LAB 1

Let us implement two JavaScript programs, **proxy.js** and **controlador.js**, that constitute a valid solution to a similar problem to that of the reverse proxy posed in Section 3 of the first practice bulletin.

Controller	proxy
• It receives 4 input arguments from the command line: - **IP** address of the **proxy**. - Proxy **port** to be reprogrammed (8001 .. 8003) - **IP** address of the **new remote** server. - **Port** of attention of the **new remote** server. • It sends a message to port 8000 of the proxy with the format illustrated by means of the following example: `{inPort:8001, remote:{ip:'158.42.4.23', port:80}}`	• It does not have any input arguments • It listens to requests from **customers** through ports **8001, 8002, 8003** • It listens to the **controller** through port **8000**. These messages serve for reprogramming to which servers client requests are forwarded.

In the code of the programs, the fragments of code labelled as **COMPLETE-X** remain incomplete.

controlador.js

```
var net = require('net');

var PROG_PORT = 8000;

if(process.argv.length != 6) {
  console.log("Use:  node controller
      proxyIP port remoteIP remotePort")
  process.exit()
}

var PROXY_IP = process.argv[2]
var PORT = process.argv[3]
var REMOTE_IP = process.argv[4]
var REMOTE_PORT = process.argv[5]

COMPLETE4

client.on('end', function() {
  console.log('reprogrammed REMOTE')
})
```

proxy.js

```
var net =require('net');
rserver ={}
rserver[8001]={Port: 80, IP: 'www.upv.es'},
rserver[8002]={Port: 80, IP: 'www.dsic.upv.es'},
rserver[8003]={Port: 80, IP: 'www.google.es'}

for (i=8001; i<=8003; i++) {
  var server = net.createServer (manejador(i)).listen(i)
  console.log ('TCP server listening on port ' + i)
}
var server2 =net.createServer (controlador).listen(8000)

function manejador(i) {
     return function (socket)  {
        socket.on ('data', function (msg) {
          var serviceSocket = new net.Socket();
          serviceSocket.connect (COMPLETE1, COMPLETE2,
            function () {
              serviceSocket.write (msg);
          });
          serviceSocket.on ('data', function (data) {
              socket.write (data)
          })
       })
    }
}
function controlador (socket) {
    socket.on ('data', function (msg) {
        COMPLETE3
    })
}
```

Based on this context, please answer the following questions:

1. Choose the option that defines correct **COMPLETE1** and **COMPLETE2** fragments:

	COMPLETE1	**COMPLETE2**
a)	`8000`	`REMOTE_IP`
b)	`REMOTE_PORT`	`REMOTE_IP`
c)	`rserver[i].Port`	`rserver[i].address`
d)	`rserver[i].Port`	`rserver[i].IP`

2. Choose the option that defines a correct **COMPLETE3** fragment:

a)
```
var req = JSON.parse(msg)
rserver[req.Port].Port = parseInt(req.remote.port)
rserver[req.Port].IP   = req.remote.ip
```

b)
```
var req = JSON.parse(msg)
rserver[req.inPort].Port = parseInt(req.remote.port)
rserver[req.inPort].IP   = req.remote.ip
```

c)
```
var req = JSON.parse(msg)
rserver[req.inPort].Port = parseInt(req.inPort)
rserver[req.inPort].IP   = req.remote.ip
```

d)
```
var req = JSON.parse(msg)
rserver[req.remote.port].Port = parseInt(req.inPort)
rserver[req.remote.port].IP   = req.remote.ip
```

3. Choose the option that defines a correct **COMPLETE4** frament:

a)
```
var client = net.connect(PROG_PORT, PROXY_IP,
   function() {
      var req = {inPort: PORT, remote:{ip: REMOTE_IP, port:REMOTE_PORT}}
      client.write(JSON.stringify(req))
      client.end
})
```

b)
```
var client = net.connect(REMOTE_PORT, REMOTE_IP,
   function() {
      var req = {inPort: PROG_PORT, remote:{ip: REMOTE_IP, port:PORT}}
      client.write(JSON.stringify(req))
      client.end
})
```

c)
```
var client = net.connect(PROG_PORT, PROXY_IP,
   function() {
      var req = {inPort: PORT, remote:{ip: REMOTE_IP, port:REMOTE_PORT}}
      client.write(JSON.parse(req))
      client.end
})
```

d)
```
var client = net.connect(PORT, PROXY_IP,
  function() {
    var req = {inPort: REMOTE_PORT, remote:{ip: REMOTE_IP, port:PROG_PORT}}
    client.write(JSON.stringify(req))
    client.end
})
```

4. We want to use **proxy** as a proxy between a client program called **buscaDatos** and a server program called **calculaDatos:**

- **buscaDatos** receives as its first input argument the IP of the server and as a second argument the port of that server.
- **calculaDatos** does not have any input arguments.

The deployment of those components is the following:

- **buscaDatos** runs in machine M1, with IP address 158.42.156.6.
- **calculaDatos** runs in machine M2, with IP address 158.42.156.7 and it receives requests in port 5688.
- **proxy** runs in machine M3, with IP address 158.42.156.4.
- **controlador** runs in machine M4.

Choose which of the following invocations of those programs (in the specified order) allows that **buscaDatos** makes properly the request to **calculaDatos** through **proxy**

a)	**M2:**	`$node calculaDatos`
	M3:	`$node PROXY`
	M4:	`$node controlador 158.42.156.7 8003 158.42.156.4 5688`
	M1:	`$node buscaDatos 158.42.156.4 8003`
b)	**M3:**	`$node PROXY`
	M4:	`$node controlador 158.42.156.4 8003 158.42.156.7 5688`
	M2:	`$node calculaDatos`
	M1:	`$node buscaDatos 158.42.156.4 8003`
c)	**M2:**	`$node calculaDatos`
	M3:	`$node PROXY`
	M4:	`$node controlador 158.42.156.4 8003 158.42.156.7 5688`
	M1:	`$node buscaDatos 158.42.156.4 8000`
d)	**M2:**	`$node calculaDatos`
	M3:	`$node PROXY`
	M4:	`$node controlador 158.42.156.4 8003 158.42.156.7 5688`
	M1:	`$node buscaDatos 158.42.156.7 5688`