# TSR - Rec_Primer Parcial. 2025-01-30

Aquest examen consta de 17 qüestions, amb una puntuació total de 10 punts. Cada qüestió té 4 alternatives, de les quals únicament una és certa. La nota es calcula de la següent manera: després de descartar la pitjor qüestió, cada encert suma 10/16 punts, i cada error descompta 10/48 punts. Has de contestar en el full de respostes.

A

1. ***Aquesta aplicació pot considerar-se un servei distribuït quan es desplega en l'entorn esmentat:***
   - **A.** Un compilador de Java en ser utilitzat en tots els ordinadors d'un mateix laboratori per a resoldre una mateix pràctica d'alguna assignatura de programació
   - **B.** Microsoft Word en Office 365 quan és utilitzada per múltiples usuaris en els seus respectius ordinadors per a editar un mateix document compartit entre tots ells
   - **C.** L'intèrpret d'ordres bash en ser utilitzat per un usuari al seu ordinador per a iniciar aplicacions localment, sense utilitzar la xarxa en elles
   - **D.** Totes les afirmacions són certes

2. ***En estudiar l'evolució dels serveis de programari s'arriba a identificar una etapa on el seu model de servei aconsegueix automatitzar la monitorització de paràmetres rellevants, permet expressar punts d'elasticitat i automatitza la reconfiguració del servei en funció de la càrrega existent. Eixe model de servei és:***
   - **A.** SaaS
   - **B.** PaaS
   - **C.** Cluster altament disponible
   - **D.** IaaS

3. ***Quan s'analitzen els paradigmes de programació s'indica que per a què un servei siga escalable, els seus servidors no haurien de suspendre's al gestionar cada petició. Això pot proporcionar-se adoptant aquest paradigma:***
   - **A.** Multi-fil, ja que encara que els seus fils puguen compartir recursos, això no conduirà al bloqueig de cap activitat
   - **B.** Dirigit per esdeveniments, associant un fil a cada esdeveniment i compartint memòria entre tots ells
   - **C.** Multi-fil, ja que els múltiples fils que es puguen generar no compartiran mai recursos
   - **D.** Asincrònic, o dirigit per esdeveniments, ja que els nous esdeveniments generats poden mantenir-se en una cua, sense interrompre o bloquejar l'activitat en curs

4. ***Què es mostra en pantalla quan s'execute el següent programa?***

```
const k=2
if (k==1)
  console.log("k=1")
else {
  var j=3
  console.log("j-k=" + (j-k))
}
console.log("j="+j)
```

   - **A.** Aquest contingut: `j-k=1, ...Reference error: j is not defined...`
   - **B.** Un error indicant que no es pot restar una constant a una variable
   - **C.** Dues línies amb: `j-k=1, j=3`
   - **D.** Dues línies amb: `k=1, j=undefined`

5. ***Què es mostra en pantalla quan s'execute el següent programa?***

```
const k=2
if (k==1)
  console.log("k=1")
else {
  let j=3
  console.log("j-k=" + (j-k))
}
console.log("j="+j)
```

**A.** Un error indicant que no es pot restar una constant a una variable
**B.** Dues línies amb: `j-k=1, j=3`
**C.** Dues línies amb: `k=1, j=undefined`
**D.** Aquest contingut: `j-k=1, ...Reference error: j is not defined...`

6. ***Què es mostra en pantalla quan s'execute el següent programa?***

```
function greet( x="John", y ) {
  console.log("Hi, "+x); console.log("Hi, "+y)
}
greet(undefined, "Mary", "Peter")
```

**A.** `Hi, Mary`
`Hi, Peter`
**B.** `Hi,`
`Hi, Mary`
**C.** `Hi, John`
`Hi, Mary`
**D.** Un error, ja que la funció `greet` espera dos arguments i n'estem passant tres

7. ***Quin missatge mostra primer aquest programa, quants callbacks s'utilitzen en ell i quantes clausures?***

```
function generateF(x) {
  return function () {
    console.log("Writing after "+x+" seconds.")
  }
}
setTimeout(generateF(0), 0)
console.log("End!")
```

**A.** `Writing after 0 seconds`, amb un callback i una clausura
**B.** `End!`, amb un callback i una clausura
**C.** `Writing after 0 seconds`, sense callbacks i sense clausures
**D.** `Writing after 0 seconds`, sense callbacks i amb una clausura

8. ***Considere que aquest programa s'inicia en un ordenador on ja s'executa un servidor net que atèn connexions al port 9000 i que aquest últim sempre torna alguna resposta a qualsevol petició rebuda. Seleccione l'afirmació certa sobre aquest programa client:***

```
const net=require('net')
let counter = 0
let client = net.connect({port:9000},
  function () {
    console.log("client connected!")
    client.write(counter+' world')
  })
client.on('data', function (data) {
  console.log(data.toString())
  if (counter == 9) client.end()
  else client.write((++counter) +' world')
})
client.on('end', function () {
  console.log("client disconnected")
})
```

**A.** El codi utilitzat en aquest client per a enviar les seues peticions garanteix persistència dèbil en la comunicació resultant
**B.** Aquest client finalitza en rebre la resposta a la seua dècima petició
**C.** Si se substituira a l'octava línia el primer argument del mètode `on` per la cadena `message`, el programa seguiria funcionant d'igual manera
**D.** Aquest client és incapaç d'interactuar amb cap servidor, ja que no arriba a enviar-li cap missatge

9. ***Aquest és un exemple de middleware:***
**A.** Ubuntu 24.10
**B.** Node.js 22.12.0
**C.** LibreOffice Writer 24.8
**D.** Apache ActiveMQ Classic 6.1.4

10. ***Selecciona l'afirmació correcta sobre els sistemes de comunicació no persistent:***
**A.** Si utilitzen un gestor són més eficients que quan no el necessiten
**B.** Si no utilitzen un gestor són més eficients que quan el necessiten
**C.** Solen mantenir els missatges en cues de l'emissor quan el receptor no estiga disponible
**D.** Exigeixen que el receptor estiga preparat per a que l'emissor transmeta els seus missatges

11. ***Si `so` és un socket ZeroMQ de tipus REQ, aleshores al fer l'operació `so.send("Exemple")`:***
    - **A.** Eixe enviament quedarà bloquejat mentres no es reba prèviament alguna petició des d'un altre socket REP, ja que REQ només serveix per a contestar i no pot iniciar la comunicació
    - **B.** Al transmetre el missatge al receptor, tindrà dos segments: la identitat de `so` i `"Exemple"`
    - **C.** El missatge `"Exemple"` romandrà necessàriament en alguna cua d'eixida durant un interval llarg, ja que el patró REQ-REP és sincrònic i bloquejant
    - **D.** Al transmetre el missatge al receptor, tindrà dos segments: `""` i `"Exemple"`

12. ***Es vol desenvolupar, utilitzant sockets ZeroMQ, un servei de distribució de notícies format per tres components:***
    ***(a) corresponsal, especialitzat en una determinada temàtica i que emetrà els seus missatges utilitzant dos segments en ells (temàtica i text de la notícia),***
    ***(b) agència, que rebrà la informació dels corresponsals i la reenviarà a la cadena o cadenes interessades i***
    ***(c) cadena, que mitjançant un procés periòdic es connectarà durant cert interval a l'agència per a rebre la informació d'interés per als seus potencials usuaris. Una mateixa cadena pot estar interessada en diferents temàtiques.***
    ***La comunicació serà unidireccional: l'agència no respon mai al corresponsal i tampoc ho fa la cadena a l'agència. La informació que envie un corresponsal no ha de perdre's si l'agència encara no ha iniciat la seua activitat. Al contrari, la cadena només està interessada en la informació que l'agència publique des del moment en què faça la seua connexió a l'agència, descartant els missatges que l'agència haja publicat abans.***
    **Quins tipus de sockets podrien utilitzar-se per a desenvolupar aquest servei?**
    - **A.** Corresponsal: PUSH; Agència: SUB i PUB; Cadena: PULL
    - **B.** Corresponsal: REQ; Agència: REP i PULL; Cadena: PUSH
    - **C.** Corresponsal: PUB; Agència: SUB i REQ; Cadena: REP
    - **D.** Corresponsal: PUSH; Agència: PULL i PUB; Cadena: SUB

13. ***Considere el següent programa:***

```
const zmq = require("zeromq")
const sub = zmq.socket('sub')

sub.connect("tcp://localhost:5555")
sub.on("message", function(msg) {
  console.log("Received: " + msg)
})
```

    **S'ha iniciat un procés subscriptor que executa aquest programa i en eixe mateix ordinador s'ha iniciat prèviament un procés que utilitza un socket PUB `p`, sobre el que s'ha fet amb èxit l'operació `p.bind("tcp://*:5555")`. Eixe procés publicador emet un missatge cada segon, de manera ininterrompuda, el contingut del qual són notícies breus de text, entre 50 i 80 caracters. No obstant això, el procés subscriptor no aconsegueix rebre ni mostrar cap missatge. Per què?**
    - **A.** La gestió de les connexions és errònia, perquè els sockets SUB han de fer `bind` i els PUB `connect`
    - **B.** L'esdeveniment a utilitzar per a rebre els missatges ha de ser `"data"` en comptes de `"message"`
    - **C.** El programa publicador ha d'usar `"tcp://localhost:5555"` com a argument de l'operació `bind`, en comptes de la URL que ha usat
    - **D.** El programa subscriptor ha d'incloure una instrucció `sub.subscribe("")` per a rebre els missatges

14. ***Un procés A emet, usant un socket PUSH amb un `bind("tcp://localhost:8888")` i `setInterval(...,1000)`, sis missatges el contingut dels quals és, respectivament, `"1", "2", "3", "4", "5"` i `"6"`, abans de ser finalitzat per l'usuari. En ser iniciat i abans de que transcòrrega el primer segon, tres processos B, C i D s'han connectat, en eixe ordre, a eixe port 8888 i reben missatges amb un socket PULL. Quins missatges rep cada procés?***
    - **A.** B: "1" i "4"; C: "2" i "5"; D: "3" i "6"
    - **B.** B, C i D reben, cadascun, tots els missatges
    - **C.** És imprevisible quin conjunt de missatges rebrà cadascun
    - **D.** Cap, ja que haurien d'haver usat un altre tipus de socket per a que la recepció fóra possible

15. ***Donat el següent programa:***

```
function f1 (a,b,c) {
  return a+b+c
}
let vector = [1,2,3,4]
console.log ("resultv1: " + f1 (...vector))
```

**Tria quin missatge mostra en pantalla:**

**A.** `resultv1: 1,2,3,4undefinedundefined`

**B.** `Uncaught ReferenceError: ...vector is not defined`

**C.** `resultv1: 6`

**D.** `resultv1: [1,2,3,4]`

16. ***A la segona sessió de la pràctica 1 es demanava ampliar un parell de programes netClient.js i netServer.js per a que el segon tornara al primer un valor proporcional a la seua càrrega en eixe moment. Per a fer això, la versió ampliada de netClient.js necessitava rebre dos arguments des de la línia d'ordres: la IP del servidor i la IP del client, en eixe ordre. Amb quines instruccions podríem accedir a eixos arguments?***

**A.** Totes les opcions són correctes

**B.**
```
let args = process.args.slice(2)
let ipServer = args[0]
let ipClient = args[1]
```

**C.**
```
let ipServer = process.argv[2]
let ipClient = process.argv[3]
```

**D.**
```
let pa = argv
let ipServer = pa[2]
let ipClient = pa[3]
```

17. ***A la tercera sessió de la pràctica 1 es demanava revisar o implementar tres tipus de proxies: bàsic, configurable i programable. Qualsevol d'ells rebia els missatges emesos per un navegador web (o qualsevol altre procés client que utilitzara connexions TCP) per a reenviar-los a un determinat servidor (normalment, un lloc web). Quina és la principal diferència entre el bàsic i el configurable?***

**A.** El bàsic només permet gestionar un servidor, mentres que el configurable pot interactuar amb dos servidors simultàniament

**B.** El bàsic no gestiona cap memòria cau, mentres que al configurable podem configurar la grandària de la memòria cau de pàgines recents que mantindrà

**C.** El codi del bàsic manté l'adreça i port del servidor en constants, mentres que el configurable rep eixa informació des de la línia d'ordres

**D.** El bàsic rep l'adreça i port del servidor des de la línia d'ordres, mentres que el configurable rep eixa informació en un missatge de configuració