NIST: Unit 2. Activities

This bulletin presents a series of multiple-choice questions related to the content of Topic 2. Each element begins with a specific question, to which four possible answers are offered. Only one of these answers is correct.

This type of question is typically used in exams for the subject. On these exams, leaving a question unanswered would result in zero points, while a wrong answer would result in a loss of one-third of the grade for a correct answer.

1. Specify what will be displayed on the screen when the following Node.js program is run:

```
function counter(initialValue) {
  return function () {
          return ++initialValue
  }
}

let increase = counter(0)

console.log(increase())
console.log(increase())
```

a	function() {return ++initialValue} function() {return ++initialValue}
b	1 2
c	0 0
d	Some error messages.

2. Specify the order in which the three numbers will be displayed on the screen when the following Node.js program is run:

```
setTimeout(()=>{console.log("2")},10)
setTimeout(()=>{console.log("1")},0)
console.log("0")
```

a	2 1 0
b	0 1 2
c	0 2 1
d	1 0 2

3. **Knowing that process.stdout.write() displays the string received as an argument on the standard output (screen), without subsequently skipping to the next line, indicate what will be displayed on the screen when running the following Node.js program:**

```
function counter(initialValue) {
  return function () {
          return ++initialValue
  }
}
let increase = counter(0)
process.stdout.write(increase()+"")
let another = counter(increase())
process.stdout.write(another()+"")
process.stdout.write(increase()+"")
```

initialValue = 1
initialValue = 2 -> another.initialValue = 2
another.initialValue = 3
initialValue = 3

a	134
b	122
c	133
d	Some error messages.

4. **How many *callbacks* are shown in the listing of this program?**

```
const fs = require('fs')
fs . readdir ( '.' , function( err , files ) {
   let count = files . length; let results = {}
   files . forEach (function( filename ) {
      fs . readFile ( filename , function( err , data ) {
         console . log ( filename , 'has been read' )
         results [ filename ] = data
         if (-- count <= 0 )
            console . log ( '\nTOTAL:' , files . length , 'files have been read' )
      })
   })
})
```

a	None
b	One
c	Two
d	Three

NIST

5. How many times would a *callback be invoked* when running this program if there were three entries (i.e., three file names) in that directory?

```
const fs = require('fs')
fs . readdir ( '.' , function( err , files ) {
   let count = files . length; let results = {}
   files . forEach (function( filename ) {
      fs . readFile ( filename , function( err , data ) {
         console . log ( filename , 'has been read' )
         results [ filename ] = data
         if (-- count <= 0 )
            console . log ( '\nTOTAL:' , files . length , 'files have been read' )
      })
   })
})
```

<- we are calling this callback 3 times, each for each file

a	None
b	One
c	Three
d	Seven

6. When will this program display the line indicating how many files ("TOTAL: … files have been read") it has read or is going to read?

```
const fs = require('fs')
fs.readdir('.', function(err, files) {
   let count = files.length;  let results = {}
   files.forEach(function(filename) {
     fs.readFile(filename, function(err, data) {
       console.log(filename, 'has been read')
       results[filename] = data
       if (--count <= 0)
         console.log('\nTOTAL:', files.length, 'files have been read')
     })
   })
})
```

a	At no time, since it does not show it.
b	Certainly, before displaying any file name.
c	Certainly, after displaying the name of all the files read.
d	At some position within the list of lines displayed on the screen, since the *callbacks* used for this are asynchronous.

7. When will this program display the line indicating how many files ("TOTAL: … files have been read") it has read or is going to read?

```
const fs = require('fs')
fs.readdir('.', function(err, files) {
   let count = files.length;  let results = {}
   files.forEach(function(filename) {
     if (--count <= 0)
       console.log('\nTOTAL:', files.length, 'files have been read')
     fs.readFile(filename, function(err, data) {
       console.log(filename, 'has been read')
       results[filename] = data
      })
   })
})
```

a	At no time, since it does not show it.
b	Certainly, before displaying any file name.
c	Certainly, after displaying the name of all the files read.
d	At some position within the list of lines displayed on the screen, since the *callbacks* used for this are asynchronous.

8. Select the true statement about these two Node.js programs, the purpose of which is to display the total size of all files whose name is given as an argument:

```
// File: readNFiles.js
const fs=require('fs')

let filenames = process.argv.slice(2)
let totalLength = 0

if (filenames.length < 2) {
   console.error("At least two file names are needed!")
   process.exit(1)
}

filenames.forEach(function (name) {
   try {
      totalLength += (fs.readFileSync(name) + "").length
   } catch (e) {}
})

console.log("Total length: ", totalLength)
```

```
// File: readNFiles2.js
const fs=require('fs')
let filenames = process.argv.slice(2)
let totalLength = 0
let readFiles = 0
if (filenames.length < 2) {
   console.error("At least two file names are needed!")
   process.exit(1)
}
filenames.forEach(function (name) {
   fs.readFile(name, function (err,data) {
      readFiles++
      if (!err)
         totalLength += (data+"").length
      if (readFiles == filenames.length)
         console.log("Total length: ", totalLength)
   })
})
```

a	Neither program meets the stated objective.
b	Only readNFiles.js fulfills the stated objective.
c	Only readNFiles2.js fulfills the stated objective.
d	Both programs meet the stated objective.

NIST

9. The following programs attempt to display the total size of all files whose names are given as arguments. Do any of them obtain the size of each of those files in the exact order in which they are given as arguments?

```
// File: readNFiles.js
const fs=require('fs')

let filenames = process.argv.slice(2)
let totalLength = 0

if (filenames.length < 2) {
   console.error("At least two file names are needed!")
   process.exit(1)
}

filenames.forEach(function (name) {
   try {
      totalLength += (fs.readFileSync(name) + "").length
   } catch (e) {}
})

console.log("Total length: ", totalLength)
```

```
// File: readNFiles2.js
const fs=require('fs')
let filenames = process.argv.slice(2)
let totalLength = 0
let readFiles = 0
if (filenames.length < 2) {
   console.error("At least two file names are needed!")
   process.exit(1)
}
filenames.forEach(function (name) {
   fs.readFile(name, function (err,data) {
      readFiles++
      if (!err)
         totalLength += (data+"").length
      if (readFiles == filenames.length)
         console.log("Total length: ", totalLength)
   })
})
```

a	Neither program meets the stated objective.
b	Only readNFiles.js fulfills the stated objective.
c	Only readNFiles2.js fulfills the stated objective.
d	Both programs meet the stated objective.

10. This program is intended to display the name and size of each of the files whose names it receives as arguments from the command line. If errors aren't needed to be handled, what code is missing from the declaration of its function f?

```
const fs=require('fs')
let filenames = process.argv.slice(2)
function f(n) {
   // f code
}
filenames.forEach(function (name) {
   fs.readFile(name, f(name))
})
```

a	console.log(n, ": ", n.length)
b	return (e,d) => {console.log(n,": ", (d+"").length)}
c	return (e,d) => {console.log(n,": ", (e+"").length)}
d	There is no possible solution if we can only add code in the f function.

NIST

11. The following programs attempt to display the total size of all files whose name is given as an argument. Do any of them allow concurrent reads of those files?

```
// File: readNFiles.js
const fs=require('fs')

let filenames = process.argv.slice(2)
let totalLength = 0

if (filenames.length < 2) {
   console.error("At least two file names are needed!")
   process.exit(1)
}

filenames.forEach(function (name) {
   try {
      totalLength += (fs.readFileSync(name) + "").length
   } catch (e) {}
})

console.log("Total length: ", totalLength)
```

```
// File: readNFiles2.js
const fs=require('fs')
let filenames = process.argv.slice(2)
let totalLength = 0
let readFiles = 0
if (filenames.length < 2) {
   console.error("At least two file names are needed!")
   process.exit(1)
}
filenames.forEach(function (name) {
   fs.readFile(name, function (err,data) {
      readFiles++
      if (!err)
         totalLength += (data+"").length
      if (readFiles == filenames.length)
         console.log("Total length: ", totalLength)
   })
})
```

a	Neither program meets the stated objective.
b	Only readNFiles.js fulfills the stated objective.
c	Only readNFiles2.js fulfills the stated objective.
d	Both programs meet the stated objective.

12. This Node.js program displays the contents of the two files whose names it receives from the command line. If either of these two files doesn't exist, the process aborts. Which changes would be appropriate so that, instead of aborting, this type of error generates a message and the process terminates normally after displaying the error?

```
const fs=require('fs').promises
let filenames = process.argv.slice(2)
async function showContents() {
   console.log( await fs.readFile(filenames[0])+"")
   console.log( await fs.readFile(filenames[1])+"")
}
showContents()
```

a	This program does not do what is stated, so no changes are applicable.
b	Replace the first line with: **const fs=require('fs')**
c	Remove both **await** and **async** wherever they appear.
d	Replace the last line with: **showContents().catch(console.log)**

13. The following programs attempt to display the total size of all files whose names are given as arguments. If the name of a nonexistent file is supplied as an argument on the command line, the program should not abort and should not modify the total size value to be displayed. Do any of these programs meet these conditions?

```
// File: readNFiles.js
const fs=require('fs')

let filenames = process.argv.slice(2)
let totalLength = 0

if (filenames.length < 2) {
   console.error("At least two file names are needed!")
   process.exit(1)
}

filenames.forEach(function (name) {
   try {
      totalLength += (fs.readFileSync(name) + "").length
   } catch (e) {}
})

console.log("Total length: ", totalLength)
```

```
// File: readNFiles2.js
const fs=require('fs')
let filenames = process.argv.slice(2)
let totalLength = 0
let readFiles = 0
if (filenames.length < 2) {
   console.error("At least two file names are needed!")
   process.exit(1)
}
filenames.forEach(function (name) {
   fs.readFile(name, function (err,data) {
      readFiles++
      if (!err)
         totalLength += (data+"").length
      if (readFiles == filenames.length)
         console.log("Total length: ", totalLength)
   })
})
```

to	Neither program meets the stated conditions.
b	Only readNFiles.js fulfills the stated conditions.
c	Only readNFiles2.js fulfills the stated conditions.
d	Both programs meet the stated conditions.

14. A program needs to be written to determine how many lines of a text file (whose name will be received as the first argument from the command line) contain a given word (to be received as the second argument from the command line). To develop this program, an initial, incomplete sketch has been written, which is presented below. LinesWithWord returns a function that will return an array with all the lines of contents that include the string w, and lineCount counts the lines in the received array. Indicate what code, based on the use of promises, should be used to complete it (without having to worry about handling any errors that may occur during execution):

```
const fsp=require('fs').promises
let filename = process.argv[2]
let word = process.argv[3]

function linesWithWord(w) {
   return function(contents) {
      result = []
      contents+=""
      contents.split("\n").forEach(function(line) {
         if (line.indexOf(w)>-1)
            result.push(line)
      })
   return result
   }
}

function lineCount(lineArray) {
   return lineArray.length
}

// The code to be used would be written here…
```

a	fsp.readFile(filename).linesWithWord(word).lineCount()
b	fsp.readFile(filename).then(linesWithWord(word)).then(lineCount).then(console.log)
c	fsp.readFile(filename).then(linesWithWord(word)).then(lineCount)
d	This program cannot be completed using promises.

15. A program needs to be written to determine how many lines of a text file (whose name will be received as the first argument from the command line) contain a given word (to be received as the second argument from the command line). To develop this program, an initial, incomplete draft has been written, which is presented below. LinesWithWord returns a function that will return an array with all the lines of contents that include the string w, and lineCount counts the lines in the received vector. Indicate what code, based on the use of synchronous functions, should be used to complete it (without having to worry about handling any errors that may occur during execution):

```
const fs=require('fs')
let filename = process.argv[2]
let word = process.argv[3]

function linesWithWord(w) {
   return function(contents) {
      result = []
      contents+=""
      contents.split("\n").forEach(function(line) {
         if (line.indexOf(w)>-1)
            result.push(line)
      })
   return result
   }
}

function lineCount(lineArray) {
   return lineArray.length
}

// The code to be used would be written here…
```

a	console.log(fs.readFileSync(filename).linesWithWord(word)(this).lineCount(this))
b	console.log(linesWithWord(word)(lineCount(fs.readFileSync(filename))))
c	console.log(lineCount(linesWithWord(word)(fs.readFileSync(filename))))
d	None of the other solutions are valid.

16. Select the true statement about the behavior of the following program, used as an example of a basic proxy at the end of practice 1:

```
const net = require('net')
const LOCAL_PORT  = 8000
const LOCAL_IP  = '127.0.0.1'
const REMOTE_PORT = 80
const REMOTE_IP = '158.42.4.23' // www.upv.es

const server = net.createServer(function (socket) {
      const serviceSocket = new net.Socket()
      serviceSocket.connect(parseInt(REMOTE_PORT),
         REMOTE_IP, function () {
          socket.on('data', function (msg) {
               serviceSocket.write(msg)
          })
          serviceSocket.on('data', function (data) {
             socket.write(data)
          })
      })
}).listen(LOCAL_PORT, LOCAL_IP)
```

a	When this program is executed, the resulting process immediately connects to port 80 of the computer whose IP address is 158.42.4.23
b	Only one client process can connect to this proxy: the one that manages to “listen” on port 8000 of address 127.0.0.1
c	Multiple client processes can connect to this proxy, and they will all share the same “serviceSocket” instance used to connect to the remote server.
d	All other statements are false.

17. Select the true statement about the behavior of the program presented in the previous question, used as an example of a basic proxy at the end of practice 1:

a	If no connection is dropped, there will be as many connections between the proxy and the remote server as there are clients connected to the proxy.
b	The data that the proxy receives from the remote server is simultaneously forwarded to all clients connected to the proxy.
c	Data received by the proxy from one client is simultaneously forwarded to all other clients connected to the proxy.
d	All other statements are false.

18. What Node.js program is needed to write the message "Hello" to the screen numTimes so that one second elapses between each pair of consecutive messages in that sequence?

a	setInterval(()=>{console.log("Hello")}, 1000, numTimes)
b	setInterval(console.log("Hello"), 1000, numTimes)
c	for(let i=0; i<numTimes; i++) setTimeout(function () {console.log("Hello")}, i*1000)
d	None of the other alternatives are correct.

19. This is an incomplete program that should generate theEvent once per second and receive as an argument in its *listener* for that event the number of times it has been generated so far. Select the code needed to complete the program so that it meets these conditions.

```
const ev = require('events')
const ee = new ev.EventEmitter()
const name = 'theEvent'
ee.on(name, function (arg) {
   console.log( "Event", name,"has happened", arg, "times")
})
function evGenerator() {
// Código a añadir.
}
setInterval(evGenerator(),1000)
```

a	let c=0; c++; ee.emit(name); return c
b	let c=0; return () => {c++; ee.emit(name,c)}
c	return ()=> {let c=0; c++; ee.emit(name,c)}
d	None of the other alternatives are correct.

20. We've programmed a function called Example that takes an integer argument and, among other things, displays the value of that argument on the screen. We want to invoke this function once per second indefinitely, always passing the value 501 as its argument. How could we solve this?

a	setTimeout(Example(501), 1000)
b	setInterval(Example(501), 1000)
c	setInterval(() => {Example(501)}, 1000)
d	setInterval(Example, 501, 1000)