ignorem i fem:
 * x = x + 8 = 4 + 8 = 12
 * Amb el que tenim que:
 * x = 12
 * x = 12
 */


/**
 * Apartat E
 * @type {Promise<never>}
 */
p = Promise.reject(0).then(x => x + 1, null).catch(x => x + 2).catch(x => x + 4);
p.then(x => console.log('Apartat E: ' + x));

/**
 * La Promesa es rebutja retornant null.
 * Com la promesa ha sigut rebutjada ens saltem el primer 'then', amb el que tenim que:
 * x = null
 * Al primer catch, com la promesa va ser rebutjada, no l'ignorem i fem:
 * x = x + 2 = null + 2 = 2
 * Al segon catch, com ja s'ha executat un catch, l'ignorem:
 * Amb el que tenim que:
 * x = 2
 */
```

Exercici 24

```
let antipromise = function (promise) {
    return new Promise((resolve, reject) => {
        promise.then((data) => {
            reject(data + ' then rejected!');
        }).catch((data) => resolve(data + ' then accepted!'));
    });
};

antipromise(Promise.reject('Rejected')).then(console.log);
antipromise(Promise.resolve('Accepted')).catch(console.log);
```

Exercici 25

```
let promiseToCallback = function (f) {
    return function (x, callback) {
        f(x).then((response) => callback(null, response), (response) => callback(response, null));
    };
};

let isEven = x => new Promise((resolve, reject) => {
    x % 2 ? reject(x) : resolve(x);
});

let isEvenCallback = promiseToCallback(isEven);

isEven(2).then(() => console.log("OK"), () => console.log("KO"));
isEvenCallback(2, (err, res) => console.log(err, res));
isEven(3).then(() => console.log("OK"), () => console.log("KO"));
isEvenCallback(3, (err, res) => console.log(err, res));
```

Exercici 26

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

let readToPromise = function (file) {
    let basePath = './';

    return new Promise((resolve, reject) => {
        fs.readFile(basePath + file, function (err, data) {
            if (err) {
                reject(err);
            }

            resolve(data);
        });
    });
};

readToPromise("a1.txt").then(x => console.log("Contents: ", x))
    .catch(x => console.log("Error: ", x));

readToPromise("notfound.txt").then(x => console.log("Contents: ", x))
    .catch(x => console.log("Error: ", x));
```

Exercici 27

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

let callbackToPromise = function (f) {
    return function (file) {
        return new Promise((resolve, reject) => {
            f(file, function (err, data) {
                if (err) {
                    reject(err);
                } else {
                    resolve(data);
                }
            });
        });
    };
};

let readToPromise2 = callbackToPromise(fs.readFile);
readToPromise2("a1.txt").then(x => console.log("Contents: ", x))
    .catch(x => console.log("Error: ", x));
```

Exercici 28

```javascript
const fs = require('fs');
const {StringDecoder} = require('string_decoder');
const decoder = new StringDecoder('utf8');

var enhancedFutureToPromise = function () {
    return new Promise(
        (resolve, reject) => {
            enhancedFuture.registerCallback(ef => {
                resolve(ef.result);
            })
        }
    );
}

var asyncToEnhancedFuture = function (f) {
    return (fileName) => {
        let callback = null;

        let resFuture = {
            isDone: false,
            result: null,
            registerCallback: function (p) {
                if (resFuture.isDone){
                    p(resFuture);
                } else {
                    callback = p;
                }
            }
        };

        f(fileName, (err, data) => {
            resFuture.result = decoder.write(data);
            resFuture.isDone = true;
            if (callback !== null){
                callback(resFuture);
            }

            decoder.end();
        });

        return resFuture;
    }
}

readIntoEnhancedFuture = asyncToEnhancedFuture(fs.readFile);
let enhancedFuture = readIntoEnhancedFuture('a1.txt');
let promise = enhancedFutureToPromise(enhancedFuture);
promise.then(console.log);
```

Exercici 29

```javascript
let mergedPromise = function (p) {
    return new Promise((resolve) => {
        p.then((data) => {
            resolve(data);
        }).catch((data) => {
            resolve(data);
        });
    });
};

mergedPromise(Promise.resolve(0)).then(console.log);
mergedPromise(Promise.reject(1)).then(console.log);
```

Exercici 30

```
let functionPromiseComposer = (function (f1, f2) {
   return (function (x) {
      return new Promise((resolve, reject) => {
         f2(x).then((result) => {
            f1(result).then((result) => {
               resolve(result);
            }).catch((result) => {
               reject(result);
            });
         }).catch((result) => {
            reject(result);
         });
      });
   });
});

let f1 = x => new Promise((resolve, reject) => resolve(x+1));
functionPromiseComposer(f1, f1)(3).then(console.log);

let f3 = x => new Promise((resolve, reject) => reject('always fails'));
functionPromiseComposer(f1, f3)(3).catch(console.log);
```

Exercici 31

```
let parallelPromise = function (p1, p2) {
   return new Promise((resolve, reject) => {
      let rp1;
      let rp2;
      let halfWay = false;

      p1.then(x => {
         rp1 = x;
         if (halfWay) {
            resolve([rp1, rp2]);
         } else {
            halfWay = true;
         }
      });

      p2.then(x => {
         rp2 = x;
         if (halfWay) {
            resolve([rp1, rp2]);
         } else {
            halfWay = true;
         }
      });
   });
};

let p2 = parallelPromise(Promise.resolve(0), Promise.resolve(1));
p2.then(console.log);
```

Exercici 32

```
let promiseBarrier = function(n) {
    let list = [];
    let functions = [];
    let counter = 0;
    let params = [];

    for(let i=0; i<n; i++) {
        list[i] = function(x1) { // f1, f2, f3
            counter++;

            return new Promise((resolve, reject) => {
                //resolve(x1);
                functions[i] = resolve;
                params[i] = x1;

                // (2)
                if(counter === n) {
                    for (let j=0; j<n; j++){
                        let r = functions[j];
                        r(params[j]);
                    }
                }
            });
        };
    }

    return list;
};

var [f1, f2] = promiseBarrier(2);
Promise.resolve(0)
    .then(x => { console.log("c1 s1"); return x; })
    .then(x => { console.log("c1 s2"); return x; })
    .then(x => { console.log("c1 s3"); return x; })
    .then(x => { console.log("c1 s4"); return x; })
    .then(f1);
Promise.resolve(0)
    .then(f2)
    .then(x => { console.log("c2 s1"); return x; })
    .then(x => { console.log("c2 s2"); return x; });

var [f1, f2, f3] = promiseBarrier(3);
Promise.resolve(0)
    .then(x => { console.log("c1 s1"); return x; })
    .then(x => { console.log("c1 s2"); return x; })
    .then(x => { console.log("c1 s3"); return x; })
    .then(f1)
    .then(x => { console.log("c1 s4"); return x; });
Promise.resolve(0)
    .then(x => { console.log("c2 s1"); return x; })
    .then(f2)
    .then(x => { console.log("c2 s2"); return x; });
Promise.resolve(0)
    .then(f3)
    .then(x => { console.log("c3 s1"); return x; })
    .then(x => { console.log("c3 s2"); return x; })
    .then(x => { console.log("c3 s3"); return x; });

var [f1, f2] = promiseBarrier(2);
Promise.resolve(1).then(f1).then(console.log);
Promise.resolve(2).then(f2).then(console.log);
```