Problemes 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**.

   Exemple d'us:
   ```
   f1(3)
   ```
   Resultat:
   ```
   3
   ```

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

   Exemple d'us:
   ```
   f2(3)
   ```
   Resultat:
   ```
   6
   ```

   Exemple d'us:
   ```
   f2(-2)
   ```
   Resultat:
   ```
   -1
   ```

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

   ```
   llista2 = f3(llista)
   ```

   Cada element **y** de **llista2** ha de ser el resultat d'aplicar

   ```
   f2(x) + 23
   ```

   a cada element **x** de **llista**, on **f2** és la funció de l'exercici anterior.

   Exemple d'us:
   ```
   f3([1,2,3])
   ```
   Resultat:
   ```
   [25,27,29]
   ```

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

   Exemple d'us:
   ```
   console.printaki()
   ```
   Resultat:
   ```
   aqui
   ```

5. Primer fes una funció **f4** que sumi dos números (i.e., **f4(a,b)** $\mapsto$ **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.

   Exemple d'us:
   ```
   llistaB
   ```
   Resultat:
   ```
   [24,25,26,27]
   ```

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) {
           // a és un objecte, b és una funció, i c és una funció.
   }
   ```

   La funció **f5** ha d'aplicar la funció **b** a l'objecte **a**. El resultat se li ha de passar a **c**. La funció **c** ha de ser una funció *callback* amb un paràmetre (i.e., aquesta funció s'ha de cridar quan la feina que faci **f5** s'hagi acabat, i el resultat de la feina se li ha de passar com a paràmetre).

   Exemple d'us:
   ```
   f5(1, f2, function(r) { console.log(r) })
   ```
   Resultat:
   ```
   2
   ```

7. Afegeix una nova funció **printaki2** a l'objecte **console** que imprimeixi per consola "aqui 1", "aqui 2", "aqui 3", etc. És a dir "aqui {número anterior + 1}". No facis servir cap variable global (ni cap objecte global) per guardar el comptador; fes servir una clausura.

   Exemple d'us:
   ```
   console.printaki2()
   ```
   Resultat:
   ```
   aqui 1
   ```

   Exemple d'us:
   ```
   console.printaki2()
   ```
   Resultat:
   ```
   aqui 2
   ```

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**, on a cada posició **resultat[i]** hi ha d'haver el contingut de l'arxiu anomenat a **llista[i]**.

   Fes servir **.forEach(...)** i **fs.readFile(filename, callback)**.

   Quan la funció hagi acabat de llegir *tots* els arxius (quan es cridi l'última funció callback passada a **fs.readFile(...)**), s'ha de cridar la funció callback que **f4** rep com a paràmetre (i.e., **callback_final**) amb la llista resultant.

   Exemple d'us:
   ```
   f6(['a1.txt','a2.txt'], function (result) { console.log(result) })
   ```
   Resultat:
   ```
   ['contingut a1.txt', 'contingut a2.txt']
   ```

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

   ```
   asyncMap(list,
            function f(a,
                       function callback1(err, result)
                      ),
            function callback2(err, resultList)
           )
   ```

   Fixa't en que **f(...)** té la mateixa forma que **fs.readFile(...)**).

   La funció **asyncMap** aplica **f** a cada element de **list** i crida **callback2** quan acaba d'aplicar **f** a tots els elements de la llista. La funció **callback2** s'ha de cridar, o bé amb el primer **err != null** que se li hagi passat a **callback1**, o bé amb **resultList** contenint el resultat de la feina feta per **asyncMap** (en l'ordre correcte).

   Indicació: fixa't en els paral·lelismes entre aquest exercici i l'anterior. Intenta entendre que vol dir fer un map asíncron.

   Exemple d'us:
   ```
   asyncMap(['a1.txt'], fs.readFile, function (a, b) { console.log(b) })
   ```
   Resultat:
   ```
   ['contingut 1']
   ```

10. 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.

    Exemple d'us:
    ```
    o1.notify = null; o1.inc()
    ```
    Resultat:

    Exemple d'us:
    ```
    o1.count = 1; o1.notify = function(a) { console.log(a) }; o1.inc()
    ```
    Resultat:
    ```
    2
    ```

11. 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`**.

    El següent codi és un exemple de module pattern que pots reaprofitar:

```
var testModule = (function() {
        var count = 1;
        return {
                inc : function() { count++; },
                count: function() { return count; }
        };
})();
```

    Exemple d'us:

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

    Resultat:

```
2
```

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

    El següent codi és un exemple d'una classe que pots reaprofitar:

```
function Counter() {
        this.a = 1;
        this.inc = function () { this.a++; },
        this.count = function() { return this.a; }
}

new Counter();
```

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