This exam consists of 20 multiple choice questions. In every case only one of the answers is correct. You should answer in a separate sheet. If correctly answered, each question contributes 0,5 points to your grade. If the answer is wrong, the contribution is negative: -0,1 points.

In the answer sheet, fill carefully your chosen slot. To this end, use a dark pen or pencil. You may use “Tipp-Ex” or any other correction fluid to change your answer. In that case, please do not try to rebuild the empty slot. Leave it without any surrounding box.

THEORY

1. Distributed systems...

[✓ A]	...usually consist of multiple agents that are executed concurrently. Those agents maintain some independent state. In order to have a distributed system we need multiple processes (i.e., agents) that collaborate and are running in multiple computers providing a single system image. Despite this collaboration, each process may maintain some local state that is inknown by all other processes; i.e., some independent state.
B	...do not need any inter-computer communication mechanism. Those agents should be run in at least two different computers in order to be a distributed system. Because of this, those systems need a communication mechanism (usually a computer network) among computers.
C	...always use a client-server interaction mechanism. We have seen in Unit 1 that there are other inter-process interaction mechanisms; e.g., a peer-to-peer interaction approach. Therefore, not all the interactions need to be between clients and servers following a request-response pattern.
D	...always prevent race conditions from appearing. No. Race conditions may occur as in any other concurrent system. Distributed systems cannot guarantee, per se, that race conditions never happen.
E	All the above.
F	None of the above.

2. Cloud computing...

A	...is one of the current service provision paradigms in distributed computing.

B	...aims at providing software services in a scalable and efficient way.
E	All the above. Cloud computing is regularly oriented towards providing the appropriate mechanisms for developing, deploying and providing software services in an efficient and scalable way. In the regurlar case, those services are provided following a “pay per use” model and virtualisation mechanisms are employed in order to use as much as possible the available hardware resources.
C	...follows a “pay per use” model.
D	...uses virtualised infrastructures in the common case.
F	None of the above.

3. In a distributed system, its agents interact…

A	…in no way. If they interact, then the system is concurrent, but not distributed. Its agents should interact. Otherwise, the resulting system would be a large set of independent and isolated processes instead of a system able to combine the functionality of multiple basic components in order to solve in an efficient and fault-tolerant way some complex problems or tasks.
B	…exchanging messages or sharing memory. These are the two regular communication mechanisms in concurrent systems. Distributed systems also use both approaches.
C	…in any way that does not imply any sharing of memory. Memory sharing is forbidden in distributed systems. No. When several of the agents being used in a distributed system are located in the same computer, they may share memory. There is no rule that constrains the usage of shared memory in a distributed application when that memory may be shared among a subset of its agents.
D	…sharing all agents the same computer. No. It is a nonsense to compell that all the agents in a distributed system were deployed in a single computer. The failure of that computer would lead to the complete failure of that distributed application or system.
E	All the above.
F	None of the above.

4. The guard / action programming model is used in...

A	Multi-threaded programmes, where all critical sections are equivalent to guards and the threads are equivalent to actions. No. A guard is something similar to a precondition and a critical section is always a block of code. Therefore, they are not equivalent. Moreover a thread is a dynamic concept (a thread of execution) and an action could provide the sequence of instructions to be executed by a thread. Again, both concepts are not equivalent.
B	Asynchronous programming, where its events are actions and its *callback* functions (i.e., event listeners) are guards. See the explanation in part D. Events are guards and listeners are actions.
C	Multi-threaded programmes, where its threads are guards and critical sections are actions. See the explanation in part A. These four concepts are not equivalent (nor related).
✓ D	Asynchronous programming, where its events are guards and its event listeners are actions. In an algorithm, an action is a block of statements that makes sense on its own, since it is able to define some useful function. In the guard / action model, those actions are associated to their preconditions that are known as "guards" in that model. Once a precondition or guard is true, the action gets enabled and it may be executed. Asynchronous programming matches this guard / action model since its event listeners define blocks of statements equivalent to actions (they are run in an atomical way since they cannot be interrupted by other enabled actions) and an event can be considered equivalent to a guard that is accomplished when the event is raised.
E	All the above.
F	None of the above.

5. Some expected features in distributed system models are…

A	They are centred on the main properties of the system behaviour.
B	They provide a good basis to reason about the correctness of the algorithms and protocols based on them.
C	A high level of abstraction.
D	They provide a basis for discussing about the impossibility of solving problems in some kinds of distributed systems; e.g., consensus in asynchronous systems.
[✓ E]	All the above. A model must be centred in the essential properties of the element or system it represents (A) and this means that it needs a high level of abstraction (C) in order to ignore many concrete details that could be considered irrelevant. The goal of models is to provide a comfortable image for designing algorithms able to solve problems in the real system and to discuss their correctness before starting their implementation (B). Those discussions may be focused on the impossibility of solving some problems in those systems (D). Those discarding or negative results (i.e., impossibilities) may be based on the conditions that define the system model being assumed.
F	None of the above.

6. The elements of a system model can be...

A	Computer architecture, operating system, middleware, and programming language. All these elements are irrelevant at algorithm design time. System models should provide an abstract image that is useful for defining algorithms that solve some of the existing problems in a real system. Therefore, this list doesn't include any of the elements to be found in a system model.
[✓ B]	Processes, events, communication aspects, failures, time management and level of synchrony. These are the elements that define a system model: what kind of **processes** we should assume, which **events** should be considered in the algorithm being defined (internal, input, output...) since they will define the interface of that algorithm, which **communication** mechanisms and interaction steps will be used by the involved agents, what types of **failures** may occur in that system and which consequences each kind of failure may have, how **time** and synchrony are handled by the agents, which level of **synchrony** is assumed in the interactions among agents...
C	Physical layer, data link layer, network layer, transport layer and application layer. No. These are the common layers in a standard computer communications architecture.
D	Database management system, middleware and user interface. No. These are examples for implementing a three-layered client/server architecture.
E	All the above.
F	None of the above.

7. In distributed computing, the use of middleware is recommended because...

A	It introduces multiple transparencies, hiding low-level details and providing a uniform interface.
B	It facilitates the implementation of programmes and reduces the complexity of the elements being handled.
C	It provides a standardised, well understood and well defined way of doing things.
D	It improves interoperability; i.e., the interaction with products from third parties.
[✓ E]	All the above. These are four of the advantages being introduced by a middleware layer in a distributed system architecture.
F	None of the above.

8. Problems in distributed object-oriented systems:

A	All objects seem to be local to their caller and this may hide lengthy invocation intervals. This is a problem since the time needed for completing an object invocation is difficult to forecast in that environment.
B	Objects maintain state and that state is shared by all agents that are invoking their methods. This may lead to consistency problems. Shared state may lead to race conditions and race conditions may introduce state inconsistencies. Besides this, when an object is replicated, its methods are invoked by many agents and those agents are served by different replicas, the updates being generated by those calls may, again, generate inconsistencies among the replicas.
C	Their shared state needs concurrency control mechanisms. Those mechanisms lead to blocking intervals that prevent these systems from being scalable. Shared state defines critical sections. Those critical sections should be protected by a CS-entry and a CS-exit protocol. Those protocols are regularly implemented using concurrency control mechanisms (e.g., locks, semaphores, monitors...) and those mechanisms block the running agents when needed. Therefore, this is an important problem when we want to develop scalable services.
D	Their invocation mechanisms provide a high degree of location transparency. This fact demands complex recovery protocols in order to deal with failures. When a server fails, it should be replaced by another of its replicas, preserving location transparency. This means that the recovery protocol should react as fast as possible. The protocols needed to ensure that fast reaction and recovery aren't trivial and they depend on the replication model (active, passive or any of their intermediate variants) being used.
[✓ E]	All the above.
F	None of the above.

SEMINARS

9. **Considering this (incomplete) node programme:**

```
function logarithm(x,b) { return Math.log(x)/Math.log(b) }
function logBase ... // to be completed
log2 = logBase(2);
log8 = logBase(8);
console.log("Logarithm base 2 of 1024 = " + log2(1024));
console.log("Logarithm base 8 of 4096 = " + log8(4096));
```

Which implementation of `logBase()` is correct?

A	`function logBase(b) { return logarithm(x,b) }`

B	`function logBase(b) {`
	`   return function(x) { return logarithm(x,b) }`
	`}`
	`logBase()` should be a function that returns another function as its result (since both `log2()` and `log8()` are functions in this example). Moreover, its returned function should remember the argument that has been used in the call to `logBase()` using it as the base of the logarithm to be computed. Besides this, its parameter should be the number (i.e., “x”) on which to compute its logarithm in base b. This part B complies with all these requirements.
	
	Part A does not return a function but a value.
	
	Part C returns a function that does not return any thing.
	
	Part D returns a function but that function expects the logarithm base as its parameter and we need a function that expects “x” as its parameter instead of “b”.

C	`function logBase(x) {`
	`   return function(b) { logarithm(x,b) }`
	`}`

D	`function logBase(x) {`
	`   return function(b) { return logarithm(x,b) }`
	`}`

E	All the above.
F	None of the above.

10. Considering this node programme:

```
var fruits = ["Banana", "Orange", "Lemon", "Apple"];
var numbers = [7, 3, "Cloud", 9];
var funcs = [function(x) {return 2*numbers[x]},
             function(x) {return fruits[x]}];
var s = "";
for (var i=0; i<2; i++)
  for (var j=0; j<5; j=j+2)
     s += funcs[i](j) + ", ";
console.log(s);
```

Its output (once it is run) will be:

A	14, 6, NaN, Banana, Orange, Lemon,
B	14, NaN, NaN, Banana, Lemon, undefined, This programme consists of two nested loops. The external one (variable "i") defines two iterations, assigning values 0 and 1 to "i". The internal one (variable "j") defines three iterations, assigning values 0, 2 and 4 to "j". In each iteration its single statement concats to string "s" (that is initially empty) the result of calling the function "funcs[i]" passing as its single argument the current value of "j". Function funcs[0] returns the result of two times the value contained in numbers[j]. That value should be a number (otherwise, the result of using the operator "*" is NaN). Function funcs[1] returns component "j" in array fruits[]. Therefore, while i is 0, the values returned in the call to funcs[0] are: twice 7 (i.e., 14), twice "Cloud" (i.e., NaN) and twice undefined (i.e., NaN). While i is 1, the values returned in calls to funcs[1] are "Banana" (i.e., fruits[0]), "Lemon" (i.e., fruits[2]) and undefined (i.e., fruits[4]). As a result, the sequence obtained in this execution is: "14, NaN, NaN, Banana, Lemon, undefined," ...and this means that the correct answer is B.
C	14, 2Cloud, undefined, Banana, Lemon, undefined,
D	Banana, 14, Lemon, NaN, undefined,
E	No output. Only an error message saying that the "numbers" array has been incorrectly defined, since it holds values of different types.
F	None of the above.

NOTE: The third element in the correct result is NaN. It is obtained multiplying 2 and *undefined* since the programme accesses to a nonexistent component of an array (numbers[4]). In spite of this, there is another apparently valid result: *undefined*. Therefore, we'll accept F (none of the above) as an acceptable (i.e., valid) response.

11. Considering the following node function:

```
function f(x,y) {
   x = x || 'orange'; y = y || 98;
   console.log('x='+x+' y='+y);
}
```

Please choose will be its output when it is invoked in the following ways:

	`f(36);`	`f(undefined,'apple');`	`f(45,0,67);`
A	x=36 y=98	x=undefined y=apple	x=45 y=0
B	x=36 y=98	x=orange y=apple	x=45 y=0
C	x=36 y=98	x=orange y=apple	x=45 y=98
	This question is about the '||' (OR) logical operator and the usage of arguments in function calls. The '||' operator returns its left operand when it is different to Boolean **false** and its right operand otherwise, but in JavaScript there are multiple false values. In the set of numbers, 0 is **false** and all other values are **true**. On the other hand, undefined is also considered **false**. Therefore, in the f(36) call we are using only one argument while f has two parameters. This means that parameter "y" receives an undefined value. As a result of this, the second statement in f ensures that "y" will be 98. So, **x=36 and y=98**. In call f(undefined, 'apple'), variable "x" will get value 'orange' since the latter is the right operand in the "x = x || 'orange'" statement. So, **x=orange** and **y=apple**. Finally, in call f(45,0,67), argument 67 (the third one) is ignored and y gets value 98 since 0 is equivalent to **false**. Therefore, **x=45** and **y=98**. Part C is the unique that presents the correct values in the three calls.		
D	x=36 y=36	x=orange y=apple	x=45 y=67
E	Nothing is shown besides some error messages since there are incorrect calls (due to an incorrect number of arguments) to function f.		
F	None of the above.		

12. Considering the following node programme...

```
var eve = new (require('events')).EventEmitter;
var s = "print";
var n = 0;
var handler = setInterval( function(){eve.emit(s);}, 1000 );
eve.on(s, function() {
   if ( n < 2 ) console.log("Event", s, ++n, "times.");
   else clearInterval(handler);
});
```

Please select the correct choice regarding programme output (first part) and execution time (second part):

A	Event print 1 times. Event print 2 times.	The programme is terminated in 3 seconds.
	This programme generates a "print" event every second (line 4). In the "print" listener, it checks whether n is lower than 2 (initially 0) and if that is true, a message is printed (starting with n=1 since that variable is pre-increased in that console.log statement). Otherwise the "print" event emission interval is cleared. When that happens, the programme terminates since there is no other event to be managed nor pending turn. Therefore, when the programme is started, its line 4 sets an interval of 1 second for raising consecutive "print" events. One second later, message "Event print 1 times" is printed. Again, one second later, message "Event print 2 times" is printed. At the third second, the listener checks whether n is lower than 2 and now n is 2. So, it takes the else path and it clears the interval. As a result of this, the programme terminates in 3 seconds but only two messages have been printed.	
B	Event print 1 times. Event print 2 times. Event print 3 times.	The programme is terminated in 4 seconds.
C	Event print 1 times. Event print 2 times. …	The programme is never terminated. Each second it shows a new line with an increased number.
D	Event print 0 times. Event print 1 times. …	The programme is never terminated. Every 10 seconds it shows a new line with an increased number.
E	Nothing is shown since the listener has not been defined correctly.	The programme is never terminated, since it emits cyclically the "print" event.
F	None of the above.	

13. Considering this node.js programme...

```
var http = require('http');
var fs = require('fs');
http.createServer(function(request,response) {
  fs.readdir(__dirname, function(err,data) {
    if (err) {
      response.writeHead(404, {'Content-Type':'text/plain'});
      response.end('Unable to read directory '+__dirname);
    } else {
      response.writeHead(200, {'Content-Type':'text/plain'});
      response.write('Directory: ' + __dirname + '\n');
      response.end(data.toString());
    }
  })
}).listen('1337');
```

The following sentences are true:

A	This programme raises an exception and aborts when it is unable to read the contents of the current directory. No. If it finds any error trying to read that directory, the callback in the readdir() call will receive an object in its first parameter and this web server will return a response to its requesting client without raising any exception or aborting.
B	This programme is a web server that replies with the name and list of files in its current directory. This is a correct description of the tasks developed by this programme. The http.createServer() operation provides the base to write an HTTP server. Independently on the request being receive, this server always replies with a response that contains the name and contents of the directory where it has been started. The name of that directory is contained in the __dirname global variable.
C	This programme does not work since it has not declared variable "__dirname" and it has not imported the 'process' module where such variable is defined. No. "__dirname" is a variable that is already declared by default. You do not need to import any module, nor declare it, to use it.
D	This programme does not work since 'data' is an array of file names and arrays cannot be cast into strings. False. An array of strings (since file names are strings) can be cast to a string without any problem.
E	All the above.
F	None of the above.

14. Some problems of the central server algorithm for mutual exclusion are...

A	It does not respect its liveness condition. All correct distributed algorithms should satisfy their safety and liveness conditions. This algorithm respects its liveness condition; i.e., it ensures that all requestors eventually enter the critical section.
B	It does not respect its safety condition. All correct distributed algorithms should satisfy their safety and liveness conditions. This algorithm respects its safety condition; i.e., it ensures that never two or more processes could be simultaneously in the critical section.
C	It requires more messages than the other algorithms seen in Seminar 2 for solving the mutual exclusion problem. No. Indeed this is one of the algorithms requiring a minimal amount of messages for managing a critical section.
[✓ D]	It is fragile when failures arise. The central server is a single point of failure. Yes. If the central server crashes none of the participants may progress. None of the requestors could enter the critical section in that case.
E	All the above.
F	None of the above.

15. Leader election algorithms...

A	...are a subset of consensus algorithms. Yes. In order to choose a leader the processes must reach a consensus on which is the best candidate.
B	...require that all processes have a different identifier. Yes. Otherwise it would be impossible to elect any of them since the decision is based on the identifiers of each process.
C	...use a deterministic criterion to select the leader. Yes, and this criterion should be known by all participating processes.
D	...require that only one process is elected. Yes. The leader must be unique.
[✓ E]	All the above.
F	None of the above.

16. We want to implement a chat service using node.js and ØMQ. The server multicasts the user messages and should never block trying to send a message (of any kind). The client programmes send user messages to the server, wait for user messages forwarded by the server and report to the server when a user enters or abandons the system. In order to implement this chat service...

A	The server must use a PULL and a REP socket to interact with clients. No. User messages should be multicast. Only PUB sockets can multicast a message using a single send() operation.

B	The server must use a SUB and a REQ socket to interact with clients. No. User messages should be multicast. Only PUB sockets can multicast a message using a single send() operation.
C	The server must use a PUB and a PULL socket to interact with clients. Yes. User messages should be multicast. Only PUB sockets can multicast a message using a single send() operation. Besides this, another socket is needed in order to accept and process the messages sent by the client processes to tell the server that a user has joined the system or wants to leave it. This second socket should be able to receive messages and the PULL one can do this without blocking the server.
D	The server must use a REP and a SUB socket to interact with clients. No. User messages should be multicast. Only PUB sockets can multicast a message using a single send() operation.
E	All the above.
F	None of the above.

17. Let us assume that we have implemented a service supported by several (e.g., 10) server processes placed in different computers. Those servers use REP sockets and their clients use REQ sockets. If we build an intermediate broker with a front-end ROUTER socket and a back-end DEALER socket (both bound to static URLs), then...

A	Clients do not need to know how many servers exist. Yes. The broker is the single process that directly interacts with servers. Therefore client processes do not need any information about server processes.
B	Clients do not need to know the addresses and ports of each server process. Yes. The broker is the single process that directly interacts with servers. Therefore client processes do not need any information about server processes.
C	The amount of server processes may be dynamically changed. They need to connect to the back-end socket in order to be available for the broker. Yes. We may modify the amount of servers in a transparent way. They only need to connect to the DEALER socket.
D	The broker does not need to modify any message segment in order to correctly forward messages from front-end to back-end and from back-end to front-end. Yes. Both ROUTER and DEALER sockets do not need to worry about the message contents. None of the message segments needs to be modified (nor added or removed). With this strategy the DEALER sockets distributed in a circular way the request messages among all connected servers.
E	All the above.
F	None of the above.

In order to answer the following two questions (i.e., 18 and 19), please consider the following node programmes with ZeroMQ. A server (*server.js*)...

```
var zmq = require('zmq')
var rep = zmq.socket('rep')
rep.bindSync('tcp://127.0.0.1:'+process.argv[2])
var n = 0
rep.on('message', function(msg) {
  console.log('Request: ' + msg)
  rep.send('World ' + ++n)
})
```

...and a client (*client.js*)...

```
var zmq = require('zmq')
var req = zmq.socket('req')
req.connect('tcp://127.0.0.1:'+process.argv[2])
req.connect('tcp://127.0.0.1:'+process.argv[3])
var n = 0
setInterval( function() { req.send('Hello ' + ++n) }, 100 )
req.on('message', function(msg) {
  console.log('Response: ' + msg)
})
```

---

18. Considering the programmes (*server.js* and *client.js*) shown previously... Let us assume that 2 servers and 1 client have been started in three consoles using:

```
node server 8001
node server 8002
node client 8001 8002
```

The first lines printed in the server consoles will be:

A	In a console: `Request: Hello 1` `Request: Hello 3` And in the other: `Request: Hello 2` `Request: Hello 4` Since the REQ socket in the client is connected to all REP sockets maintained by servers, that REQ socket distributes in a circular way its sent messages. This means that it sends th first request to the first server, the second request to the second server, the third request again to the first server, and so on... As a result of this, the messages are printed as seen in this part since each request increases the same n local counter and the first message had value 1 for that counter.
B	In both consoles: `Request: Hello 1` `Request: Hello 2` No. See explanation in part A.
C	In both consoles: (being x, y, z... numbers such that x < y < z < ...): `Request: Hello x` `Request: Hello y` `Request: Hello z` `...` No. See explanation in part A.
D	In a console: `Request: Hello 1` `Request: Hello 2` And in the other: `Request: Hello 3` `Request: Hello 4` No. See explanation in part A.
E	Nothing is printed since the client is unable to decide to which server it must send its requests. (To fix this problem, the client should be connected to a ROUTER socket.) No. See explanation in part A.
F	None of the above.

19. Considering again those two programmes and the same execution scenario than in the previous question... The first lines printed in the client console will be:

A	`Response: World 1`
	`Response: World 2`
	`Response: World 3`
	`Response: World 4`
	No. See the explanation in part B.

B	`Response: World 1`
	`Response: World 1`
	`Response: World 2`
	`Response: World 2`
	Yes. As its has been explained in question 18, the client is sending each consecutive request to a different server, balancing the load in both of them. Each server has a local counter for assigning a number to each response. Therefore, the first response has number 1 and is sent by the first server, the second response has again number 1 and is sent by the second server, the third response is sent by the first server with number 2, and so on.

C	`Response: World 1`
	`Response: World 3`
	`Response: World 5`
	`Response: World 7`
	No. See the explanation in part B.

D	`Response: World 1`
	`Response: World 3`
	`Response: World 2`
	`Response: World 4`
	No. See the explanation in part B.

E	Nothing is printed. Since the client cannot send its requests, no server will answer. No. See the explanation in part B.
F	None of the above.

20. Considering the following programmes... A publisher:

```
var zmq = require('zmq')
var pub = zmq.socket('pub').bindSync('tcp://*:5555')
var count = 0
setInterval(function() {
  pub.send('PRG ' + count++)
  pub.send('TSR ' + count++)
}, 1000)
```

And a subscriber:

```
var zmq = require('zmq')
var sub = zmq.socket('sub')
sub.connect('tcp://localhost:5555')
sub.subscribe('TSR')
sub.on('message', function(msg) {
  console.log('Received: ' + msg)
})
```

Assuming that the publisher has been launched first and, three seconds later, we have started the subscriber... The first lines shown in the subscriber

console will be:

A	`Received: TSR 1`
	`Received: TSR 2`
	`Received: TSR 3`
	…
	No. See the explanation in part D.

B	`Received: TSR 1`
	`Received: TSR 3`
	`Received: TSR 5`
	…
	No. See the explanation in part D.

C	`Received: PRG 4`
	`Received: TSR 5`
	`Received: PRG 6`
	…
	No. See the explanation in part D.

D	`Received: TSR 5`
	`Received: TSR 7`
	`Received: TSR 9`
	…
	Yes. Note that the publisher sends two messages every second, each one with a different prefix, but using all of them the same counter. So, the messages sent by this publisher are...
	At second 1: PRG 0, TSR 1
	At second 2: PRG 2, TSR 3
	At second 3: PRG 4, TSR 5
	At second 4: PRG 6, TSR 7
	
	The subscriber demands only messages with the TSR prefix. It is started at time 3 seconds. Since the PUB-SUB channels aren't strongly persistent, all messages multicast by the publisher before the subscriber is connected cannot be delivered to the latter. Therefore it will receive all TSR messages starting from 5; i.e., TSR 5, TSR 7, TSR 9, etc.

E	`Received: PRG 0`
	`Received: TSR 1`
	`Received: PRG 2`
	…
	No. See the explanation in part D.

F	None of the above.