Pràctica 0 – Introducció a JavaScript
Part 2: Exemples

Tecnologia dels Sistemes d'Informació en la Xarxa

Índex

1. Un exemple erroni

- Si no estem acostumats a programar en JavaScript, els nostres primers programes pot ser que no funcionen bé.
- Possibles problemes:
  - Algunes funcions podran proporcionar els seus resultats asincrònicament
    - En utilitzar ***callbacks***
      - és a dir, arguments per a una funció A que també són funcions i que seran invocades quan A finalitze la seua tasca.
      - Si A utilitza alguna crida al sistema que deixe a l'invocador en estat "suspès", llavors A necessitarà un llarg interval per a acabar.
      - Això trenca l'execució normal (és a dir, seqüencial) del nostre procés...
        - Perquè els *callbacks* no s'executen immediatament, sinó una mica més tard.
  - En execucions asincròniques, podrem trobar valors inesperats en algunes variables.

1. Un exemple erroni

- Suposem un programa que...:
  - mostra quin fitxer és el més gran (i la seua grandària en bytes)...
  - ...d'una llista de noms de fitxers rebuda com a arguments des de la línia d'ordres.
- Una possible primera versió és:

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: for (var i=0; i<args.length; i++)
 6:   fs.readFile(args[i],'utf8',
 7:     function(err,data) {
 8:       if (!err) {
 9:         console.log('Processing'
10:                 +' %s...',args[i])
11:         if (data.length >
12:           maxLength) {
13:           maxLength = data.length
14:           maxName=args[i]
15:         }
16:       } // if (!err)…
17:     }) // readFile()…
18: console.log('The  longest file is '
19: +'%s and its length is %d bytes.',
20:   maxName, maxLength)
```

- …però aquesta versió no funciona com esperem.

1. Un exemple erroni

- Per a comprovar si aquest programa és correcte, hauríem de:
  1. Executar-lo amb arguments vàlids.
  2. Comprovar la seua eixida.
  3. Si l'eixida no fóra correcta, seguir una traça de la seua execució.
- Així, en cas d'error, podríem detectar on es comporta malament el programa.
- Suposem que tenim els següents fitxers:
  - A: 2300 bytes
  - B: 180 bytes
  - C: 4500 bytes
  - D: 470 bytes
- I que executem el programa (el nom del qual és "files.js") utilitzant aquesta ordre:

  node files A C D B

1. Un exemple erroni

- I amb això obtenim els següents resultats:

```
$ node files A C D B
The longest file is NONE and its length is 0 bytes.
Processing undefined...
Processing undefined...
Processing undefined...
Processing undefined...
$
```

- Revisem la seua traça per a entendre aquesta eixida...

1. Un exemple erroni

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: for (var i=0; i<args.length; i++)
 6:   fs.readFile(args[i],'utf8',
 7:     function(err,data) {
 8:       if (!err) {
 9:         console.log('Processing'
10:               +' %s...',args[i])
11:         if (data.length >
12:           maxLength) {
13:           maxLength = data.length
14:           maxName=args[i]
15:         }
16:       } // if (!err)…
17:     }) // readFile()…
18: console.log('The  longest file is '
19: +'%s and its length is %d bytes.',
20:   maxName, maxLength)
```

- Les quatre primeres línies fan el següent:
  1. Importar el mòdul "fs" en la constant **fs**.
  2. Assignar els arguments de la línia d'ordres al vector **args**:
     - ["A","C","D","B"]
  3. Assignar "NONE" a maxName .
  4. Assignar 0 a maxLength .

1. Un exemple erroni

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: for (var i=0; i<args.length; i++)
 6:   fs.readFile(args[i],'utf8',
 7:     function(err,data) {
 8:       if (!err) {
 9:         console.log('Processing'
10:                 +' %s...',args[i])
11:         if (data.length >
12:           maxLength) {
13:           maxLength = data.length
14:           maxName=args[i]
15:         }
16:       } // if (!err)…
17:     }) // readFile()…
18: console.log('The  longest file is '
19: +'%s and its length is %d bytes.',
20:   maxName, maxLength)
```

- La línia 5 és un bucle "for" amb quatre iteracions:
  - La variable "i" rep els valors de 0 a 3 en aquestes iteracions.
  - En cadascuna es crida la funció "readFile()":
    - Utilitzant cada argument de la línia d'ordres com a primer argument en cada crida.
    - Observe's que el tercer paràmetre és *un callback*.

1. Un exemple erroni

```
1:  const fs=require('fs')
2:  var args=process.argv.slice(2)
3:  var maxName='NONE'
4:  var maxLength=0
5:  for (var i=0; i<args.length; i++)
6:    fs.readFile(args[i],'utf8',
7:      function(err,data) {
8:        if (!err) {
9:          console.log('Processing'
10:               +' %s...',args[i])
```

Aquest *callback* serà invocat quan el fitxer corresponent s'haja llegit. Així, el procés reacciona a la finalització d'aquesta operació. En aquest moment, es mostra el missatge "Processing <nomFitxer>…" en la pantalla.

- La línia 5 és un bucle "for" amb
  - La variable "i" rep els valors de 0 a
  - En cadascuna es crida la funció "readFi
    - Utilitzant cada argument de la línia d'ord om a primer argument en cada crida.
    - Observe's que el tercer paràmetre és *un callback*.

1. Un exemple erroni

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: for (var i=0; i<args.length; i++)
 6:   fs.readFile(args[i],'utf8',
 7:     function(err,data) {
 8:       if (!err) {
 9:         console.log('Processing'
10:               +' %s...',args[i])
11:         if (data.length >
12:           maxLength) {
13:           maxLength = data.length
14:           maxName=args[i]
15:         }
16:       } // if (!err)…
17:     }) // readFile()…
18: console.log('The  longest file is '
19: +'%s and its length is %d bytes.',
20:   maxName, maxLength)
```

- Però "readFile()" té un comportament asincrònic:
  - Els processos JavaScript es comporten com a processos amb un únic fil d'execució.
  - readFile() s'utilitza per a llegir un fitxer.
    - Amb això, l'invocador roman suspès quan el sistema operatiu (SO) rep aquesta crida.
  - Per a evitar la suspensió del procés, es crea un nou fil en cada crida a readFile().
    - Aquest nou fil intern invoca al SO, roman suspès i prepararà el context d'execució del *callback* una vegada siga reactivat.
    - Aquest fil és gestionat de forma transparent pel mòdul "fs". Queda ocult al programador.
  - Mentrestant, el fil principal continua.

1. Un exemple erroni

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: for (var i=0; i<args.length; i++)
 6:   fs.readFile(args[i],'utf8',
 7:     function(err,data) {
 8:       if (!err) {
 9:         console.log('Processing'
10:               +' %s...',args[i])
11:         if (data.length >
12:           maxLength) {
13:           maxLength = data.length
14:           maxName=args[i]
15:         }
16:       } // if (!err)…
17:     }) // readFile()…
18: console.log('The  longest file is '
19: +'%s and its length is %d bytes.',
20:   maxName, maxLength)
```

- Per tant, el fil d'execució principal…
  - Executa totes les iteracions sense suspendre's.
    - La seua execució no és interrompuda pels altres fils del seu procés.
  - I continua una vegada finalitze el bucle "for".

1. Un exemple erroni

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: for (var i=0; i<args.length; i++)
 6:   fs.readFile(args[i],'utf8',
 7:     function(err,data) {
 8:       if (!err) {
 9:         console.log('Processing'
10:                 +' %s...',args[i])
11:         if (data.length >
12:           maxLength) {
13:           maxLength = data.length
14:           maxName=args[i]
15:         }
16:       } // if (!err)…
17:     }) // readFile()…
18: console.log('The  longest file is '
19: +'%s and its length is %d bytes.',
20:    maxName, maxLength)
```

- Així, arriba a la instrucció que segueix al bucle (línies 18 a 20).
  - Per això, imprimeix:

    `The longest file is NONE and its length is 0 bytes.`

    - Perquè ningú ha modificat encara els valors de maxName o **maxLength**.
  - Una vegada fet això, aquest fil ha completat totes les seues instruccions. Per tant, cerca altres "torns d'execució":
    - Hi ha o hi haurà alguns: aquells corresponents als contextos d'execució dels *callbacks* que passen a estar preparats.

1. Un exemple erroni

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: for (var i=0; i<args.length; i++)
 6:   fs.readFile(args[i],'utf8',
 7:     function(err,data) {
 8:       if (!err) {
 9:         console.log('Processing'
10:                 +' %s...',args[i])
11:         if (data.length >
12:           maxLength) {
13:           maxLength = data.length
14:           maxName=args[i]
15:         }
16:       } // if (!err)…
17:     }) // readFile()…
18: console.log('The  longest file is '
19: +'%s and its length is %d bytes.',
20:   maxName, maxLength)
```

- Aquests contextos d'execució dels *callbacks* passen a preparats quan finalitzen les seues crides al sistema corresponents.
  - El temps necessari per a fer això dependrà de la grandària de cada fitxer.
  - Així, l'ordre en què es mostrarà cada fitxer podrà diferir de l'ordre en què aquests noms apareixien en la línia d'ordres.

1. Un exemple erroni

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: for (var i=0; i<args.length; i++)
 6:   fs.readFile(args[i],'utf8',
 7:     function(err,data) {
 8:       if (!err) {
 9:         console.log('Processing'
10:                 +' %s...',args[i])
11:         if (data.length >
12:           maxLength) {
13:           maxLength = data.length
14:           maxName=args[i]
15:         }
16:       } // if (!err)…
17:     }) // readFile()…
18: console.log('The  longest file is '
19: +'%s and its length is %d bytes.',
20:   maxName, maxLength)
```

- De fet, quan aquests *callbacks* s'executen, el fil principal ja ha finalitzat totes les iteracions del bucle "for" (Fulla 12)
  - Així, quin és el valor de la variable "i" en aquest moment?
    - És args.length; és a dir, 4 en el nostre exemple.
    - Però process.argv[4] és "**undefined**" perquè únicament manté noms de fitxer en les seues components 0 a 3!

1. Un exemple erroni

I això explica l'eixida obtinguda:

```
The longest file is NONE and its length is 0 bytes.
Processing undefined...
Processing undefined...
Processing undefined...
Processing undefined...
```

- De fet, quan a[illegible]*cks* s'executen, el fil principal ja ha finalitzat totes les [illegible]s del bucle "for" (Fulla 12)
  - Així, quin és el valor de [illegible]able "i" en aquest moment?
    - És args.length; és a dir, 4 en el nostre exemple.
    - Però process.argv[4] és "**undefined**" perquè únicament manté noms de fitxer en les seues components 0 a 3!

1. Un exemple erroni

- Hi ha dos problemes principals en aquest programa:
  1. Els missatges de traça que mostren el nom de cada fitxer no ofereixen valors correctes:
     - Estan basats en el valor de la variable "i", però aquest valor no pot donar-se com a argument del *callback*.
     - Per això, el nom de fitxer és incorrecte.
  2. El teòric missatge final que reportaria el nom i grandària del major fitxer es mostra abans del que voldríem, quan encara no hi ha informació vàlida.
     - Aquest missatge no hauria d'imprimir-se en un codi que seguisca al bucle "**for**".
       - Perquè aquestes instruccions s'executen abans que s'invoquen els *callbacks*.
     - El missatge hauria de mostrar-se en alguna instrucció situada DINS DEL *CALLBACK*!
       - Quan tots els noms de fitxer hagen sigut processats.
       - Necessitarem un comptador per a assegurar això.

Índex

1. Un exemple erroni
2. El mòdul "debug"
3. Una primera versió correcta
4. Segona versió correcta

2. El mòdul "debug"

- En lloc d'intentar corregir un programa erroni sense ajuda, és convenient afegir missatges de traça en ell.
  - No obstant això, després de detectar i corregir els errors, solem eliminar aquests missatges de traça.
  - El mòdul 'debug' és una bona ajuda:
    - Permet mostrar els missatges de traça quan se sol·licite.
    - I ocultar-los per omissió, sense incórrer en un sobrecost excessiu.
      - D'aquesta manera, ja no és necessari eliminar-los manualment.
    - La seua documentació està disponible en: https://www.npmjs.com/package/debug
  - En el nostre exemple, utilitzarem aquest mòdul per a esbrinar les causes dels resultats incorrectes explicats en la secció anterior.

2. El mòdul "debug"

- Per a usar aquest mòdul:
  - Cal instal·lar-lo, amb l'ordre:

    npm install debug
  - Ha d'importar-se en el programa a depurar...
    - Amb una o múltiples línies d'aquest estil:

      const deb = require('debug')('label')
      - On:
        - L'identificador de la constant ('deb' en aquest exemple) s'usarà com una funció que reemplace a console.log() per a mostrar els missatges de traça.
        - El nom de l'etiqueta utilitzada en els segons parèntesis ('label' en aquest exemple) és el valor a utilitzar per a habilitar aquests missatges des de la línia d'ordres.
          - En llançar el procés, hem d'assignar aquesta etiqueta a la variable d'entorn DEBUG.
    - Per a definir múltiples parts en el programa a depurar, podrem utilitzar múltiples línies "require", amb diferents etiquetes.
      - Posteriorment, seleccionarem quines parts integrar en la traça abans d'iniciar el procés.

2. El mòdul "debug"

- Suposem que s'han afegit missatges de traça al programa descrit en la Secció 1.
- El programa resultant podria ser com aquest:

```
 1: const var_i=require('debug')('var_i')
 2: const names=require('debug')('names')
 3: const fs=require('fs')
 4: var args=process.argv.slice(2)
 5: var maxName='NONE'
 6: var maxLength=0
 7: for (var i=0; i<args.length; i++) {
 8:     var_i('in loop: %d',i)
 9:     fs.readFile(args[i],'utf8',
10:         function(err,data) {
11:             if (!err) {
12:                 names('Processing %s...', args[i])
13:                 var_i('in callback : %d', i)
14:                 if (data.length>maxLength) {
15:                     maxLength=data.length
16:                     maxName=args[i]
17:                 }
18:             }
19:         })
20: }
21: console.log('The longest file is %s and its'
22:     + ' length is %d bytes.', maxName,
23:     maxLength);
```

2. El mòdul "debug"

- En aquest exemple, s'han afegit dues etiquetes:
  - var_i: Mostra missatges amb el valor de la variable "i" en el cos del bucle i en el cos del *callback*.
  - names: Reemplaça els missatges originals de traça.

```
 1: const var_i=require('debug')('var_i')
 2: const names=require('debug')('names')
 3: const fs=require('fs')
 4: var args=process.argv.slice(2)
 5: var maxName='NONE'
 6: var maxLength=0
 7: for (var i=0; i<args.length; i++) {
 8:     var_i('in loop: %d',i)
 9:     fs.readFile(args[i],'utf8',
10:         function(err,data) {
11:             if (!err) {
12:                 names('Processing %s...', args[i])
13:                 var_i('in callback : %d', i)
14:                 if (data.length>maxLength) {
15:                     maxLength=data.length
16:                     maxName=args[i]
17:                 }
18:             }
19:         })
20: }
21: console.log('The longest file is %s and its'
22:     + ' length is %d bytes.', maxName,
23:     maxLength);
```

2. El mòdul "debug"

- Ara, quan executem aquest programa no obtindrem cap missatge de traça per omissió:
  - Per això, si executem aquesta ordre:

    node files A C D B
  - Obtindrem únicament aquesta eixida:

```
The longest file is NONE and its length is 0 bytes.
```

2. El mòdul "debug"

- Però podrem triar quins són els missatges de traça a mostrar.
- Per a fer això, assignarem les seues etiquetes a la variable d'entorn DEBUG.
- Comencem amb els missatges originals, mostrats en la primera versió del programa. Necessitarem aquesta declaració en el *shell*:

  export DEBUG=names # En Windows, usaríem: **set DEBUG=names**

- Després d'això, llançaríem el procés:
  - Així, en donar l'ordre:

    node files A C D B

  - Obtindríem aquesta eixida:

```
The longest file is NONE and its length is 0 bytes.
  names Processing undefined... +0ms
  names Processing undefined... +2ms
  names Processing undefined... +0ms
  names Processing undefined... +0ms
```

2. El mòdul "debug"

- Però necessitarem més missatges per a entendre per què falla.
  - Aquells etiquetats amb 'var_i'
  - Per a fer això, podrem usar o bé...

    export DEBUG=names,var_i
  - ...o:

    export DEBUG=*
- I ja podrem iniciar aquest procés:
  - Amb l'ordre **node files A C D B** obtindrem ara:

```
  var_i in loop: 0 +0ms
  var_i in loop: 1 +4ms
  var_i in loop: 2 +0ms
  var_i in loop: 3 +1ms
The longest file is NONE and its length is 0 bytes.
  names Processing undefined... +0ms
  var_i in callback: 4 +4ms
  names Processing undefined... +1ms
  var_i in callback: 4 +1ms
  names Processing undefined... +0ms
  var_i in callback: 4 +1ms
  names Processing undefined... +1ms
  var_i in callback: 4 +0ms
```

2. El mòdul "debug"

Les primeres quatre línies de l'eixida mostren que el bucle "**for**" s'ha executat en primer lloc. En ell la variable "**i**" s'ha anat incrementant com s'esperava.

```
  var_i in loop: 0 +0ms
  var_i in loop: 1 +4ms
  var_i in loop: 2 +0ms
  var_i in loop: 3 +1ms
The longest file is NONE and its length is 0 bytes.
  names Processing undefined... +0ms
  var_i in callback: 4 +4ms
  names Processing undefined... +1ms
  var_i in callback: 4 +1ms
  names Processing undefined... +0ms
  var_i in callback: 4 +1ms
  names Processing undefined... +1ms
  var_i in callback: 4 +0ms
```

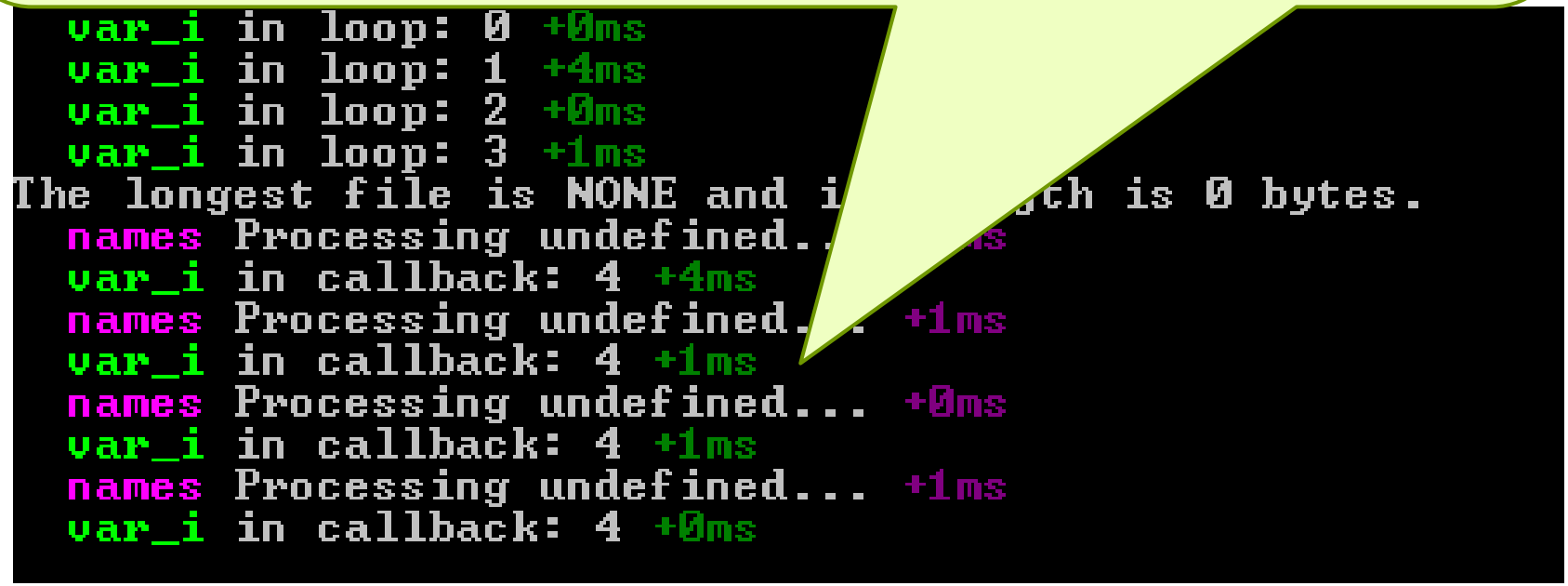
2. El mòdul "debug"

Quan el bucle finalitza, el procés mostra el missatge gestionat en les línies 21 a 23 del programa.
No obstant això, en aquest moment cap dels *callbacks* s'ha executat encara. Això explica els resultats incorrectes del programa.

```
  var_i in loop: 0 +0ms
  var_i in loop: 1 +4ms
  var_i in loop: 2 +0ms
  var_i in loop: 3 +1ms
The longest file is NONE and its length is 0 bytes.
  names Processing undefined... +0ms
  var_i in callback: 4 +4ms
  names Processing undefined... +1ms
  var_i in callback: 4 +1ms
  names Processing undefined... +0ms
  var_i in callback: 4 +1ms
  names Processing undefined... +1ms
  var_i in callback: 4 +0ms
```

2. El mòdul "debug"

Finalment, s'executen els *callbacks*, però quan això ocorre la variable "**i**" té un valor inesperat, 4, perquè el bucle "**for**" ja ha finalitzat.
A més, en aquest moment el missatge que havia de mostrar els resultats ja ha sigut imprès, oferint valors erronis.



2. El mòdul "debug"

- Aquesta secció ha mostrat com:
  - Els missatges de traça poden ser gestionats fàcilment amb el mòdul "debug".
    - Així és més fàcil identificar els errors en els nostres programes.
- Una vegada els errors han sigut localitzats i corregits, els missatges de traça poden seguir integrats en el codi...
  - Perquè tard o d'hora s'ampliarà o modificarà el programa i altres errors podrien introduir-se en aquestes extensions.
- Per a ocultar els missatges de traça hem d'assegurar que la variable d'entorn DEBUG no tinga valor.
  - DEBUG=

Índex

1. Un exemple erroni
2. El mòdul "debug"
3. Una primera versió correcta
4. Segona versió correcta

3. Una primera versió correcta

- Recordem els problemes identificats en la Secció 1:
  - El segon problema és fàcil de resoldre.
    - Ja es va donar una guia en la Fulla 16.
  - El primer problema es deu al fet que el *callback* no pot accedir a "i" en la iteració del bucle, sinó després.
    - Els *callbacks* de les biblioteques tenen una signatura ja definida.
      - No té sentit afegir paràmetres o arguments.
      - Per tant, necessitem algun mecanisme per a passar el nom de fitxer apropiat al codi del *callback*.
        - La solució es basa en l'àmbit de declaració de les variables.
          - Una funció pot accedir a tota variable o paràmetre declarat en un àmbit més extern.
          - Així, convindrà escriure una funció que mantinga el nom de fitxer en algun dels seus paràmetres i retorne com a resultat el *callback* a utilitzar.
            - Amb això, el codi del *callback* podrà conèixer el nom de fitxer.
            - Això és una CLAUSURA.

3. Una primera versió correcta

- Aquest programa resol els problemes que hem vist. Així:
  - Escriu quin és el fitxer més gran (i la seua grandària)...
  - …d'una llista de noms de fitxer rebuts com a arguments.
- El seu codi és:

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:         maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:         maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

- Ara, la funció generator() resol el segon problema:
  - Rep el nom de fitxer en el seu paràmetre.
    - Per tant, tot el seu codi pot utilitzar-lo.
  - I retorna una funció la signatura de la qual coincideix amb la dels *callbacks* de readFile(). Aquesta funció processarà el resultat d'aquesta lectura.
- A més, en la seua línia 16 comprova si s'han processat tots els fitxers.
  - En cas afirmatiu, mostra el missatge amb els resultats i el procés acaba ací.

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:         maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

- En la línia 24, quan s'especifica el tercer argument de readFile():
  - No passem un punter a la funció "generator" sinó que LA CRIDEM passant el nom de fitxer apropiat com a argument.
  - Això genera la funció a utilitzar com a *callback*.

3. Una primera versió correcta

- Utilitzem aquesta nova versió del programa, amb els mateixos arguments explicats en la Fulla 5:

```
$ node files A C D B
Processing D...
Processing B...
Processing A...
Processing C...
The longest file is C and its length is 4500 bytes.
$
```

- Seguim una traça per a entendre per què ara l'eixida proporcionada és correcta…

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:         maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

- Les seues primeres 5 línies fan el següent:
  1. Importar el mòdul "fs" en la constant **fs**.
  2. Assignar els arguments de la línia d'ordres al vector **args**.
  3. Assignar "NONE" a maxName .
  4. Assignar 0 a maxLength .
  5. Inicialitzar el comptador de noms processats a zero.

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:           maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

- Les línies 6 a 21 defineixen la funció generator().
  - Però aquesta funció no s'invoca encara.
  - Quan s'invoque, retornarà com a resultat una funció anònima.
    - Aquesta funció té dos paràmetres (*err* i *data*) que casen amb la signatura exigida als *callbacks* de fs.readFile().
- De moment, l'execució prosseguirà en la línia 22.

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:           maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

- Les línies 22 a 24 defineixen un bucle amb tantes iteracions com noms de fitxer. En cada iteració:
  - Es crida la funció readFile(). Així, el procés inicia la lectura del fitxer corresponent.
  - El seu tercer paràmetre és una crida a la funció generator().
    - Amb ella, el nom del fitxer processat en cada iteració és rebut en el paràmetre 'name', accessible al codi del *callback*.
    - Per tant, això resol el primer problema citat en la Fulla 16.

3. Una primera versió correcta

- Quan totes les iteracions han acabat, el fil inicial ja no té més instruccions a executar.
  - Aparentment, ha acabat tot el programa.
  - Però el procés en execució encara no ha acabat...
    - …perquè hi ha quatre crides a readFile() que encara no han finalitzat!
- Aquestes crides a readFile() acabaran en algun moment.
  - Cada vegada que acabe alguna, el seu *callback* s'executarà.
  - S'ha cridat a readFile() amb aquesta seqüència de noms: "A", "C", "D" i "B".
    - Però "C" és el fitxer major i "B" el més xicotet.
      - Per això és impredictible el seu ordre de finalització!
- Continuem amb la traça…

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:         maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

Segons l'eixida mostrada en la Fulla 34, el primer fitxer que finalitza el seu readFile() és D. La seua grandària és 470 bytes. El *callback* mostra això en la pantalla:

Processing D…

…i després modifica el valor de maxLength (deixant-lo a 470) i maxName ("D"). La variable "counter" s'incrementa.

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:         maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

Posteriorment finalitza B. La seua grandària és 180 bytes. El seu *callback* mostra això en la pantalla:

Processing B…

…però ara maxLength i maxName no es modifiquen. La variable "counter" s'incrementa. Tindrà valor 2. No es mostra res més.

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:         maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<args.length; i++)
23:   fs.readFile(args[i], 'utf8',
24:     generator(args[i]))
```

Després finalitza A. La seua grandària és 2300 bytes. El seu *callback* mostra:

Processing A…

…i es modifiquen maxLength (2300) i maxName ("A"). La variable "counter" s'incrementa (valor 3). No es mostra res més.

3. Una primera versió correcta

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: function generator(name) {
 7:   return function(err,data) {
 8:     if (!err) {
 9:       console.log('Processing %s...',
10:         name)
11:       if (data.length>maxLength) {
12:         maxLength=data.length
13:         maxName=name
14:       }
15:     }
16:     if (++counter==args.length)
17:       console.log('The longest file is %s '
18:       +'and its length is %d bytes.',
19:       maxName, maxLength)
20:   }
21: }
22: for (var i=0; i<a                    -)
23:   fs.readFile(
24:     gen
```

Finalment finalitza C (4500 bytes). El seu *callback* escriu:

Processing C…

…i es modifiquen maxLength i maxName. La variable "counter" s'incrementa (valor 4). Per tant, es mostra el missatge amb els resultats, indicant que el major fitxer és C.

Com ja no hi ha altres fitxers, el procés finalitza ací.

Índex

4. Segona versió correcta

- Els programes vists fins ara han usat "**var**" per a declarar variables.
  - El primer problema discutit en la Secció 1 estava causat parcialment per aquestes declaracions!
  - Una solució més compacta a aquest problema consisteix a utilitzar "**let**" en lloc de "**var**".
    - "**let**" defineix la variable en l'àmbit del bloc actual.
      - Un bloc és un grup d'instruccions tancades entre un parell de claus {}
      - Així, aquestes variables poden utilitzar-se en aquest bloc i altres continguts en ell.
    - Quan "**let**" s'use en l'àmbit global, aquestes variables podran usar-se a partir d'aquest punt.
      - No defineixen propietats de l'objecte "**global**".
  - A més, la instrucció "**for**" defineix un bloc implícit que inclou totes les instruccions del bucle.
    - Així, totes les instruccions del bucle "recorden" quin ha sigut el valor actual de la variable iteradora en aquesta sentència "**for**".

4. Segona versió correcta

- Per tant, aquest programa és també correcte i facilita la mateixa eixida que aquell mostrat en la Secció 3:

```
 1: const fs=require('fs')
 2: var args=process.argv.slice(2)
 3: var maxName='NONE'
 4: var maxLength=0
 5: var counter=0
 6: for (let i=0; i<args.length; i++)
 7:   fs.readFile(args[i],'utf8',
 8:     function(err,data) {
 9:       if (!err) {
10:         console.log('Processing %s...',
11:             args[i])
12:         if (data.length>maxLength) {
13:           maxLength=data.length
14:           maxName=args[i]
15:         }
16:       }
17:       if (++counter==args.length)
18:         console.log('The longest file is %s'
19:           +' and its length is %d bytes.',
20:           maxName, maxLength)
21:   })
22:
23:
24:
```

4. Segona versió correcta

Com "i" es defineix amb "**let**", el seu àmbit és sol el conjunt d'instruccions d'aquest bucle.
Cada iteració utilitza una "nova" definició de "i", amb un valor diferent.

```
 1: const fs=
 2: var args=                    lice(2)
 3: var max                      ONE'
 4: var ma                       n=0
 5: var co                       ter=0
 6: for (let i=0; i<args.length; i++)
 7:   fs.readFile(args[i],'utf8',
 8:     function(err,data) {
 9:       if (!err) {
10:         console.log('Processing %s...',
11:             args[i])
12:         if (data.length>maxLength) {
13:           maxLength=data.length
14:           maxName=args[i]
15:         }
16:       }
17:       if (++counter==args.length)
18:         console.log('The longest file is %s'
19:           +' and its length is %d bytes.',
20:           maxName, maxLength)
21:   })
22:
23:
24:
```

4. Segona versió correcta

Per això, el *callback* utilitzat en cada iteració "recorda" quin valor de la "i" ha sigut utilitzat en ella.
Així, escriu el nom de fitxer correcte en les línies 10 i 11 i l'assigna en la línia 14. Observe's que el bucle "**for**" ja haurà acabat quan s'executen aquests *callbacks*.

```
 1: const fs=
 2: var args
 3: var ma
 4: var ma
 5: var co
 6: for (
 7:   fs.r
 8:     function(err,data) {
 9:       if (!err) {
10:         console.log('Processing %s...',
11:             args[i])
12:         if (data.length>maxLength) {
13:           maxLength=data.length
14:           maxName=args[i]
15:         }
16:       }
17:       if (++counter==args.length)
18:         console.log('The longest file is %s'
19:           +' and its length is %d bytes.',
20:           maxName, maxLength)
21:   })
22:
23:
24:
```