Programació de serveis i processos - Sebastian Ciscar

None

None

Table of contents

1. Presentacio

1.1 Presentació del mòdul

El mòdul de *PRogramació de Serveis i Processos* pertany al 2n curs del CFGS de Desenvolupament d'Aplicacions Multiplataforma, i contribueix a adquirir diversesde les unitats de competència que s'hi estableixen al Catàleg Nacional de Qualificacions Professionals.

Segons el RD del títol, la formació en aquest mòdul contribueix a assolir els següents objectius del mòdul:

- b) Identificar les necessitats de seguretat analitzant vulnerabilitats i verificant el pla * preestablit per a aplicar tècniques i procediments relacionats amb la seguretat en el sistema.
- e) Seleccionar i emprar llenguatges, eines i llibreries, interpretant les especificacions per a desenvolupar aplicacions multiplataforma amb accés a bases de dades.
- i) Seleccionar i emprar tècniques, motors i entorns de desenvolupament, avaluant les seues possibilitats, per a participar en el desenvolupament de jocs i aplicacions en l'àmbit de l'entreteniment.
- l) Valorar i emprar eines específiques, atenent a l'estructura dels continguts, per a crear tutorials, manuals d'usuari i altres documents associats a una aplicació.
- n) Analitzar i aplicar tècniques i llibreries específiques, simulant diferents escenaris, per a desenvolupar aplicacions capaces d'oferir serveis en xarxa.
- ñ) Analitzar i aplicar tècniques i llibreries de programació, avaluant la seua funcionalitat per a desenvolupar aplicacions multiprocés i multithread.
- t) Descriure els rols de cadascun dels components del grup de treball, identificant en cada cas la responsabilitat associada, per a establir les relacions professionals més convenients.
- w) Identificar els canvis tecnològics, organitzatius, econòmics i laborals en la seua activitat, analitzant les seues implicacions en l'àmbit de treball, per a mantindre l'esperit d'innovació.

1.2 Objectius. Resultats d'aprenentatge

D'acord amb la normativa, els resultats d'aprenentatge del mòdul de *PRogramació de Serveis i Processos* són:

RA1	Desenvolupa aplicacions compostes per diversos processos reconeixent i aplicant principis de programació paral·lela.
RA2	Desenvolupa aplicacions compostes per diversos fils d'execució analitzant i aplicant llibreries específiques del llenguatge de programació.
RA3	Programa mecanismes de comunicació en xarxa emprant sockets i analitzant l'escenari d'execució.
RA4	Desenvolupa aplicacions que ofereixen serveis en xarxa, utilitzant llibreries de classes i aplicant criteris d'eficiència i disponibilitat.
RA5	Protegeix les aplicacions i les dades definint i aplicant criteris de seguretat en l'accés, emmagatzematge i transmissió de la informació.

1.3 Resultats d'aprenentatge i criteris d'avaluació

RA1	Desenvolupa aplicacions compostes per diversos processos reconeixent i aplicant principis de programació paral·lela.	**RA1.a)** S'han reconegut les característiques de la programació concurrent i els seus àmbits d'aplicació. **RA1.b)** S'han identificat les diferències entre programació paral·lela i programació distribuïda, els seus avantatges i inconvenients. **RA1.c)** S'han analitzat les característiques dels processos i de la seua execució pel sistema operatiu. **RA1.d)** S'han caracteritzat els fils d'execució i descrit la seua relació amb els processos. **RA1.e)** S'han utilitzat classes per programar aplicacions que creen subprocessos. **RA1.f)** S'han utilitzat mecanismes per compartir informació amb els subprocessos iniciats. **RA1.g)** S'han utilitzat mecanismes per sincronitzar i obtenir el valor retornat pels subprocessos iniciats. **RA1.h)** S'han desenvolupat aplicacions que gestionen i utilitzen processos per a l'execució de diverses tasques en paral·lel. **RA1.i)** S'han depurat i documentat les aplicacions desenvolupades.
RA2	Desenvolupa aplicacions compostes per diversos fils d'execució analitzant i aplicant llibreries específiques del llenguatge de programació.	**RA2.a)** S'han identificat situacions en les quals resulta útil la utilització de diversos fils en un programa. **RA2.b)** S'han reconegut els mecanismes per crear, iniciar i finalitzar fils. **RA2.c)** S'han programat aplicacions que implementen diversos fils. **RA2.d)** S'han identificat els possibles estats d'execució d'un fil i programat aplicacions que els gestionen. **RA2.e)** S'han utilitzat mecanismes per compartir informació entre diversos fils d'un mateix procés. **RA2.f)** S'han desenvolupat programes formats per diversos fils sincronitzats mitjançant tècniques específiques. **RA2.g)** S'ha establert i controlat la prioritat de cadascun dels fils d'execució. **RA2.h)** S'han depurat i documentat els programes desenvolupats. **RA2.i)** S'ha analitzat el context d'execució dels fils. **RA2.j)** S'han analitzat llibreries específiques del llenguatge de programació que permeten la programació multihil. **RA2.k)** S'han reconegut els problemes derivats de la compartició d'informació entre els fils d'un mateix procés.
RA3	Programa mecanismes de comunicació en xarxa emprant sockets i analitzant l'escenari d'execució.	**RA3.a)** S'han identificat escenaris que precisen establir comunicació en xarxa entre diverses aplicacions. **RA3.b)** S'han identificat els rols de client i de servidor i les seues funcions associades. **RA3.c)** S'han reconegut llibreries i mecanismes del llenguatge de programació que permeten programar aplicacions en xarxa. **RA3.d)** S'ha analitzat el concepte de socket, els seus tipus i característiques. **RA3.e)** S'han utilitzat sockets per programar una aplicació client que es comunique amb un servidor. **RA3.f)** S'ha desenvolupat una aplicació servidor en xarxa i verificat el seu funcionament. **RA3.g)** S'han desenvolupat aplicacions que utilitzen sockets per intercanviar informació. **RA3.h)** S'han utilitzat fils per possibilitar la comunicació simultània de diversos clients amb el servidor. **RA3.i)** S'han caracteritzat els models de comunicació més usuals en les arquitectures d'aplicacions distribuïdes. **RA3.j)** S'han depurat i documentat les aplicacions desenvolupades.
RA4	Desenvolupa aplicacions que ofereixen serveis en xarxa, utilitzant llibreries de classes i aplicant criteris d'eficiència i disponibilitat.	**RA4.a)** S'han identificat diferents protocols estàndard de comunicació per a la implementació de serveis en xarxa. **RA4.b)** S'han reconegut els avantatges de la utilització de protocols estàndard per a la comunicació entre aplicacions i processos. **RA4.c)** S'han analitzat llibreries que permeten implementar serveis en xarxa utilitzant protocols estàndard de comunicació. **RA4.d)** S'han desenvolupat i provat serveis de comunicació en xarxa. **RA4.e)** S'han utilitzat clients de comunicacions per verificar el funcionament dels serveis. **RA4.f)** S'han incorporat mecanismes per possibilitar la comunicació simultània de diversos clients amb el servei. **RA4.g)** S'ha verificat la disponibilitat del servei. **RA4.h)** S'han depurat i documentat les aplicacions desenvolupades.
RA5	Protegeix les aplicacions i les dades definint i aplicant criteris de seguretat en l'accés, emmagatzematge i transmissió de la informació.	**RA5.a)** S'han identificat i aplicat principis i pràctiques de programació segura. **RA5.b)** S'han analitzat les principals tècniques i pràctiques criptogràfiques. **RA5.c)** S'han definit i implantat polítiques de seguretat per limitar i controlar l'accés dels usuaris a les aplicacions desenvolupades. **RA5.d)** S'han utilitzat esquemes de seguretat basats en rols. **RA5.e)** S'han emprat algoritmes criptogràfics per protegir l'accés a la informació emmagatzemada. **RA5.f)** S'han identificat mètodes per assegurar la informació transmesa. **RA5.g)** S'han desenvolupat aplicacions que utilitzen comunicacions segures per a la transmissió d'informació. **RA5.h)** S'han depurat i documentat les aplicacions desenvolupades.

1.4 M'ho pots explicar millor?

La formació professional (FP) està dissenyada per preparar-vos per al món laboral, aportant-vos coneixements teòrics, i sobretot pràctics.

Cada mòdul formatiu cobreix un aspecte específic relacionat amb la vostra futura professió, i es divideix en diversos ***Resultats d'aprenentatge***. És a dir... què heu de *saber fer* després de passar pel mòdul.

A més, cadascun d'aquesta *Resultats d'Aprenentatge* es divideixen en diferents ***Criteris d'Avaluació***, que és el que s'usa pe avaluar-vos. Aquests criteris ens ajuden a veure si heu assolit els objectius.

Per què és important conèixer els resultats d'aprenentatge i els criteris d'avaluació?

- **Orientació**: Us ajuden a saber exactament què s'espera de vosaltres i què heu d'aprendre.
- **Autoavaluació**: Podeu utilitzar-los per avaluar el vostre propi progrés i veure en què necessiteu millorar.
- **Transparència**: Us donen una idea clara de com sereu avaluats, així no hi ha sorpreses al final del mòdul.

Com veieu, els *resultats d'aprenentatge* i els criteris d'avaluació són com un mapa que us guia durant el curs. I segons el qual, girarà tot el que es veu al mòdul.

⚠ **I com influeix això en la qualificació?**

- Els *Resultats d'aprenentatge* i els *Criteris d'Avaluació* ens serviran com a guia per avaluar els vostres coneixements i destresses.
- Heu de tindre en compte, que per superar un mòdul, **heu de tindre tots els resultats d'aprenentatge superats**.
- Cada resultat d'aprenentatge, influïrà en major o menor mesura en la qualificació final del mòdul.
- Alguns d'estos CAs seran avaluats en la formació en empresa

Ara que ja sabeu què són els *resultats d'aprenentatge i els criteris d'Avaluació*, podeu tornar a fer una ullada ràpida als apartats anteriors, per tal d'entendre millor de què va el mòdul!

1.5 Continguts del mòdul. Unitats didàctiques

Els continguts del mòdul s'organitzen en les següents unitats didàctiques:

- **UD1.** Programació multiprocés
- **UD2.** Programació multifil. Corrutines.
- **UD3.** Programació de comunicacions en xarxa. Serveis amb Kotlin.
- **UD4.** Programació concurrent amb NodeJS
- **UD5.** Serveis amb NodeJS

1.5.1 Continguts per unitat

Els continguts per cada unitat seran:

UD1. Programació multiprocés

- Els Processos i el sistema operatiu
- Executables. Processos. Serveis.
- Estats d'un procés.
- Fils.
- Dimonis i serveis
- Planificació de processos
- Comunicació entre processos.
- Gestió de processos. Monitoratge.
- Sincronització entre processos.
- Programació multiprocés
- Programació concurrent.
- Programació paral·lela i distribuïda.
- Les classes Runtime i Process
- Creació de processos a Java.
- ProcessBuilder
- Gestió de l'entrada i eixida de processos en Java
- Connexió de la E/S entre Java i processos
- Redireccions a fitxers, mitjançant E/S i mitjançant ProcessBuilder
- Programació multiprocés
- Programació d'aplicacions multiprocés
- Creació de processos del sistema
- Anàlisi dels principals mecanismes per a la creació de processos
- Gestió de l'entrada i eixida de processos.

UD2. Programació multifil. Corrutines.

- Threads
- Les classe Thread i Runtime
- Estats d'un fil. Canvis d'estat.
- Context d'execució. Recursos compartits.
- Elements relacionats amb la programació de fils. Llibreries i classes.
- La interfície Runnable
- Mètodes sincronitzats
- Recursos compartits pels fils.
- Gestió de fils. Prioritats.
- Sincronització de fils.
- Compartició d'informació entre fils. Problemes.
- Coordinació de Threads
- El problema del productor-consumidor
- Programació d'aplicacions multifil.
- Corrutines en Kotlin

UD3. Programació de comunicacions en xarxa. Serveis amb Kotlin.

- Programació de comunicacions en xarxa:
- Comunicació entre aplicacions.
- Rols client i servidor.
- Llibreries per a la programació d'aplicacions en xarxa. Sockets.
- La classe URL
- Elements de programació d'aplicacions en xarxa. Llibreries.
- Sockets. Tipus. Característiques.
- Creació de sockets.
- Enllaçat i establiment de connexions.
- Utilització de sockets per a la transmissió i recepció d'informació.
- Aplicacions client-servidor
- Programació d'aplicacions client i servidor.
- Utilització de fils en la programació d'aplicacions en xarxa.
- Comunicacions i programació multifil
- Utilització de tècniques de programació segura:
- Pràctiques de programació segura.
- Criptografia de clau pública i clau privada.
- Principals aplicacions de la criptografia
- Protocols criptogràfics.
- Política de seguretat.
- Programació de mecanismes de control d'accés.
- Encriptació d'informació.
- Protocols segurs de comunicacions.
- Programació d'aplicacions amb comunicacions segures.

UD4. Programació concurrent amb NodeJS

- El framework NodeJS
- Programació de comunicacions en xarxa:
- Comunicació entre aplicacions.
- Rols client i servidor.
- Llibreries per a la programació d'aplicacions en xarxa. Sockets.
- La classe URL
- Elements de programació d'aplicacions en xarxa. Llibreries.
- Sockets. Tipus. Característiques.
- Creació de sockets.
- Enllaçat i establiment de connexions.
- Utilització de sockets per a la transmissió i recepció d'informació.
- Aplicacions client-servidor
- Programació d'aplicacions client i servidor.
- Utilització de fils en la programació d'aplicacions en xarxa.
- Comunicacions i programació multifil
- Utilització de tècniques de programació segura:
- Pràctiques de programació segura.
- Criptografia de clau pública i clau privada.
- Principals aplicacions de la criptografia
- Protocols criptogràfics.
- Política de seguretat.
- Programació de mecanismes de control d'accés.
- Encriptació d'informació.
- Protocols segurs de comunicacions.
- Programació d'aplicacions amb comunicacions segures.
- El llenguatge Javascript
- Instal·lació de nodeJS
- Execució d'scripts
- El gestor de paquets NPM
- Creació de paquets
- Execució de paquets
- Instal·lació de paquets
- El mòdul fs
- Concurrència en nodejs
- Asincronía
- Concurrència
- El bucle d'esdeveniments (event loop)
- L'objecte Process de nodejs
- El mòdul OS
- El mòdul Child Process

UD5. Serveis amb NodeJS

- Serveis en xarxa
- Protocols estàndard de comunicació a nivell d'aplicació (telnet, http, etc)
- Web Services
- Components
- Arquitectura
- Serveis REST
- Programació de servidors: La llibrería ExpressJS
- Middleware
- Endpoints
- Peticions GET/POST
- Autenticació basada en rols: JWT
- Microserveis
- Contenidors: Docker
- Docker Compose
- Desplegament de serveis

En ressum... què anem a vore en el mòdul?

La idea principal d'aquest mòdul és que coneixeu diferents mecanismes i tecnologies per crear aplicacions que oferisquen serveis i que treballen de forma concurrent. Se centrarà sobretot en la part de programació de servidors, però molts conceptes ens aprofitaran també per entendre com funcionen aplicacions del costat del client.

1.6 Temporalització de continguts

La temporalització de continguts prevista serà la següent:

1r Trimestre	2n Trimestre
UD1, UD2, UD3	UD4, UD5
Avaluació: 3 - 5 h.	*Avaluació: 3 - 5 h.*

1.7 Aspectes metodològics

1.7.1 Material Didàctic

El material didàctic del mòdul pot ressumir-se en el següent:

- Apunts proporcionats pel professor, en format web.
- Textos d'ampliació i enllaços a articles i documentació oficial relacionats amb cada unitat.
- Pràctiques i exercicis resolts per reforçar el que s'ha exposat als apunts.
- Vídeos explicatius d'aquells aspectes que requerisquen d'una explicació més visual.

Tot aquest material s'oferirà a través de l'aula virtual durant el desenvolupament de cada unitat.

A més, dins l'aula virtual, disposarem d'un fòrum general per comentar aspectes globals del mòdul, i un fòrum per cada unitat didàctica, per tal deresoldre dubtes i tractar aspectes relacionats amb la unitat.

1.7.2 Programari

El programari a utilitzar serà principalment lliure, i es donaran instruccions en cada unitat per a la seua descàrrega i instal·lació. A grans trets, el programari a utilitzar serà:

- **Kubuntu 24.04 o derivades.** Com a sistema operatiu de base a l'aula fem servir Kubuntu 24.04 LTS, tot i que per al modul, en principi qualsevol distribucio basada en Ubuntu 24.04 serà equivalent. No obstant això, l'alumne pot utilitzar qualsevol altre sistema operatiu, ja que tot el programari amb què treballarem és multiplataforma.
- **Visual Studio Code** com a editor de codi, que suporta diferents llenguatges, i és bastant ampliable amb extensions. Tot i que serà l'editor de referència, també s'utilitzaran IDEs com NetBeans i InteliJ.
- **Android Studio**, com a IDE per al desenvolupament de projectes per a Android.
- **Kotlin i NodeJS**, com a llenguatge de programació de servidors.

A més, cada unitat podrà requerir d'algunes eines més específiques, com editors de diagrames o analitzadors de codi, entre altres.

Eines web i col·laboratives

A banda del programari esmentat anteriorment, també s'utilitzaran el següent portals web i plataformes de treball col·laboratiu:

- **Portals Aules/Moodle**: Com a aula virtual, i que articularà el funcionament del mòdul. Serà aci on s'ubiquen els diferents recursos, fòrums, etc.
- **MS Teams**: Des del curs Des del 22/23, el nostre centre és *Centre Digital Col·laboratiu*, pel que cada alumne i professor disposa d'una identitat digital que li proporciona accés a tota la suite d'enies d'Office 365. Entre aquestes, la que utilitzarem de forma més freqüent serà *MS Teams*, a través de la qual s'organitzaran les diferents tutoríes col·lectives mitjançant videoconferència.
- **Github**: En alguns projectes en grup, serà de gran ajuda treballar amb sistemes de control de versions distribuits, com *Github* o *Gitlab*, de manera que puguen realitzar desenvolupaments de forma col·laborativa.

1.7.3 Metodologia. Què farem a classe?

La metodologia a utilitzar a l'aula pretén ser el més pràctica i realista possible, acostant la forma de treballar a l'aula a un entorn real de treball. Optarem per fer ús de metodologies actives, com l'aprenentatge basat en projectes o en reptes, on haureu de prendre un paper actiu. La ide aés potenciar l'aprenentatge per descobriment, significatiu, deductiu i el treball col·laboratiu i en equip.

Segons aquestes premisses, la metodología utilitzada al mòdul es regirà per les següents pautes:

- En iniciar cada unitat didàtica es realitzarà una presentació inicial d'aquesta, dels conceptes bàsics, i de què sereu capaços de fer en finalitzar-la.
- Disposareu de material per a la seua lectura comprensiva i estudi, així com de documentació addicional que es considere interessant,
- Disposareu d'exercicis i pràctiques guiades que acompanyen la teoría de la unitat i que ens ajudaran a entendre els conceptes de la unitat, fonamentant-se en coneixements previs, facilitant així l'aprenentatge per descobriment, significatiu i deductiu.
- Les session a l'aula tindran un caràcter fonamentalment pràctic, i s'aprofitaran exemples i casos pràctics per exposar els principals conceptes.
- Es fomentarà la realització de projectes i treballs en equip, simulant el treball real en una empresa, on cadascú tindrà un paper ben definit dins el grup. Per a això, es podran aplicar metodologíes àgils de desenvolupament, tipus *Scrum*, que implica una fase de planificació, el treball en parelles i la revisió periòdica mitjançant exposicions del treball realitzat en els equips.
- Es fomentarà la realització de projectes que relacionen continguts d'aquest amb altres mòduls, per tal que descobriu la relació i la importància de tots ells.
- Per a alguns continguts, es podran realitzar tallers d'aprofundoment o classes magistrals, on els conceptes a exposar siguen més densos i complexos, però sempre fonamentant-se en aspectes pràctics i sobre exemples reals.

1.8 Avaluació

Donat que es tracta d'ensenyaments presencials, l'avaluació serà continuada, i consistirà en el seguiment del treball que aneu realitzant, i tindrà en compte la vostra evolució en l'assoliment dels diferents Resultats d'Aprenentatge a partir dels Criteris d'Avaluació d'aquests.

Els diferents intruments d'avaluació podran ser:

- Exercicis d'avaluació, tant escrites com pràctiques,
- Tests i questionaris,
- Treballs individuals i en equip,
- Projectes en equip.

Cadascuna d'aquestes proves estarà guiada pels criteris d'avaluació i contribuirà als resultats d'aprenentatge.

La qualificació final de cada avaluació s'obtindrà a partir del grau de consecució dels resultats d'aprenentatge assolits, en relació als resultats d'aprenentatge tractats durant el trimestre.

Per tal d'aprovar cada mòdul, caldrà haver assolit **tots els resultats d'aprenentatge**

2. Exercicis

2.1 Exercicis de Kotlin variables i tipus de dades

2.1.1 1. Variables i Tipus de Dades

Exercici 1.1:

Enunciat: Declara una variable `var` que continga el número enter 10 i, després, canvia el seu valor a 20.

Solució

```
var numero = 10
numero = 20
println(numero)
```

Exercici 1.2:

Enunciat: Declara una variable `val` amb el valor de text "Hola" i intenta canviar el seu valor a "Adéu". Observa l'error que es genera.

Solució

```
val salutacio = "Hola"
// salutacio = "Adéu" // Error: Val cannot be reassigned
println(salutacio)
```

Exercici 1.3:

Enunciat: Declara una variable `var` amb el valor "Bon dia" i canvia-la a "Bona vesprada".

Solució

```
var frase = "Bon dia"
frase = "Bona vesprada"
println(frase)
```

2.1.2 2. Declaració explícita de tipus

Exercici 2.1:

Enunciat: Declara una variable de tipus `Int` amb el valor 25 explícitament.

Solució

```
val edat: Int = 25
println(edat)
```

Exercici 2.2:

Enunciat: Declara una variable de tipus `Double` amb el valor 3.14 explícitament.

Solució

```
val pi: Double = 3.14
println(pi)
```

Exercici 2.3:

Enunciat: Declara una variable de tipus `Boolean` amb el valor true.

Solució

```
val esVeritat: Boolean = true
println(esVeritat)
```

2.1.3 3. Inferència de tipus

Exercici 3.1:

Enunciat: Declara una variable amb el valor 100 sense especificar el tipus. Què inferix Kotlin?

Solució

```
val numero = 100
println(numero::class) // Class kotlin.Int
```

Exercici 3.2:

Enunciat: Declara una variable amb el valor 2.5 sense especificar el tipus. Què inferix Kotlin?

Solució

```
val decimal = 2.5
println(decimal::class) // Class kotlin.Double
```

Exercici 3.3:

Enunciat: Declara una variable amb el text "Kotlin" sense especificar el tipus. Què inferix Kotlin?

Solució

```
val llenguatge = "Kotlin"
println(llenguatge::class) // Class kotlin.String
```

2.1.4 4. Concatenació de cadenes

Exercici 4.1:

Enunciat: Declara dues variables amb els valors "Hola" i "Món" i concatena-les utilitzant l'operador +.

Solució

```
val part1 = "Hola"
val part2 = "Món"
val resultat = part1 + " " + part2
println(resultat) // Hola Món
```

Exercici 4.2:

Enunciat: Crea dues variables que contindran un nom i una edat, i concatena-les en una frase.

Solució

```
val nom = "Joan"
val edat = 30
val frase = "El meu nom és " + nom + " i tinc " + edat + " anys."
println(frase)
```

Exercici 4.3:

Enunciat: Declara una variable amb una frase i concatena una altra cadena que l'usuari introduïsca per teclat.

Solució

```
val fraseInicial = "Benvingut, "
print("Introduïx el teu nom: ")
val nom = readLine()
val fraseFinal = fraseInicial + nom
println(fraseFinal)
```

2.1.5 5. Plantilles de cadena

Exercici 5.1:

Enunciat: Declara una variable amb el nom "Maria" i utilitza-la dins d'una plantilla de cadena per a mostrar-la.

Solució

```
val nom = "Maria"
println("Hola, $nom!")
```

Exercici 5.2:

Enunciat: Declara dues variables numèriques, suma-les, i mostra el resultat utilitzant plantilles de cadena.

Solució

```
val num1 = 5
val num2 = 3
println("La suma de $num1 i $num2 és ${num1 + num2}.")
```

Exercici 5.3:

Enunciat: Utilitza una plantilla de cadena per a mostrar el valor d'una variable dins d'una frase.

Solució

```
val ciutat = "València"
println("Estic visitant $ciutat aquest cap de setmana.")
```

2.1.6 6. Null Safety

Exercici 6.1:

Enunciat: Declara una variable `String?` nullable i comprova si té un valor abans d'imprimir-la.

Solució

```
val nom: String? = null
if (nom != null) {
    println(nom)
} else {
    println("La variable és nul·la.")
}
```

Exercici 6.2:

Enunciat: Utilitza l'operador segur (`?.`) per a accedir a les propietats d'una variable nullable.

Solució

```
val nom: String? = "Joan"
println(nom?.length) // Mostra la longitud només si no és null
```

Exercici 6.3:

Enunciat: Proporciona un valor predeterminat per a una variable nullable utilitzant l'operador Elvis (`?:`).

Solució

```
val nom: String? = null
val valorPredeterminat = nom ?: "Anònim"
println(valorPredeterminat) // Mostra "Anònim"
```

2.1.7 7. Conversió de tipus

Exercici 7.1:

Enunciat: Declara una variable `Int` amb el valor 50, converteix-la a `String` i concatena-la amb un text.

Solució

```
val numero = 50
val text = numero.toString() + " és un número gran."
println(text)
```

Exercici 7.2:

Enunciat: Declara una variable `Double` amb el valor 8.7, converteix-la a `Int`, i mostra el resultat abans i després.

Solució

```
val decimal = 8.7
println("Valor original: $decimal")
val enter = decimal.toInt()
println("Convertit a enter: $enter")
```

Exercici 7.3:

Enunciat: Declara una variable `Char` amb el valor 'A' i converteix-la a número utilitzant `toInt()`.

Solució

```
val lletra = 'A'
val valorNumeric = lletra.toInt()
println(valorNumeric) // Mostra el valor numèric corresponent al caràcter 'A'
```

2.1.8 8. Treballant amb el tipus `Char`

Exercici 8.1:

Enunciat: Declara una variable `Char` amb el valor 'B' i mostra el seu valor numèric utilitzant `toInt()`.

Solució

```
val lletra = 'B'
println(lletra.toInt()) // Mostra el valor numèric de 'B'
```

Exercici 8.2:

Enunciat: Crea una variable `Char` a partir d'un valor numèric utilitzant `toChar()`.

Solució

```
val numero = 67
val lletra = numero.toChar()
println(lletra) // Mostra el caràcter corresponent al número 67
```

Exercici 8.3:

Enunciat: Declara una variable de tipus `Char` i converteix-la a majúscula utilitzant una funció integrada.

Solució

```
val lletra = 'b'
println(lletra.uppercaseChar()) // Converteix a majúscula
```

2.1.9 9. Operacions aritmètiques

Exercici 9.1:

Enunciat: Declara dues variables `var` de tipus `Int`, realitza una suma i mostra el resultat.

Solució

```
var num1 = 7
var num2 = 5
val suma = num1 + num2
println("La suma és $suma")
```

Exercici 9.2:

Enunciat: Declara dues variables `Double`, realitza una divisió i mostra el resultat.

Solució

```
val num1 = 9.0
val num2 = 3.0
val divisio = num1 / num2
println("La divisió és $divisio")
```

Exercici 9.3:

Enunciat: Declara dues variables, realitza una resta i una multiplicació, i mostra els resultats.

Solució

```
val num1 = 10
val num2 = 4
val resta = num1 - num2
val multiplicacio = num1 * num2
println("La resta és $resta")
println("La multiplicació és $multiplicacio")
```

2.2 2. Fluxe de control

2.2.1 Exercici 1: Conversió de temperatures

Enunciat: Escriu un programa que demane a l'usuari una temperatura en graus Celsius i la convertisca a Fahrenheit. La fórmula per a la conversió és: `F = C * 9/5 + 32`.

Solució:

Solució

```
fun main() {
    print("Introdueix la temperatura en graus Celsius: ")
    val celsius = readLine()!!.toDouble()
    val fahrenheit = (celsius * 9/5) + 32
    println("La temperatura en Fahrenheit és: $fahrenheit")
}
```

2.2.2 Exercici 2: Número més gran

Enunciat: Crea un programa que demane a l'usuari tres números enters i determine quin és el més gran.

Solució:

Solució

```
fun main() {
    print("Introdueix tres números: ")
    val num1 = readLine()!!.toInt()
    val num2 = readLine()!!.toInt()
    val num3 = readLine()!!.toInt()
    val major = if (num1 > num2 && num1 > num3) num1 else if (num2 > num3) num2 else num3
    println("El número més gran és: $major")
}
```

2.2.3 Exercici 3: Generador de contrasenyes aleatòries

Enunciat: Escriu un programa que genere una contrasenya aleatòria de 8 caràcters, que continga lletres majúscules, minúscules i números.

Solució:

Solució

```
import kotlin.random.Random

fun main() {
    val charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
    val password = (1..8)
        .map { charset.random() }
        .joinToString("")
    println("Contrasenya generada: $password")
}
```

2.2.4 Exercici 4: Calculadora de l'IMC

Enunciat: Crea un programa que calcule l'Índex de Massa Corporal (IMC) en base al pes i l'altura introduïts per l'usuari. La fórmula és: `IMC = pes / altura^2`.

Solució:

Solució

```
fun main() {
    print("Introdueix el pes (kg): ")
    val pes = readLine()!!.toDouble()
    print("Introdueix l'altura (m): ")
    val altura = readLine()!!.toDouble()
    val imc = pes / (altura * altura)
    println("El teu IMC és: %.2f".format(imc))
}
```

2.2.5 Exercici 5: Taula de multiplicar

Enunciat: Fes un programa que demane a l'usuari un número i mostre la seua taula de multiplicar fins al 10.

Solució:

Solució

```
fun main() {
    print("Introdueix un número: ")
    val num = readLine()!!.toInt()
    for (i in 1..10) {
        println("$num x $i = ${num * i}")
    }
}
```

2.2.6 Exercici 6: Adivina el número

Enunciat: Crea un joc on l'ordinador trie un número aleatori entre 1 i 100, i l'usuari haja d'adivinar-lo. L'ordinador donarà pistes si el número introduït és massa alt o massa baix.

Solució:

Solució

```
import kotlin.random.Random

fun main() {
    val secret = Random.nextInt(1, 101)
    var guessed = false
    while (!guessed) {
        print("Adivina el número (1-100): ")
        val guess = readLine()!!.toInt()
        if (guess == secret) {
            println("Enhorabona! Has encertat.")
            guessed = true
        } else if (guess < secret) {
            println("Massa baix!")
        } else {
            println("Massa alt!")
        }
    }
}
```

2.2.7 Exercici 7: Comptador de vocals

Enunciat: Escriu un programa que demane una frase a l'usuari i compte quantes vocals conté.

Solució:

Solució

```
fun main() {
    print("Introdueix una frase: ")
    val frase = readLine()!!.toLowerCase()
    val vocals = "aeiou"
    var comptador = 0
    for (caracter in frase) {
        if (vocals.contains(caracter)) comptador++
    }
    println("La frase conté $comptador vocals.")
}
```

2.2.8 Exercici 8: Nombre perfecte

Enunciat: Fes un programa que determine si un número donat per l'usuari és un nombre perfecte (un nombre perfecte és aquell la suma dels seus divisors, excepte ell mateix, és igual al mateix nombre).

Solució:

Solució

```
fun main() {
    print("Introdueix un número: ")
    val num = readLine()!!.toInt()
    var suma = 0
    for (i in 1 until num) {
        if (num % i == 0) suma += i
    }
    if (suma == num) {
        println("$num és un nombre perfecte.")
    } else {
        println("$num no és un nombre perfecte.")
    }
}
```

2.2.9 Exercici 9: Conversió d'hores a minuts i segons

Enunciat: Crea un programa que convertisca una quantitat d'hores donada per l'usuari en minuts i segons.

Solució:

Solució

```
fun main() {
    print("Introdueix el número d'hores: ")
    val hores = readLine()!!.toInt()
    val minuts = hores * 60
    val segons = minuts * 60
    println("$hores hores són $minuts minuts i $segons segons.")
}
```

2.2.10 Exercici 10: FizzBuzz personalitzat

Enunciat: Escriu un programa que recórrega els números de l'1 al 50 i mostre "Fizz" si el número és múltiple de 3, "Buzz" si és múltiple de 5, i "FizzBuzz" si és múltiple de tots dos. A més, personalitza el programa perquè mostre un altre missatge per als múltiples de 7.

Solució:

Solució

```
fun main() {
    for (i in 1..50) {
        when {
            i % 3 == 0 && i % 5 == 0 -> println("FizzBuzz")
            i % 3 == 0 -> println("Fizz")
            i % 5 == 0 -> println("Buzz")
            i % 7 == 0 -> println("Boom")
            else -> println(i)
        }
    }
}
```

2.3 3. Funcions

2.3.1 Exercici 1: Funció de suma amb valor per defecte

Enunciat: Escriu una funció que accepte dos números i retorne la seua suma. Fes que un dels números tinga un valor per defecte.

Solució

```
fun suma(a: Int = 5, b: Int): Int {
    return a + b
}

fun main() {
    println(suma(b = 10)) // Output: 15
}
```

2.3.2 Exercici 2: Funció que retorna múltiples valors

Enunciat: Crea una funció que retorne un parell d'elements: el quocient i el residu de dividir dos números.

Solució

```
fun divisio(dividend: Int, divisor: Int): Pair<Int, Int> {
    return Pair(dividend / divisor, dividend % divisor)
}

fun main() {
    val (quocient, residu) = divisio(10, 3)
    println("Quocient: $quocient, Residu: $residu")
}
```

2.3.3 Exercici 3: Filtratge de llista amb lambda

Enunciat: Escriu una funció que filtre una llista d'enters i retorne només els parells, utilitzant una expressió lambda.

Solució

```
fun filtraParells(nums: List<Int>): List<Int> {
    return nums.filter { it % 2 == 0 }
}

fun main() {
    println(filtraParells(listOf(1, 2, 3, 4, 5, 6))) // Output: [2, 4, 6]
}
```

2.3.4 Exercici 4: Concatenació de cadenes

Enunciat: Crea una funció que accepte una quantitat variable de cadenes i les concatene en una sola cadena.

Solució

```
fun concatena(vararg cadenes: String): String {
    return cadenes.joinToString(" ")
}

fun main() {
    println(concatena("Hola", "a", "tots!")) // Output: "Hola a tots!"
}
```

2.3.5 Exercici 5: Funció d'ordre superior

Enunciat: Escriu una funció que accepte un altre funció com a paràmetre i aplique aquesta funció a dos nombres enters.

Solució

```
fun operacio(x: Int, y: Int, funcio: (Int, Int) -> Int): Int {
    return funcio(x, y)
}

fun main() {
    val suma = operacio(3, 4, { a, b -> a + b })
    println(suma) // Output: 7
}
```

2.3.6 Exercici 6: Lambda amb llista de preus

Enunciat: Escriu una funció que, donada una llista de preus, aplique un descompte del 10% a cadascun dels elements, utilitzant una expressió lambda.

Solució

```
fun aplicarDescompte(preus: List<Double>): List<Double> {
    return preus.map { it * 0.9 }
}

fun main() {
    println(aplicarDescompte(listOf(100.0, 200.0, 300.0))) // Output: [90.0, 180.0, 270.0]
}
```

2.3.7 Exercici 7: Funció anònima

Enunciat: Crea una funció que trie el número més gran entre tres valors, utilitzant una funció anònima.

Solució

```
fun majorDeTres(a: Int, b: Int, c: Int): Int {
    return fun(): Int {
        return if (a > b && a > c) a else if (b > c) b else c
    }()
}

fun main() {
    println(majorDeTres(3, 7, 5)) // Output: 7
}
```

2.3.8 Exercici 8: Filtres múltiples amb lambda

Enunciat: Escriu una funció que filtre una llista de cadenes, retornant només aquelles que tinguen més de 5 caràcters, utilitzant expressions lambda.

Solució

```
fun filtraCadenes(llista: List<String>): List<String> {
    return llista.filter { it.length > 5 }
}

fun main() {
    println(filtraCadenes(listOf("Hola", "Programació", "Kotlin", "Lambda"))) // Output: ["Programació", "Lambda"]
}
```

2.3.9 Exercici 9: Conversió de temperatures amb funcions d'extensió

Enunciat: Crea una funció d'extensió per a la classe `Double` que convertisca temperatures de Celsius a Fahrenheit.

Solució

```
fun Double.aFahrenheit(): Double {
    return this * 9 / 5 + 32
}

fun main() {
    println(25.0.aFahrenheit()) // Output: 77.0
}
```

2.3.10 Exercici 10: Funció recursiva

Enunciat: Escriu una funció recursiva que calcule el factorial d'un número donat.

Solució

```
fun factorial(n: Int): Int {
    return if (n == 0) 1 else n * factorial(n - 1)
}

fun main() {
    println(factorial(5)) // Output: 120
}
```

2.4 4. POO

2.4.1 Exercici 1: Creació d'una classe Persona

Enunciat: Escriu una classe `Persona` amb propietats per al nom, l'edat i el gènere. Afig un mètode per a mostrar la informació de la persona.

Solució

```
class Persona(val nom: String, var edat: Int, val genere: String) {
    fun mostrarInfo() {
        println("Nom: $nom, Edat: $edat, Gènere: $genere")
    }
}

fun main() {
    val persona = Persona("Anna", 25, "Femení")
    persona.mostrarInfo()
}
```

2.4.2 Exercici 2: Constructor secundari

Enunciat: Crea una classe `Cotxe` amb un constructor primari que accepte el model i l'any. Afig un constructor secundari que permeta crear un cotxe sense especificar l'any.

Solució

```
class Cotxe(val model: String, val any: Int) {
    constructor(model: String) : this(model, 2024)
    fun mostrarInfo() {
        println("Model: $model, Any: $any")
    }
}

fun main() {
    val cotxe1 = Cotxe("Toyota", 2020)
    val cotxe2 = Cotxe("Honda")
    cotxe1.mostrarInfo()
    cotxe2.mostrarInfo()
}
```

2.4.3 Exercici 3: Herència en classes

Enunciat: Crea una classe base `Animal` amb un mètode que faça un soroll. Crea dues subclasses, `Gos` i `Gat`, que sobreescriguen el mètode per a fer el seu propi soroll.

Solució

```
open class Animal {
    open fun soroll() {
        println("L'animal fa soroll")
    }
}

class Gos : Animal() {
    override fun soroll() {
        println("El gos borda")
    }
}

class Gat : Animal() {
    override fun soroll() {
        println("El gat miola")
    }
}

fun main() {
    val gos = Gos()
    val gat = Gat()
    gos.soroll()
    gat.soroll()
}
```

2.4.4 Exercici 4: Getters i setters personalitzats

Enunciat: Crea una classe `CompteBancari` amb un saldo que no pot ser negatiu. Utilitza getters i setters personalitzats per a assegurar-te que no es pot retirar més diners dels que hi ha.

Solució

```
class CompteBancari(private var saldo: Double) {
    var saldoDisponible: Double
        get() = saldo
        set(value) {
            if (value >= 0) saldo = value else println("No pots tenir saldo negatiu!")
        }

    fun retirar(diners: Double) {
        if (diners <= saldo) saldo -= diners else println("No tens prou diners!")
    }
}

fun main() {
    val compte = CompteBancari(100.0)
    compte.retirar(50.0)
    println("Saldo actual: ${compte.saldoDisponible}")
    compte.saldoDisponible = -10.0
}
```

2.4.5 Exercici 5: Objecte Singleton

Enunciat: Crea un objecte `Calculadora` amb funcions estàtiques per a sumar, restar, multiplicar i dividir.

Solució

```
object Calculadora {
    fun sumar(a: Int, b: Int) = a + b
    fun restar(a: Int, b: Int) = a - b
    fun multiplicar(a: Int, b: Int) = a * b
    fun dividir(a: Int, b: Int) = a / b
}

fun main() {
    println(Calculadora.sumar(10, 5))
    println(Calculadora.dividir(10, 2))
}
```

2.4.6 Exercici 6: Classes abstractes

Enunciat: Defineix una classe abstracta `Forma` amb un mètode abstracte `area()`. Crea subclasses `Cercle` i `Quadrat` que implementen aquest mètode.

Solució

```
abstract class Forma {
    abstract fun area(): Double
}

class Cercle(val radi: Double) : Forma() {
    override fun area(): Double = Math.PI * radi * radi
}

class Quadrat(val costat: Double) : Forma() {
    override fun area(): Double = costat * costat
}

fun main() {
    val cercle = Cercle(5.0)
    val quadrat = Quadrat(4.0)
    println("Àrea del cercle: ${cercle.area()}")
    println("Àrea del quadrat: ${quadrat.area()}")
}
```

2.4.7 Exercici 7: Interfícies

Enunciat: Crea una interfície `Volar` amb un mètode `vola()`. Fes que una classe `Avió` i una classe `Ocell` implementen aquesta interfície.

Solució

```
interface Volar {
    fun vola()
}

class Avio : Volar {
    override fun vola() {
        println("L'avió vola alt!")
    }
}

class Ocell : Volar {
    override fun vola() {
        println("L'ocell vola pel cel!")
    }
}

fun main() {
    val avio = Avio()
    val ocell = Ocell()
    avio.vola()
    ocell.vola()
}
```

2.4.8 Exercici 8: Polimorfisme

Enunciat: Crea una classe `Vehicle` amb un mètode `moure()`. Crea subclasses `Bicicleta` i `Cotxe` que sobreescriguen aquest mètode. Crea una funció que accepte un `Vehicle` i cride al seu mètode `moure()`.

Solució

```
open class Vehicle {
    open fun moure() {
        println("El vehicle es mou")
    }
}

class Bicicleta : Vehicle() {
    override fun moure() {
        println("La bicicleta roda")
    }
}

class Cotxe : Vehicle() {
    override fun moure() {
        println("El cotxe circula")
    }
}

fun movimentVehicle(vehicle: Vehicle) {
    vehicle.moure()
}

fun main() {
    val bicicleta = Bicicleta()
    val cotxe = Cotxe()
    movimentVehicle(bicicleta)
    movimentVehicle(cotxe)
}
```

2.4.9 Exercici 9: Funcions d'àmbit

Enunciat: Crea una classe `Llibre` amb propietats títol i autor. Utilitza funcions d'àmbit com `apply` i `run` per a inicialitzar un llibre i mostrar la seua informació.

Solució

```
class Llibre(val titol: String, val autor: String)

fun main() {
    val llibre = Llibre("El Quixot", "Cervantes").apply {
        println("Llibre: $titol, Autor: $autor")
    }

    llibre.run {
        println("Lectura del llibre: $titol per $autor")
    }
}
```

2.4.10 Exercici 10: Companion object

Enunciat: Crea una classe `Matemàtiques` amb un `companion object` que continga una funció per a calcular el factorial d'un número.

Solució

```
class Matematiques {
    companion object {
        fun factorial(n: Int): Int {
            return if (n == 0) 1 else n * factorial(n - 1)
        }
    }
}

fun main() {
    println(Matematiques.factorial(5)) // Output: 120
}
```

2.5 5. EDD

2.5.1 Exercici 1: Creació d'un array de números enters

Enunciat: Escriu un programa que cree un array de 5 números enters i després mostre la suma de tots els elements.

????? info "Solució"

```
fun main() {
    val numeros = arrayOf(1, 2, 3, 4, 5)
    val suma = numeros.sum()
    println("La suma és: $suma")
}
```

2.5.2 Exercici 2: Accés a elements d'un array

Enunciat: Crea un array de cadenes amb els noms de la setmana. Mostra el nom del tercer dia utilitzant tant l'operador `[]` com la funció `get()`.

????? info "Solució"

```
fun main() {
    val dies = arrayOf("Dilluns", "Dimarts", "Dimecres", "Dijous", "Divendres")
    println(dies[2]) // Amb []
    println(dies.get(2)) // Amb get()
}
```

2.5.3 Exercici 3: Llista immutable de números

Enunciat: Crea una llista immutable de 5 números enters. Imprimeix la suma dels elements i indica si conté el número 10.

????? info "Solució"

```
fun main() {
    val numeros = listOf(1, 2, 3, 4, 5)
    val suma = numeros.sum()
    println("Suma: $suma")
    println("Conté 10: ${numeros.contains(10)}")
}
```

2.5.4 Exercici 4: Llista mutable de cadenes

Enunciat: Crea una llista mutable de tres cadenes. Afig una quarta cadena i imprimeix el contingut de la llista.

????? info "Solució"

```
fun main() {
    val llista = mutableListOf("Cadena 1", "Cadena 2", "Cadena 3")
    llista.add("Cadena 4")
    println(llista)
}
```

2.5.5 Exercici 5: Funcions sobre llistes

Enunciat: Escriu un programa que filtre els números parells d'una llista immutable de 10 elements.

????? info "Solució"

```
fun main() {
    val numeros = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    val parells = numeros.filter { it % 2 == 0 }
    println(parells)
}
```

2.5.6 Exercici 6: Conjunt immutable

Enunciat: Crea un conjunt immutable que continga alguns números. Mostra si el conjunt conté el número 5.

????? info "Solució"

```
fun main() {
    val conjunt = setOf(1, 2, 3, 4, 5)
    println("Conté 5: ${conjunt.contains(5)}")
}
```

2.5.7 Exercici 7: Mapa immutable

Enunciat: Crea un mapa immutable amb tres parells clau-valor on les claus siguen números i els valors cadenes. Imprimeix el valor associat a la clau 2.

????? info "Solució"

```
fun main() {
    val mapa = mapOf(1 to "Un", 2 to "Dos", 3 to "Tres")
    println("Valor per a clau 2: ${mapa[2]}")
}
```

2.5.8 Exercici 8: Recorregut d'un array amb índexs

Enunciat: Escriu un programa que recórrega un array d'enters utilitzant `withIndex()` i mostre tant l'índex com el valor de cada element.

????? info "Solució"

```
fun main() {
    val numeros = arrayOf(10, 20, 30, 40)
    for ((index, valor) in numeros.withIndex()) {
        println("Índex: $index, Valor: $valor")
    }
}
```

2.5.9 Exercici 9: Operacions amb llistes mutables

Enunciat: Crea una llista mutable de números. Elimina l'últim element de la llista i imprimeix la nova llista.

????? info "Solució"

```
fun main() {
    val llista = mutableListOf(1, 2, 3, 4, 5)
    llista.removeAt(llista.size - 1)
    println(llista)
}
```

2.5.10 Exercici 10: Mapa mutable

Enunciat: Crea un mapa mutable amb tres parells clau-valor. Afig un nou element al mapa i imprimeix tot el contingut.

????? info "Solució" `kotlin`

```
fun main() {
    val mapa = mutableMapOf(1 to "Un", 2 to "Dos", 3 to "Tres")
    mapa[4] = "Quatre"
    println(mapa)
}
```