Test de positionnement

Android Jetpack Compose M1

NOM Prénom _____

Contexte

Ce test permet d'évaluer votre niveau avant le module **Android Jetpack Compose**.

Il ne s'agit pas d'un contrôle noté de connaissances avancées, mais d'un diagnostic. L'objectif est d'identifier les notions déjà maîtrisées et celles qui devront être renforcées pendant le cours.

Durée conseillée : 30 à 40 minutes.

Consignes

Pour chaque question, **une** ou **plusieurs réponses** peuvent être correctes.

Lisez attentivement les propositions avant de répondre.

Partie 1 – Kotlin

Question 1 – val et var

En Kotlin, quelles affirmations sont correctes ?

A. Une variable déclarée avec `val` peut être réaffectée après son initialisation.
B. Une variable déclarée avec `var` peut être réaffectée après son initialisation.
C. `val` signifie que l'objet référencé est toujours totalement immuable.
D. `val` empêche de changer la référence stockée dans la variable.
E. `var` ne peut être utilisé que pour les types numériques.

Réponses correctes : _____

Question 2 – Null safety

Quel code Kotlin est correct ou sûr concernant les valeurs nullables ?

A. `val name: String? = null`
B. `val length = name.length` si **name** est de type **String?**
C. `val length = name?.length`
D. `val length = name!!.length` est toujours sans risque.
E. `val displayName = name ?: "Utilisateur inconnu"`

Réponses correctes : _____

Question 3 – Classes et data class

À propos des data class en Kotlin, quelles affirmations sont correctes ?

A. Une **data class** ne peut contenir qu'une seule propriété.
B. Une **data class** génère automatiquement des méthodes utiles comme `toString()`, `equals()` et `copy()`.
C. Une **data class** est souvent utilisée pour représenter des données comme un produit, un utilisateur ou une réponse d'API.
D. Une **data class** remplace systématiquement une **interface**.
E. Une **data class** ne peut pas être utilisée avec **Retrofit** ou **Room**.

Réponses correctes : _____

Question 4 – Collections

Que retourne le code suivant dans "result" ?

```
val prices = listOf(10, 20, 30)
val result = prices.map { it * 2 }
```

A. Une liste vide
B. Une liste contenant 20, 40, 60
C. Une liste contenant 10, 20, 30, 10, 20, 30
D. Un seul entier valant 60

Réponses correctes : _____

Question 5 – Lambdas

Quelles affirmations sont correctes à propos des lambdas en Kotlin ?

A. Une **lambda** peut être passée en paramètre à une fonction.
B. Une **lambda** doit obligatoirement avoir un nom.
C. Une **lambda** ne peut jamais retourner de valeur.
D. Dans une **lambda** avec un seul paramètre, on peut souvent utiliser `it`.
E. Les **lambdas** sont interdites dans les applications Android.

Réponses correctes : _____

Question 6 – when

Quelles affirmations sont correctes concernant when en Kotlin ?

A. `when` est strictement identique à `if`, sans différence d'usage.
B. `when` ne peut tester que des chaînes de caractères.
C. `when` peut être utilisé comme expression et retourner une valeur.
D. `when` est réservé aux applications Android.
E. `else` permet de gérer les cas non prévus.

Réponses correctes : _____

Question 7 – Fonctions avec paramètres par défaut

Que permet Kotlin avec les paramètres par défaut ?

A. D'appeler une fonction sans fournir certains arguments si une valeur par défaut existe.
B. De supprimer totalement le typage des paramètres.

C. D'améliorer la lisibilité de certaines fonctions.
D. D'utiliser des arguments nommés lors de l'appel.
E. D'empêcher toute surcharge de fonction.

Réponses correctes : _____

Question 8 – sealed class et états

Dans une application Android, pourquoi peut-on utiliser une** sealed class **ou une** sealed interface **?

A. Pour représenter plusieurs états possibles d'un écran, comme *Loading*, *Success* et *Error*.
B. Pour remplacer tous les `ViewModel`.
C. Pour limiter les sous-types possibles à un ensemble connu.
D. Pour créer automatiquement une base de données **Room**.
E. Pour faciliter la gestion exhaustive des cas dans un `when`.

Réponses correctes : _____

Partie 2 – Android général

Question 9 – Activity

Quelle affirmation décrit correctement une Activity Android ?

A. Une **Activity** représente généralement un point d'entrée visuel ou un écran de l'application.
B. Une **Activity** sert uniquement à stocker des données en base locale.
C. Une application Android ne peut contenir qu'une seule **Activity**.
D. Une **Activity** possède un cycle de vie géré par le système Android.

Réponses correctes : _____

Question 10 – Cycle de vie

Quelles méthodes font partie du cycle de vie classique d'une Activity ?

A. `onCreate()`
B. `onResume()`
C. `onDestroy()`
D. `onDatabaseCreated()`
E. `onPause()`

Réponses correctes : _____

Question 11 – Ressources Android

Pourquoi utilise-t-on des fichiers de ressources comme strings.xml ?

A. Pour centraliser les chaînes de caractères.
B. Pour faciliter la traduction de l'application.
C. Pour éviter d'écrire du Kotlin.

D. Pour séparer certaines données d'affichage du code source.
E. Pour stocker les mots de passe de production.

Réponses correctes : _____

Question 12 – Permissions

Quelles affirmations sont correctes concernant les permissions Android ?

A. Certaines permissions doivent être déclarées dans le fichier `AndroidManifest.xml`.
B. Toutes les permissions sont automatiquement accordées à l'installation.
C. Certaines permissions dites dangereuses doivent être demandées à l'exécution.
D. L'accès Internet nécessite généralement une permission déclarée dans le manifeste.
E. Les permissions ne concernent que les applications écrites en Java.

Réponses correctes : _____

Question 13 – Android Studio et Gradle

Quel est le rôle principal de Gradle dans un projet Android ?

A. Compiler et construire le projet.
B. Gérer les dépendances.
C. Écrire automatiquement toute l'interface utilisateur.
D. Configurer certains paramètres de *build* comme le SDK minimum.
E. Remplacer le langage Kotlin.

Réponses correctes : _____

Partie 3 – Jetpack Compose

Question 14 – Fonction composable

Quelles affirmations sont correctes à propos d'une fonction annotée** @Composable **?

A. Elle peut décrire une partie de l'interface utilisateur.
B. Elle doit obligatoirement retourner une **View**.
C. Elle peut appeler d'autres fonctions composables.
D. Elle est utilisée par **Compose** pour construire l'UI.
E. Elle remplace automatiquement tous les fichiers Kotlin du projet.

Réponses correctes : _____

Question 15 – UI déclarative

Quelle phrase résume le mieux l'approche déclarative de Compose ?

A. Le développeur récupère chaque vue avec `findViewById()` puis la modifie.
B. L'interface est décrite en fonction d'un état.
C. Le XML reste obligatoire pour tous les écrans.
D. Compose met à jour l'interface quand les données observées changent.

Réponses correctes : _____

Question 16 – Recomposition

Quelles affirmations sont correctes concernant la recomposition ?

A. La recomposition peut se produire lorsqu'un état lu par un composable change.
B. La recomposition signifie que toute l'application est toujours reconstruite depuis zéro.
C. Il faut éviter les opérations lourdes directement dans le corps d'un composable.
D. Une fonction composable peut être appelée plusieurs fois au cours de la vie de l'écran.
E. La recomposition n'existe que dans les applications Java/XML.

Réponses correctes : _____

Question 17 – Modifier

À quoi sert « Modifier » en Jetpack Compose ?

A. À modifier l'apparence d'un composant.
B. À modifier la disposition d'un composant.
C. À déclarer une permission Android.
D. À gérer certaines interactions comme le clic ou le scroll.
E. À remplacer le fichier `AndroidManifest.xml`.

Réponses correctes : _____

Question 18 – Ordre des modifiers

Que faut-il retenir à propos de l'ordre des modifiers ?

A. L'ordre peut influencer le rendu final.
B. `Modifier.padding().background()` et `Modifier.background().padding()` peuvent produire des résultats visuels différents.
C. L'ordre des *modifiers* est toujours ignoré par **Compose**.
D. Les *modifiers* sont lus dans l'ordre où ils sont chaînés.

Réponses correctes : _____

Question 19 – remember et rememberSaveable

Quelles affirmations sont correctes ?

A. `remember` permet de conserver une valeur pendant les recompositions.
B. `remember` conserve toujours l'état après une rotation d'écran.
C. `rememberSaveable` peut conserver certains états lors d'une recréation d'activité, par exemple une rotation.
D. `rememberSaveable` est utile pour certains états simples d'interface.
E. `remember` remplace un `ViewModel` dans tous les cas.

Réponses correctes : _____

Question 20 – State hoisting

Le state hoisting consiste à :

A. Déplacer un état vers un composable parent.
B. Rendre un composable enfant plus réutilisable.

C. Forcer tous les états à être stockés dans **MainActivity**.
D. Faire descendre les données et remonter les événements.
E. Supprimer les paramètres des fonctions composables.

Réponses correctes : _____

Question 21 – Scaffold
À quoi sert principalement Scaffold en Compose Material 3 ?

A. À structurer un écran avec des zones comme `topBar`, `bottomBar`, `floatingActionButton` et contenu principal.
B. À effectuer automatiquement les appels API.
C. À fournir un squelette d'interface respectant les conventions **Material**.
D. À gérer directement une base de données **Room**.
E. À transmettre des valeurs de *padding* au contenu principal.

Réponses correctes : _____

Partie 4 – Architecture

Question 22 – MVVM
Dans une architecture MVVM Android, quelles associations sont correctes ?

A. La **View** affiche l'interface et transmet les événements utilisateur.
B. Le **ViewModel** expose l'état de l'écran et gère la logique de présentation.
C. Le **Model** ou la couche data représente les données et leur accès.
D. La **View** doit contenir directement toute la logique réseau.
E. Le **ViewModel** doit idéalement éviter de dépendre directement d'une **Activity**.

Réponses correctes : _____

Question 23 – ViewModel
Pourquoi utilise-t-on un ViewModel dans une application Android moderne ?

A. Pour conserver et exposer l'état d'un écran.
B. Pour survivre à certaines recréations de configuration, comme une rotation d'écran.
C. Pour remplacer entièrement **Retrofit**.
D. Pour isoler une partie de la logique de l'UI.
E. Pour dessiner directement les pixels à l'écran.

Réponses correctes : _____

Question 24 – Repository Pattern
Quel est l'intérêt du Repository Pattern ?

A. Fournir une interface claire entre le **ViewModel** et les sources de données.
B. Masquer l'origine exacte des données : API, base locale, cache, etc.
C. Remplacer tous les composables.

D. Faciliter les tests en isolant l'accès aux données.
E. Gérer uniquement les animations **Compose**.

Réponses correctes : _____

Question 25 – Navigation Compose

Quelles affirmations sont correctes à propos de la navigation dans une application Compose ?

A. **NavHost** permet de déclarer les destinations de navigation.
B. Une route peut représenter un écran.
C. Navigation Compose empêche d'utiliser plusieurs écrans.
D. Il est possible de passer des arguments entre destinations.
E. La navigation doit obligatoirement être codée uniquement en XML.

Réponses correctes : _____

Partie 5 – API / données / persistance

Question 26 – Retrofit

À quoi sert Retrofit dans une application Android ?

A. À consommer des API HTTP/REST.
B. À décrire des endpoints avec des interfaces Kotlin.
C. À remplacer automatiquement tous les **ViewModel**.
D. À convertir des réponses réseau en objets Kotlin avec l'aide d'un convertisseur adapté.
E. À créer directement des tables **Room**.

Réponses correctes : _____

Question 27 – Room

Quelles affirmations sont correctes concernant Room ?

A. **Room** est une bibliothèque de persistance locale.
B. Une **Entity** représente généralement une table.
C. Un **DAO** contient les méthodes d'accès aux données.
D. **Room** est utilisé uniquement pour afficher des boutons **Compose**.
E. **Room** peut être utile pour stocker des favoris localement.

Réponses correctes : _____

Question 28 – Coroutines, Flow et StateFlow

Quelles affirmations sont correctes ?

A. Une fonction suspend peut être suspendue sans bloquer le *thread* courant.
B. **viewModelScope** est souvent utilisé pour lancer des coroutines liées au **ViewModel**.
C. **Flow** permet d'émettre plusieurs valeurs dans le temps.

D. **StateFlow** est souvent utilisé pour exposer un état observable depuis un **ViewModel**.
E. Les coroutines ne peuvent pas être utilisées avec Android.

Réponses correctes : _____

Partie 6 – Qualité de code / debugging / bonnes pratiques

Question 29 – Bonnes pratiques Compose

Quelles pratiques sont recommandées dans un projet Compose ?

A. Créer des composables petits et réutilisables.
B. Placer systématiquement toute la logique métier dans les composables.
C. Passer un Modifier optionnel aux composables réutilisables.
D. Séparer autant que possible l'affichage, l'état et l'accès aux données.
E. Éviter de faire des opérations coûteuses directement dans le corps d'un composable.

Réponses correctes : _____

Question 30 – Debugging Android

Quels outils ou pratiques peuvent aider à diagnostiquer un problème dans une application Android ?

A. Lire les messages dans **Logcat**.
B. Utiliser le debugger avec des points d'arrêt.
C. Vérifier la **stack trace** lors d'un crash.
D. Ignorer systématiquement les erreurs Gradle.
E. Tester progressivement les fonctionnalités plutôt que tout développer avant de lancer l'application.

Réponses correctes : _____