Marta Díaz Muñoz
NIU : 1425967

Llista de problemes de NodeJS

1. Crea una funció f1 que rebi un paràmetre a i que l'escrigui a la consola fent servir la funció log de l'objecte console.

```
function f1(a){console.log(a)}
```

2. Crea una funció f2 que rebi un paràmetre a i que retorni 2 * a si a >= 0 i -1 en cas contrari.

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

3. Crea una funció f3 que rebi una llista com a primer paràmetre i retorni una llista.

```
f2(56)
function f3(llista){
  llista3 = llista
  for (var i = 0; i<llista.length ; i++){
    llista3[i] = f2(llista[i]) + 23
  }
  return llista3
}
llista = [2,5,7,8,9,45,3,6]
```

4. Afegeix una nova funció printaki a l'objecte console que imprimeixi "aqui" per consola.

```
console.printaki = function () {
console.log("Aqui")
}
console.printaki()
```

5. Primer fes una funció f4 que sumi dos números (i.e., f4(a,b) 7! a + b), i fes una llista
llistaA = [1,2,3,4]
Fes servir llistaB = llistaA.map(...) per sumar 23 a cada element de llistaA, i fes servir f4 per fer la suma. Indicació: et caldrà fer una funció addicional, ja que no es pot fer servir f4 directament.

```
f4 = function (p1,p2) {
  p3 = p1+p2
  return p3
}
llistaA = [1,2,3,4]
llistaB = llistaA.map(function (x) {
  return f4(x,23)
})

console.log(llistaB)
```

Marta Díaz Muñoz
NIU : 1425967

Llista de problemes de NodeJS

6. Crea una funció f5 que agafi un objecte i dues funcions per paràmetres, anomenats respectivament a, b, i c:

```
f5 = function (a, b, c){
   c(b(a))
}
f5(1, f2, function(r) { console.log(r) })
```

7. Afegeix una nova funció printaki2 a l'objecte console que imprimeixi per consola "aqui 1", "aqui 2", "aquí 3", etc.

```
function createInc(startValue) {
  return function () {
     startValue += 1;
     console.log("Aqui" + startValue);
};
}
console.printaki2 = createInc(0);

console.printaki2()
```

8. Crea una funció f6 que tingui dos paràmetres: una llista de noms d'arxiu i una funció callback. f6 = function(llista, callback_final) { ... } La funció f6 ha de llegir els arxius anomenats a llista i ha de crear una nova llista resultat amb el contingut d'aquests arxius. I.e., cada element de resultat ha de ser el contingut d'un dels arxius.

```
fs = require('fs')


listaArchivos          =          ['C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi.txt','C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt']
f6 = function(listaArchivos, callback){
  listaResultados = []
  callback2 = function (error, data){
    console.log("data: ", data)
    listaResultados.push(data)
      if(listaResultados.length == listaArchivos.length){
        callback(listaResultados)
      }
  }
  listaArchivos.forEach(function(x){
    fs.readFile(x, 'utf8', function (err,data) {
            if (err) return callback2(err)
            callback2(null, data)
          })
  })
}

f6(listaArchivos, function (result){
  console.log(result)
})
```

Marta Díaz Muñoz
NIU : 1425967

Llista de problemes de NodeJS

9. Modifica la funció f6 de l'exercici anterior perquè l'ordre de la llista resultat coincideixi amb l'ordre original de llista. ´ Es a dir que a cada posició resultat[i] hi ha d'haver el contingut de l'arxiu anomenat a llista[i]. Anomena aquesta funció modificada com f7.

```
fs = require('fs')


listaArchivos          =          ['C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi.txt','C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt']
f6 = function(listaArchivos, callback){
listaArchivos.forEach(function(x){
  fs.readFile(x, 'utf8', function (err,data) {
          if (err) return callback(err)
          callback(null, data)
})
})}

f6(listaArchivos, function (error,data){
  if (error){
     console.log("error: ", error)}
  console.log("Data: ", data)
})
```

10. Explica perquè, a l'exercici anterior, hi podria haver problemes si en comptes de fer això:

```
llista.forEach(function (element, index) { /* ... */ } )
féssim això altre
 var index = 0
llista.forEach(function (element) { /* ... */ index += 1} )
```

Quan fem ús de "var" estem gestionant aquella variable com única, en canvi, fer ús del "let" ens permet protegir la variable. El "let" ens permet que existeixin variable amb mateix nom però que apuntin a posicions de memòria diferents, cada variable "let" viu en un scope diferent.

11. Implementa la funció asyncMap. Aquesta funció té la següent convenció d'us:

```
asyncMap = function(list,f,callback2){
  listaResultados = []
  callback1 = function (error, result){
    if (error != null){
      callback2(error,listaResultados)
    }
    listaResultados.push(result)
    if (listaResultados.length == list.length ){
      callback2(null,listaResultados)
    }
  }
  list.forEach(function(x){
    f(x,'utf8', function(error,data){callback1(error,data)})
  })
```

Llista de problemes de NodeJS

```
}

asyncMap(['C:\\Users\\madim\\Desktop\\UNI\\Segon  curs\\EC\\holi2.txt'],
fs.readFile, function (a, b) { console.log(b) })
```

12. Fes un objecte o1 amb tres propietats: un comptador count, una funció inc que incrementi el comptador, i una variable notify que contindrà null o bé una funció d'un paràmetre. Feu que el comptador "notifiqui" la funció guardada a la propietat notify cada cop que el comptador s'incrementi.

```
var o1 ={
  count:0,
  notify : null,
  inc : function(){
    this.count+=1
    if (this.notify != null){
      this.notify(this.count)
    }
  }
}

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

13. Fes el mateix que a l'exercici anterior però fes servir el module pattern per amagar el valor del comptador i la funció especificada a notify. Fes un setter per la funció triada per l'usuari. Anomena l'objecte com o2.

```
var o2 = (function() {
  var notify = null
  var count = 1;
    return { //return decir que es publico
      inc : function() {
        count++;
        if (notify != null){
          notify(count)
        }
       },
      count: function() { return count; },
      setNotify: function (f){ notify=f }
  };
})();

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

Marta Díaz Muñoz
NIU : 1425967

Llista de problemes de NodeJS

14. Converteix l'exemple anterior en una classe i assigna'l a un objecte o3. En que es diferencien els dos exemples?

```
o3= function(){
  this.a = 1;
  this.notify = null
  this.inc = function () {
     this.a++;
     if (this.notify != null){
       this.notify(this.a)
     }
   },
  this.count = function() { return this.a; }

}

temp = new o3();
temp.notify = function (a) { console.log(a) };
temp.inc()
```

15. Fes una nova classe, DecreasingCounter, que estengui l'anterior per herència i que faci que el mètode inc en realitat decrementi el comptador.

```
class DecreasingCounter extends o3 {
  inc = function () {
     this.a--;
     if (this.notify != null){
       this.notify(this.a)
     }
   }
}



temp = new DecreasingCounter();
temp.notify = function (a) { console.log(a) };
temp.inc()
```

16. Definim un objecte de "tipus future" com un objecte de dos camps tal i com es mostra a continuació:
future = { isDone: false, result: null }

```
fs = require('fs')

future = { isDone: false, result: null }

readIntoFuture = function (filename){
  futureRes = future
  fs.readFile(filename,'utf8',function (err,data){
    futureRes.isDone = true
    futureRes.result = data

  })
```

Llista de problemes de NodeJS

```
    return futureRes
}

futureResultado =
readIntoFuture('C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt');
setTimeout(function() { console.log(future) }, 1000)
```

17. Suposem que tenim una funció f amb la mateixa convenció de crida que fs.readFile (això vol dir que f podria ser fs.readFile).
Generalitza l'exercici anterior de la següent manera. Fes una funció asyncToFuture(f) que "converteixi" la funció f en una nova funció equivalent però que retorni un future tal que l'exercici anterior.

```
//future = {isdone: false, result:null}
asyncToFuture = function(f){
  return function(arg){
    f(arg,(err,data)=>{
      future = {
        isdone: true,
        result: data
      }
    })
    return future
  }
}
readIntoFuture2 = asyncToFuture(fs.readFile);
future2a = readIntoFuture2('C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt');
setTimeout(function() { console.log(future2a) }, 1000)

statIntoFuture = asyncToFuture(fs.stat);
future2b = statIntoFuture('C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt');
setTimeout(function() { console.log(future2b) }, 1000)
```

18. Fes la funció asyncToEnhancedFuture que faci el mateix que la funció anterior, però que retorni un objecte de "tipus enhanced future".

```
fs = require('fs')
let ef = 'Marta'
asyncToEnhancedFuture = function(f){
  return function(arg){
    f(arg,(err,data)=>{
      future = {
        isdone: true,
        result: data,
        registerCallback: function (f2){
        return function(arg2){
```

Llista de problemes de NodeJS

```
            f2(arg2)
          }
        }
      }
    })
    return future
  }
}
readIntoEnhancedFuture = asyncToEnhancedFuture(fs.readFile);
enhancedFuture =
readIntoEnhancedFuture('C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt');
enhancedFuture.registerCallback( function(ef) {console.log(ef) }
)
```

19. Tenim que f1(callback) és una funció que rep un paràmetre de callback, i f2(error, result) és la funció de callback que faríem servir a f1, aleshores volem fer la funció when que ha de funcionar de la següent manera.

```
fs = require('fs')
f1 = function(callback) {
fs.readFile('C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt', 'utf-8', callback) }
f2 = function(error, result) { console.log(result) }
when = function (f1) {
    return  a = {do : function (ff2) {
      f1(ff2)
    }
  }
}


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

20. Modifica la solució de l'exercici anterior perquè funcioni així:
when(f1).and(f2).do(f3)

```
fs = require('fs')
f1 = function(callback) {
fs.readFile('C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt', 'utf-8', callback) }
f2 = function(callback) {
fs.readFile('C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt', 'utf-8', callback) }
f3 = function(err1, err2, res1, res2) { console.log(res1, res2) }

when = function (f) {
  return  a = {
    and : function (fand) {
       return b= {
```

Llista de problemes de NodeJS

```
          do:function (ffdo) {
            //f(callback)
            //fand(callback)
            let err_f1
            let res_f1
            let err_f2
            let res_f

            let f1_ran = false
            let f2_ran = false

            f((err, res)=> {
              err_f1 = err
              res_f1 = res
              f1_ran = true
                if (f1_ran && f2_ran){
                    ffdo((err_f1, res_f1, err_f2, res_f2))
                }

            })
            fand((err, res)=> {
              err_f2 = err
              res_f2 = res
              f2_ran = true
              if (f1_ran && f2_ran){
                  ffdo((err_f1, res_f1, err_f2, res_f2))
              }
            })
          }
        }
      }
    }
  }
}
when(f1).and(f2).do(f3)
```

21. Fes la funció composer que rebi dues funcions d'un sol paràmetre.

```
fs = require('fs')
var composer = function (f, ff){
  return function(z){
    var y = (f(ff(y)))
    console.log(y)
    return y
  }
}

f1 = function(a) { return a + 1 }
```

Marta Díaz Muñoz
NIU : 1425967

Llista de problemes de NodeJS

```
f3 = composer(f1, f1)
f3(3)
f4 = function(a) { return a * 3 }
f5 = composer(f3, f4)
f5(3)
```

22. Converteix l'exercici anterior en as´ıncron tal com s'explica a continuació. Fes la funció asyncComposer que rebi dues funcions: f1 i f2.

```
var asyncComposer = function(f, ff){
  var fff = function(a, callback){
    ff (a, (err_f2,resf2))=> {
      if (errf2){
        callback(errf2, resf2)
      }else{
        f (resf2, ((err_f1,resf1)))
        callback(errf1, resf1)
      }
    }
  }
}
```

23. Fes un p.then(x => console.log(x)) per cadascuna de les promisses p que es mostren a continuació.
Digues què s'imprimeix per pantalla i el perquè.
(a) p = Promise.resolve(0).then(x => x + 1).then(x => x + 2).then(x => x + 4);

Surt 7 perquè no hi ha hagut cap execució errònia.
Resolve --> promesa que no torna res, callback
Reject --> torna error

(b) p = Promise.reject(0).then(x => x + 1).catch(x => x + 2).then(x => x + 4);

X = 6 perquè com es comença amb reject va directament al catch i com que catch va correctament va a l'últim then.

(c) p = Promise.resolve(0).then(x => x + 1).then(x => x + 2).catch(x => x + 4).then(x => x + 8);

x = 3 ja que es resol des d'un bon principi i executa els dos then següents. No executa el catch perquè no fa un reject i el then final no s'executa perquè està concatenat amb el catch

(d) p = Promise.reject(0).then(x => x + 1).then(x => x + 2).catch(x => x + 4)
.then(x => x + 8);

x = 12 perquè la promise comença en un reject, després executa el catch i per últim, com que ha executat bé el catch, executa correctament el then del final.

Marta Díaz Muñoz
NIU : 1425967

Llista de problemes de NodeJS

(e) p = Promise.reject(0).then(x => x + 1, null).catch(x => x + 2).catch(x => x + 4);

x = 2 perquè la promesa comença amb un reject i executa el catch. Com el primer catch s'executa bé , no cal que executi el segon catch.

24. Fes una funció antipromise que rebi una promise per paràmetre i retorni una promise diferent. La promise retornada s'ha de resoldre (resolve) quan la promise original es rebutjada (reject) i viceversa.

```
function antipromise(p) {
    return new Promise((resolve, reject) => {
        p.then(reject).catch(resolve);
    })
}

antipromise(Promise.reject(0)).then(console.log)
antipromise(Promise.resolve(0)).cath(console.log)
```

25. Fes la funció promiseToCallback que converteixi la funció f en la funció g, on f retorna el resultat en forma de promesa, mentre que g retorna el resultat amb un callback.

```
//25)

let promiseToCallback = function(f){
  return function(algo, callback){
    f(algo).then(resultat => {
      callback(null,resultat);
    }, error =>{
      callback(error, null);
    })
  }
}
var isEven = x => new Promise(
(resolve, reject) => x % 2 ? reject(x) : resolve(x)
);
var isEvenCallback = promiseToCallback(isEven);
```

26. Fes la funció readToPromise(file) que llegeixi l'arxiu file amb fs.readFile i retorni el resultat en forma de promise.

```
fs = require('fs')

let readToPromise = function(file){
  return new Promise ((resolve, reject)=> {
    fs.readFile(file, function (error,result){
      if (error){
```

Llista de problemes de NodeJS

```
        reject(error)
      }else{
        resolve(result)
      }
    })
  })
}
readToPromise("C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt").then(x => console.log("Contents: ",
x)).catch(x => console.log("Error: ", x));
```

27. Generalitza l'exercici anterior de la mateixa manera com asyncToFuture generalitza la funció readIntoFuture.

```
fs = require('fs')

let callbackToPromise = function(f){
  return function(x){
    return new Promise ((resolve, reject)=> {
      f(x, function (error,result){
        if (error){
          reject(error)
        }else{
          resolve(result)
        }
      })
    })
  }
}

readToPromise2 = callbackToPromise(fs.readFile);
readToPromise2("C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt").then(x => console.log("Contents: ",
x)).catch(x => console.log("Error: ", x));
```

28. Fes la funció enhancedFutureToPromise que donat un enhancedFuture el converteixi en una promise.

```
fs = require('fs')
future = { isDone: false, result: null, registerCallback:
function(x){} }
asyncToEnhancedFuture = function(f){
  return function(arg){
    f(arg,(err,data)=>{
      future = {
        isdone: true,
        result: data,
        registerCallback: function (f2){
```

Llista de problemes de NodeJS

```
        return function(arg2){
            f2(arg2)
         }
        }
      }
    })
    return future
  }
}
var readIntoEnhancedFuture = asyncToEnhancedFuture(fs.readFile);

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

var enhancedFuture =
readIntoEnhancedFuture('C:\\Users\\madim\\Desktop\\UNI\\Segon
curs\\EC\\holi2.txt');
var promise = enhancedFutureToPromise(enhancedFuture);
promise.then(console.log)
```

29. Fes la funció mergedPromise que, donada una promesa, retorni una altra promesa. Aquesta segona promesa sempre s'ha de resoldre i mai refusar (resolve and never reject) al valor que es resolgui o es refusi la promesa original.

```
var mergedPromise = function(promesa){
  return new Promise((resolve, reject)  =>{
    promesa.then(resolve).catch(resolve);
  })
}

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

30. Donades dues funcions, f1 i f2, on totes dues funcions prenen un sol paràmetre i retornen una promesa, fes la funció functionPromiseComposer que prengui aquestes dues funcions per paràmetre i que retorni la funció f3.

```
var functionPromiseComposer = function (p1,p2){
  return function(x){
    return p2(x).then(p1)
  }}
```

Marta Díaz Muñoz
NIU : 1425967

Llista de problemes de NodeJS

31. Fes la funció parallelPromise que rep dues promises per paràmetre i retorna una tercera promise per resultat.Aquesta tercera promise serà el resultat d'executar les dues promises per paràmetre en paral·lel. ´ Es a dir, l'exercicidel when(f1).and(f2).do(f3) però amb promises.

```
var parallelPromise = function(p1,p2) {
  return Promise.all([p1,p2])
};
```

32. Fes la funció promiseBarrier. Aquesta funció rep per paràmetre un enter estrictament més gran que zero, i retorna una llista de funcions.

```
var promiseBarrier = function(numero) {
  var list = []
  var fun = []
  var par = []
  var suma = 0
  list.forEach((ele, i) => {
    list[i] = function(x1) {
      suma++
      return new Promise((resolve, reject) => {
      fun[i] = resolve
      par[i] = x1
      if(suma == numero) {
        fun.forEach((ele, j) => {
          fun[i](par[i]);

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